Fast, minimal and oblivious routing algorithms on the mesh with bounded queues

Ami Litman *  
Shiri Moran-Schein †

Abstract

This paper studies fast, deterministic, permutation routing algorithms with bounded queues on the \( n \times n \) mesh. Our main result is an \( O(n) \)-step, strongly-dimensional (and thus also source-oblivious and minimal) permutation routing algorithm. This algorithm works under a relaxed model in which nodes can freely send data to their neighbors.

In a more prevalent model, the standard model, data may be sent only when accompanied by a packet. Under this model we present the following two algorithms: An \( O(n \log n) \)-step strongly-dimensional algorithm and an \( O(n) \)-step oblivious and weakly-dimensional (and thus also minimal) algorithm.

As said, all these algorithms store only \( O(1) \) packets in a node. Moreover, they use only \( O(\log n) \) state bits in a node and transfer only \( O(\log n) \) data bits on an edge in a step.

All our algorithms are based on the following technique of open-loop flow control. An algorithm is composed of two stages: setup and transportation.

The setup stage computes certain values and stores them in the network. In particular, it computes a rational number \( o(\alpha) \) for certain critical edges \( \epsilon \).

The transportation stage moves the packets to their destinations. It uses the computed values to slow the packets so that the traffic on each critical edge \( \epsilon \) is bounded by \( o(\alpha) \); that is, at most \( o(\alpha) \cdot l \) packets traverse \( \epsilon \) during any \( l \) consecutive steps.

This bound on the burstiness of the traffic enables the algorithm to avoid hot spots and maintain bounded queues. The algorithm achieves this by an open-loop control; that is, during this stage no information is transferred in a direction opposite to that of the packets.

An additional novelty of our algorithms is the application of a dynamic routing problem to solve a static one. The dynamic problem in question seems easy, as its network is just a linear array. We show, however, that this problem is unsolvable by the techniques of Adversarial Queuing Theory.

1 Introduction

The mesh is a very attractive network due to its simplicity and economical use of space. The search for fast and practical deterministic routing algorithm for this network is still on. This search mainly concerns the canonical packet routing problem – the (Partial) Permutation-Routing-Problem – where at most one packet originates in each node and at most one packet is destined for each node. The search focuses on algorithms having certain attributes that indicate simplicity, hoping that such algorithms will be practical.

The following are common qualitative attributes that indicate simplicity. A routing algorithm is minimal if it moves each packet along a shortest path. It is oblivious [10] if the path of each packet depends only on its source and destination (and independent of other packets in the network). In a source-oblivious [8] algorithm, the path a packet takes from any intermediate location, including the origin, depends only on that location and the destination.

Let \( M \) be a \( k \)-dimensional mesh, with a given order on its dimensions. (In the case of the 2-dimensional mesh, the horizontal dimension precedes the vertical one.) A strongly-dimensional path in \( M \) is a concatenation of \( k \) sub-paths, \( p_1, p_2, \ldots, p_k \), s.t. each \( p_i \) is a simple (potentially empty) path that uses only edges of the \( i \)-th dimension. A weakly-dimensional path is similar to a strongly-dimensional one, except that the order of the sub-paths does not have to follow the given order of the dimensions.

A routing algorithm on \( M \) is a strongly-dimensional algorithm (also called a dimension order algorithm [4]) if the path of each packet is strongly-dimensional. Note that such an algorithm is source-oblivious and minimal.

An algorithm on \( M \) is weakly-dimensional if the path of each packet is weakly-dimensional. Note that such an algorithm is minimal, but not necessarily oblivious.

All the above attributes are path-oriented. The following attributes refer to other aspects of the algorithm. An algorithm is short-sighted [2] if the destination address of a packet is not completely visible to a node; rather, the node only knows which of its outgoing edges advances the packet toward its destination. An oblivious algorithm is pure [8] if any packet always moves unless the next edge of its path is busy by another packet.

In addition to the above qualitative attributes, there are several quantitative attributes that relate to practicality. These attributes are considered as complexi-
ties of the algorithm and are measured by the following numbers, all referring to the worst case. The queue complexity, \( Q \), is the number of packets that resides in a node, excluding those that have reached their destination. The state complexity, \( S \), is the number of bits required to encode the state of a node. The data complexity, \( D \), is the number of bits transferred on an edge in a step (in addition to a packet). Finally, the time complexity, \( T \), is the number of steps.

The state and data complexities of routing algorithms are rarely addressed in the literature. However, these numbers certainly are good indicators of practicality and we address them with regard to all our algorithms.

The negative results regarding the above attributes are as follows: Chinn et al. [4] proved that any permutation routing algorithm on the \( n \times n \) mesh that is minimal, short-sighted and with \( Q = \Theta(n^2) \) satisfies \( T = \Theta(n^2) \).

Krizanc [8] showed that any permutation routing algorithm on the \( n \times n \) mesh that is source-oblivious, pure and with \( Q = \Theta(1) \) satisfies \( T = \Omega(n^2) \). He raised the following open question: “Does the lower bound shown above still hold if we drop the condition that the strategy be pure?” Our paper settles this question.

The positive results regarding the above attributes are as follows: Chinn et al. [4] presented the first minimal (but not oblivious) permutation routing algorithm on the \( n \times n \) mesh with \( T = O(n) \) and \( Q = O(1) \).

Iwana and Miyano [6, 7] suggested an oblivious (but not minimal) permutation routing algorithm on the \( n \times n \) mesh with \( T = O(n) \) and \( Q = \tilde{O}(1) \). They introduced the idea of “evenly distributing” certain sets of packets in order to prevent hot spots; our work was inspired by that idea.

Our main result is a strongly-dimensional permutation routing algorithm on the \( n \times n \) mesh whose complexities are \( T = O(n) \), \( Q = O(1) \), \( S = O(\log n) \) and \( D = O(\log n) \). (The constants implicit in the ‘\( O \)’ notations are small.) This is the first algorithm having the above attributes. Moreover, this is also the first source-oblivious and minimal algorithm with complexities of \( T = O(n) \) and \( Q = O(1) \), and also the first oblivious and minimal algorithm with these complexities. This algorithm settles Krizanc’s question.

However, this algorithm works under the relaxed model in which nodes can freely send data to their neighbors. In a more prevalent model, the standard model, data may be sent only when accompanied by a packet [4]. The above algorithms of Chinn et al. and of Iwana and Miyano are under the latter model.

Under the standard model we present the following two algorithms. The first is strongly-dimensional with complexities of \( T = O(n\log n) \), \( Q = O(1) \), \( S = O(\log n) \) and \( D = O(\log n) \). This is the first strongly-dimensional routing algorithm under the standard model with \( Q = O(1) \) whose \( T \) is better than the trivial \( O(n^2) \). Moreover, this is also the first source-oblivious and minimal algorithm with the above properties. This algorithm settles Krizanc’s question under the standard model.

The second algorithm is weakly-dimensional (and thus also minimal) and oblivious (but not source-oblivious) whose complexities are \( T = O(n) \), \( Q = O(1) \), \( S = O(\log n) \) and \( D = O(\log n) \). This is also the first minimal and oblivious algorithm under the standard model with \( T = O(n) \) and \( Q = O(1) \).

Our algorithms under the standard model mimic the one under the relaxed model and present no new techniques. Moreover, the standard model has no practical advantages over the relaxed one. We converted our first algorithm into the other two mainly because the standard model is more prevalent.

Our algorithms are based on a certain flow control technique described shortly. The essence of flow control is to slow down the traffic in order to avoid hot spots, which may seriously damage the network’s performances. The importance of flow control in computer networks is well established [5]; These networks invariably use closed-loop flow control. However, closed-loop flow control is underdeveloped in contemporary algorithms for synchronous Store-and-Forward networks.

Our algorithms are based on a new technique of open-loop flow control. Under this technique, described here in the context of the relaxed model, an algorithm is composed of two stages: setup and transportation. Packets are stationary during the setup stage which computes certain values and stores then in the network. In particular, it computes a rational number \( a(\epsilon) \), \( 0 \leq a(\epsilon) \leq 1 \), for certain critical edges \( \epsilon \).

The transportation stage moves the packets to their destinations. It uses the computed values to slow the packets so that the traffic on each critical edge \( \epsilon \) is bounded by \( a(\epsilon) \); that is, at most \( a(\epsilon) \cdot |I| \) packets traverse \( \epsilon \) during any \( |I| \) consecutive steps.

This bound on the burstiness of the traffic enables the algorithm to avoid hot spots and maintain bounded queues. All this is achieved by an open-loop control; that is, during this stage no information is transferred in a direction opposite to that of the packets.

Recently, Scheideler and Vöcking [9] presented a general technique to produce a dynamic routing algorithm from a static one. Our work follows the reverse direction, applying a dynamic routing problem to solve a static one.

The study of dynamic routing is currently under an active research that follows the Adversarial Queuing Theory introduced by Borodin et al. [3] and farther extended by Andrews et al. [1].

As discussed in section 5, there are significant differences between that theory and the dynamic routing studied in this paper. The dynamic problem we solve seems easy as its network is just a linear array. We show, however, that this problem is unsolvable by the techniques of Adversarial Queuing Theory.

The paper is organized as follows: Section 2 presents the Store-and-Forward models used in this work, sections 3 to 6 present our strongly-dimensional routing algorithm under the relaxed model, section 7 presents our routing algorithms under the standard model, and section 8 shows that our dynamic routing problem is
unsolvable under the Adversarial Queuing Theory.

We use the following notations: \( \mathbb{N}, \mathbb{R}^+ \) and \( \mathbb{Q}^+ \) denote the non-negative integers, real and rational numbers, respectively.

## 2 The Store-and-Forward models

This section presents the two versions of the Store-and-Forward model used in this work. Under this model, a packet routing network is represented by a digraph \( G = (V, E) \) where the nodes denote processing elements and the edges denote directed communication links. Most networks, including the mesh, are bidirectional – each directed edge is accompanied by the reverse one; however, this is not required by the model.

The network works in a synchronous manner; that is, time proceeds in discrete steps simultaneously in all nodes.

A packet is an atomic object that the nodes handle in a restricted manner. A node can neither generate a new packet, nor duplicate, nor modify one. Most of the contents of a packet is invisible to the nodes; the only exception is its destination address.

An edge is a directed communication link that transfers information from its tail to its head. It transfers at most one packet in each step. In addition, it transfers data, restricted to a certain finite set, in every step.

A node is a Moore Finite State Machine (FSM) augmented with packet-processing capacities. As any FSM, it maintains a state that varies from step to step and whose values are members of a certain finite set. In addition, the FSM has several buffers, each can hold a single packet or be empty.

The operation of the network is derived from the operation of the nodes. Each step is divided into two sub-steps in which a node operates as follows:

In the first sub-step the node sends data, and optionally a packet, on each of its outgoing edges. The current-state of the node determines, for each outgoing edge, the data value sent on the edge and the buffer whose content is sent on the edge, if at all. Note that our FSMs are Moore rather than Mealy machines. Hence, the output of a node is independent of its current-input.

In the second sub-step the node receives, through its incoming edges, the packets and data that were sent in the first sub-step. This information is referred to as the current-input. Recall, however, that only the destination addresses of the packets are visible to the node and participate in the current-input. The current-state, combined with the current-input, determines where each incoming packet is stored (if at all) and the next-state of the FSM – its state during the next step.

The model is restricted in such a way to prevent duplication of packets; e.g., an FSM can not store an incoming packet in two buffers. The model, however, does not prevent loss of packets; the algorithms should take care of that. Clearly, a packet routing algorithm must not lose any packets.

In most applications, an FSM should keep track of the contents of its buffers – which of them contain packets and what are their destinations. Hence, an FSM usually records this information in its state.

The above describes the operation of a node under the relaxed model. The standard model differs from the relaxed one only in the operation of a node in the first sub-step: A node can send meaningful data on an outgoing edge only when it sends a packet on that edge; otherwise, it must send a predefined value. (Some authors refer to the data that accompanies a packet under the standard model as the state of the packet.)

This standard model is very prevalent and follows the model explicitly defined by Chinn et al. [4]. However, an additional feature – a rudimentary closed-loop flow control – is built into their model as follows. A packet is transferred only after the sender requests and the receiver grants a permission for this transfer; this handshake is done in every step.

We omit this feature because of the following reasons. It transfers information in a direction reverse to that of an edge; it is redundant under the relaxed model when the network is bidirectional; many algorithms do not need it; and our algorithms do not use any feedback.

An algorithm (of local control) is the programming of the FSM of each node so that the network performs a desired task. The FSM’s are not required to be identical and may depend on the size of the network.

## 3 Overview of the routing algorithm

This section provides an overview of our main result – a strongly-dimensional routing algorithm on the \( n \times n \) mesh under the relaxed model, whose complexities are \( T = O(n) \), \( Q = O(1) \), \( S = O(\log n) \) and \( D = O(\log n) \).

For the sake of simplicity we classify the packets to 4 classes according to their source-to-destination direction and route one class at a time. Due to symmetry, it suffices to consider only one class. Hence, we henceforth assume that the directions of the packets are in the upper right quadrant; i.e., packets need to move either upward or rightward or both.

It is beneficial to present this algorithm, not in the context of the mesh, but in the context of the following graph. The \( n \times n \) double-mesh is a spanning subgraph of the \( 3 \)-dimensional \( n \times n \times 2 \) mesh. The background \( n \times n \) mesh, called the horizontal mesh, has only the horizontal edges (of both directions); the foreground \( n \times n \) mesh, called the vertical mesh, has only the vertical edges.

There are directed edges from the horizontal mesh to the vertical mesh, referred to as crossing edges. There are no edges from the vertical mesh to the horizontal mesh. See figure 1.

The Mesh-to-Mesh Permutation-Routing-Problem is a Permutation-Routing Problem on the \( n \times n \) double-mesh in which the packets originate in the horizontal mesh and are destined for the vertical mesh.

Clearly, the Permutation-Routing-Problem on the \( n \times n \) mesh is reducible to the Mesh-to-Mesh Permutation-
Routing-Problem on the $n \times n$ double-mesh.

The structure of the double-mesh enforces a simple and rigid structure on any algorithm on this graph. The algorithm naturally bises into two algorithms, $A$ and $B$, operating on the horizontal and vertical meshes, respectively. The communication between the algorithms is unidirectional – from $A$ to $B$ but not from $B$ to $A$. This implies that, in our case, $B$ should solve a certain dynamic routing problem. Moreover, as both meshes are composed of unconnected linear arrays, $A$ and $B$ are actually just linear array algorithms. Due to this structure, we prefer to work in the double-mesh rather than in the mesh.

Our algorithm is based on the technique of open-loop flow control presented in the introduction. The crossing and vertical edges are the critical edges in this case.

The presentation of the algorithm is organized as follows. The horizontal part, the vertical part and the entire algorithm are presented in sections 4, 5 and 6, respectively.

### 4 The horizontal routing

In this section we present the horizontal part of our routing algorithm for the Mesh-to-Mesh Permutation-Routing-Problem.

We use the following terminology. A packet routing problem is a One-Many problem if at most one packet originates at each node, but several packets may share the same destination. A packet $p$ is **bound** to an edge $e$ if $p$ must traverse $e$ to reach its destination. Let $E'$ be a set of edges and let $\alpha : E' \rightarrow \mathbb{R}^+$. The traffic on $E'$ is **bounded by $\alpha$** if the traffic on each $e \in E'$ is bounded by $\alpha(e)$.

Let the **$n$-comb network** be the subgraph of the $n \times n$ double-mesh induced by two neighboring rows, one of the horizontal and the other of the vertical mesh. See figure 2. Denote by $R_1$ ($R_2$) the set of nodes of the first (second) row, and let $C$ be the set of the edges from $R_1$ to $R_2$, called the crossing edges. For a node $v \in R_2$, denote by $c(v)$ the crossing edge entering $v$.

Given an instance of a One-Many Packet-Routing-Problem on the $n$-comb network and an edge $e \in C$, define

$$N(e) \triangleq \text{number of packets bound to } e$$

and

$$\alpha(e) \triangleq N(e)/n.$$

**The Horizontal-Routing-Problem.** In the following we define the **Horizontal-Routing-Problem**. It is a One-Many Packet-Routing-Problem on the $n$-comb network: all the packets originate in the first row and are destined for the second one; and all the packets are directed right.

The objective of the problem is twofold: The first task is to compute $\alpha(e)$ for each $e \in C$ and to arrive at the following situation:

**S1.** Each $v \in R_2$ has $\alpha(c(v))$.

The second task is the following task:

**T1.** To move the packets to their destinations along shortest paths in such a way that the traffic on $C$ is bounded by $\alpha$.

The second task should begin only after the first one is completed.

**Remark:** When this problem is used to solve the Mesh-to-Mesh Permutation-Routing-Problem, the destination of a packet under the former problem corresponds to the destination column of the same packet under the latter problem.

As said, our algorithm for the Horizontal-Routing-Problem is composed of two stages: setup and transportation. The objective of the setup stage is to deliver to the nodes of $R_1$ some information needed for the transportation stage and to reach the (S1) situation. (Note that only the second objective is explicitly specified by the problem.) The objective of the transportation stage is to perform the (T1) task. In the next two subsections we present the transportation stage followed by the setup stage.

#### 4.1 The transportation stage

The objective of the transportation stage relates to the $\alpha$ function. We prefer, however, to consider a general case where $\alpha$ is replaced by any $\hat{\alpha} : C \rightarrow \mathbb{Q}^+$ satisfying the following condition: $\hat{\alpha}(e) > 0$ whenever $N(e) > 0$.

Let

$$\rho(v) \triangleq \begin{cases} 0 & \text{no packet originates at } v \\ \hat{\alpha}(e) & \text{the packet originating at } v \\ \text{is bound to } e \end{cases}$$

The objective of this stage is to perform the (T1) task, with $\alpha$ replaced by $\hat{\alpha}$, given that the following initial condition holds.
S2. Each \( v \in R_1 \) holds \( \rho(v) \).

The transportation algorithm. We use the following metaphor of a cab. A cab is generated in the leftmost node of \( R_1 \) and advances rightward in every step. During its travel a cab may pick up a packet. Once a cab picks up a packet it carries it to its destination and vanishes there; otherwise, it vanishes in the rightmost node of \( R_1 \).

Let \texttt{HorizontalTransportation} be the following algorithm: A cab is generated in every step. An empty cab, passing through a node \( v \) having a packet \( p \), picks up this packet unless another cab, carrying a packet destined for \( p \)'s destination, has passed through \( v \) less than 1/\( \rho(v) \) steps earlier. To this end, a node having a packet \( p \) maintains a counter that records the number of steps since a cab, carrying a packet destined for \( p \)'s destination, traversed the node.

To implement the cab metaphor, a node needs to resolve, in each step, whether it receives an occupied cab, an empty cab, or no cab at all. The first case is easily distinguished from the others, since the node receives a packet only in this case. To distinguish between the last two cases, each node should send a data bit to its right neighbor.

**Claim 1:** If a cab \( A \) carried a packet \( p \) to its destination \( v \) then \( p \) was the first packet destined for \( v \) that the cab encountered during its travel.

**Proof:** Suppose, for the sake of contradiction, that \( A \) encountered a packet \( p' \) destined for \( v \) before encountering \( p \). As \( A \) did not pick up \( p' \), a cab \( A' \) carrying a packet destined to \( v \) passed through the source of \( p' \) some \( t < 1/\hat{o}(e(v)) \) steps before \( A \). This \( A' \) also passed through the source of \( p \) the same number of steps, \( t \), before \( A \); thus \( A \) could not pick up \( p \). A contradiction.

**Claim 2:** For every \( \epsilon \in C \), at most one packet traverses \( \epsilon \) during any \( \lfloor 1/\hat{o}(\epsilon) \rfloor \) consecutive steps.

**Proof:** Assume that two packets, \( p_1 \) and \( p_2 \), violate our claim. Let \( v_1 \) and \( v_2 \) be the sources of these packets, \( A_1 \) and \( A_2 \) be the cabs that carry them and \( t_1 < t_2 \) be the steps when these cabs traverse \( \epsilon \). By our assumption, \( t_2 - t_1 < 1/\hat{o}(\epsilon) \).

The cab \( A_1 \) precedes \( A_2 \). Hence, by Claim 1, \( v_1 \) is to the left of \( v_2 \), i.e., \( A_1 \) carried \( p_1 \) when it passed through \( v_2 \). This, combined with \( t_2 - t_1 < 1/\hat{o}(\epsilon) \), implies that \( A_2 \) did not pick up \( p_2 \). A contradiction.

Claim 2 implies:

**Claim 3:** During the execution of \texttt{HorizontalTransportation} the traffic on \( C \) is bounded by \( \hat{o} \).

Let

\[
M = \max \{ [N(\epsilon)/\hat{o}(\epsilon)] \mid \epsilon \in C, \hat{o}(\epsilon) > 0 \} + n.
\]

**Claim 4:** All packets are picked by cabs generated in the first \( M \) steps.

**Proof:** A \textit{cab segment} is a group of cabs generated in consecutive steps; a cab is \textit{profitless} if it does not pick up a packet during its travel. Denote the segment of cabs generated in the first \( M \) steps by \( B \).

Suppose, for the sake of contradiction, that some packet \( p \) bound to \( \epsilon \in C \) was not picked up by any cab of \( B \). Clearly, at least \( N(\epsilon)/\hat{o}(\epsilon) \) of the cabs in \( B \) are profitless.

Denote by \( B' \) the set of cabs of \( B \) that do not traverse \( \epsilon \). \( |B| - |B'| \leq N(\epsilon) - 1 \). Thus \( B' \) is fragmented into, at most, \( N(\epsilon)/\hat{o}(\epsilon) \) segments. Hence, one of them, \( S \), has at least \( N(\epsilon)/\hat{o}(\epsilon) - N(\epsilon) = 1/\hat{o}(\epsilon) \) profitless cabs. Thus, the last profitless cab of \( S \) passed through the source of \( p \) at least \( 1/\hat{o}(\epsilon) \) steps after any cab that traversed \( \epsilon \); this profitless cab should have picked up \( p \). A contradiction.

To implement this algorithm, the FSMs should store \( \rho(v) \) and maintain the counter. By Claim 4, the counter never exceeds \( M \). In the case of \( \hat{o} = \alpha \) it holds that \( M = 2 - n \) and that \( \rho(v) \) is of the form \( k/n \) where \( k \leq n \) is an integer. Hence, \( O(\log n) \) state bits are enough. The horizontal transportation is summarized by the following claim.

**Claim 5:** The transportation stage can be done by an algorithm with \( T = O(M), Q = O(1) \) and \( D = O(1) \). Moreover, if \( \hat{o} = \alpha \) then also \( T = O(n) \) and \( S = O(\log n) \).

4.2 The setup stage

The objective of the setup stage is to establish conditions (S1) and (S2) without moving packets.

To this end we use the abstraction of a labeled-counter. This object has a fixed label and a variable counter. At the first step of this stage each node \( v \) of \( R_1 \) generates a counter labeled with the crossing edge \( e \) leading from \( v \). In the following steps all these objects march leftward, counting on their way the number of packets that are bound to their label. At the leftmost node these objects turn back and return to their origin where they vanish. On the way back they deliver the required information to the appropriate nodes.

**Claim 6:** The setup stage can be done by an algorithm with \( T = O(n), Q = O(1), S = O(\log n) \) and \( D = O(\log n) \).

4.3 The horizontal routing algorithm

Applying the above setup and transportation stages solves the Horizontal-Routing-Problem defined in the beginning of this section.

**Lemma 1:** There is an algorithm for the Horizontal-Routing-Problem with \( T = O(n), Q = O(1), S = O(\log n) \) and \( D = O(\log n) \).
5 The vertical routing

In this section we present the vertical part of the routing algorithm for the Mesh-to-Mesh Permutation-Routing-Problem.

The objective of the vertical mesh is to route the packets arriving from the horizontal mesh to their destinations while maintaining the queues below some constant (which is independent of \( n \)). Recall that the connection between the horizontal and the vertical mesh is unidirectional – there are no edges from the vertical mesh to the horizontal mesh. Thus, the vertical mesh has no control on the arrival of packets. This gives rise to a dynamic routing problem.

In a dynamic routing problem packets are injected into the system at arbitrary times. The injection process never terminates; hence, a dynamic routing problem lasts forever. This contrasts with a static routing problem in which all packets initially reside in the network and the problem ends when all of them reach their destinations.

We model this injection as follows. The network is augmented with an extra node, called the injector and denoted \( R \). There is a set \( J \) of injecting edges leading from the injector to the rest of the network (but there are no edges in the reverse direction). The injector injects packets into the network through these edges. Let \( a : J \rightarrow \mathbb{R}^+ \). The injector is bounded by \( a \) if the traffic on \( J \) is bounded by \( a \).

The graph in hand, the vertical mesh, is a set of unconnected linear arrays each working independently of the others. Hence, the network in question, denoted \( D_n \), is a linear array augmented with an injector. As we assume that the directions of the packets are in the upper right quadrant and since our algorithm does not use any feedback, the edges of the array are unidirectional. See figure 3.

We name the elements of \( D_n \) as follows. The nodes of the array are \( v_1, \ldots, v_n \); \( e_i \triangleq (v_i \leftarrow v_{i+1}) \) and \( r(v_i) \triangleq (R \leftarrow v_i) \). See figure 3.

![Figure 3: The \( D_n \) network](image)

We intend to use our dynamic routing problem (to be defined shortly) to solve the Mesh-to-Mesh Permutation-Routing-Problem. In this context, each node \( v \) will initially hold a bound \( a(r(v)) \) on the traffic over \( r(v) \); this is the bound computed and delivered to the nodes of \( R \) by the horizontal setup stage. By the definition of \( a \), \( a(r(v)) \cdot n \) is the number of packets that are bound to \( r(v) \). Since at most \( n \) packets are destined for a column, the sum of \( a(r(v)) \) taken over all nodes \( v \) of our array will be at most 1.

**The Vertical Routing Problem.** In the following we define the Vertical-Routing-Problem. It is a dynamic routing problem on the \( D_n \) network in which the injector complies with the following restrictions:

- **R1.** The injector is bounded by some function \( a : J \rightarrow \mathbb{Q}^+ \).
- **R2.** \( \sum_{r \in J} a(r) \leq 1 \).
- **R3.** This function is given to the network as follows: each node \( r \) is initially provided with \( a(r(v)) \).
- **R4.** A packet injected through \( r_i \) is destined to \( v_k \) where \( k \geq i \).

The objective of the problem is to move the packets towards their destinations in such a way that the number of packets in a node never exceeds some constant which is independent of \( n \) and \( a \). It is not required that the packets actually reach their destinations.

**Remark:** When we study the Vertical-Routing-Problem by itself, the above \( a \) function is not necessarily identical to the function \( a \) defined in section 4. However, when we will use this problem to solve the Mesh-to-Mesh Permutation-Routing-Problem then the two \( a \) functions will be identical.

Note that all the above restrictions, but R2, consider each injecting edge in isolation, independently of the other edges. So, for example, the injector may, in a single step, inject a packet through each edge \( r \) with \( a(r) > 0 \).

For our needs it suffices to solve the following relaxed version of the above dynamic problem. Firstly, the array is made bidirectional; the reverse edges may be used for feedback. Secondly, restriction R2 is replaced with \( \sum_{r \in J} a(r) \leq c \) where \( c > 0 \) is an arbitrary small constant.

It seems that some simple closed-loop flow control suffices to solve the relaxed version and, moreover, restriction R3 seems redundant. The feasibility of such a solution is suggested by Iwana and Miyano [6, 7].

Let the Ignorant-Vertical-Routing-Problem be the problem derived from the relaxed version by dropping R3. We show in section 8 that this last problem is unsolvable. In other words, the initially given bounds on the injecting edges are mandatory for solving our dynamic problem.

The study of dynamic routing is currently under an active research that follows the Adversarial Queuing Theory introduced by Borodin et al. [3] and further extended by Andrews et al. [1].
There are significant differences between that theory and the dynamic routing studied in this paper. Under that theory it is desired that the system will be stable, that is, that the queues will be bounded by a constant which is independent of time. We further require that this constant is independent of the size of the network. However, we study only a single and simple topology, namely linear array, while that theory considers arbitrary networks.

Moreover, there is a significant difference concerning the underlying technique. Under our technique of open-loop flow control, the network pre-processes the given bounds on the load and sets itself up for the actual transportation. Under that theory there is no such setup stage and the network does not take advantage of any given numbers concerning its load. Therefore, as shown in section 8, neither the Vertical-Routing-Problem, nor its relaxed version, are solvable under that theory.

We solve the Vertical-Routing-Problem by open-loop flow control. Thus the algorithm has two stages: setup and transportation.

**The setup stage.** The objective of the setup stage is to compute, for each vertical edge $e_j$, the value $a'(e_j)$ defined by

$$a'(e_j) = \sum_{i \in \overline{j}} a(r_i),$$

and to store that value in $v_{i+1}$.

This setup is easily accomplished in $n$ steps. Unfortunately, we do not know to solve the Vertical-Routing-Problem when packets are injected during the setup stage. Hence, we relax the problem by adding the following restriction on the injector:

**R5.** No packets are injected during the first $n$ steps.

### 5.1 The transportation stage

The objective of the transportation stage is to move the packets toward their destinations in such a way that the traffic on each vertical edge $e_j$ is bounded by $a'(e_j)$ and the queues are bounded by a constant. It is not required that the packets actually reach their destinations.

Due to the setup stage, each node $v_i$ holds both $a(r_i)$ and $a'(e_{i+1})$. Hence, this problem is actually reducible to a simple dynamic routing problem, involving a single node, as follows:

![Figure 4: The single node network](image)

**The Single-Node-Problem.** This problem concerns a single node with two incoming edges, $a$ and $b$, coming from the injector and one outgoing edge, $c$. See figure 4. The node is provided with the values $a(a), a(b) \in \mathbb{Q}^+$ s.t. $a(a) + a(b) \leq 1$, and the injector is bounded by $a$.

The objective of the node is to move packets in such a way that the following holds:

1. The traffic on $c$ is bounded by $a(a) + a(b)$.
2. The number of packets in the node never exceeds some constant (which is independent of $a$).

Let $\gamma = a(a) + a(b)$. Let **SingleNode** be the following single node algorithm: The algorithm maintains a rational state variable $0 \leq q < 1$. In the initial state $q = 0$. The edge $c$ is enabled in a certain step if $q + \gamma \geq 1$ in this step; it may transfer a packet only when enabled. In every step the node carries out the following:

- If $c$ is enabled and the node has some packets then one of them is sent along $c$.
- The packets received through $a$ and $b$ are stored.
- $q = (q + \gamma) - (q + \gamma)$

The new value of the variable $q$ takes effect only in the next step.

**Claim 7:** $|x + y| \leq |x| + |y| \leq |x + y|$ for any real numbers $x$ and $y$.

**Proof:** The inequalities in question are preserved under addition of integers to $x$ and $y$. Hence, it suffices to consider the case of $|x| = |y| = 0$. In this case $x \geq 0$ and $y \leq 0$. Hence,

$$|x + y| \leq |x| + |y| = |y| \leq |x + y|$$

For a non-negative integer $t$, let time (or step) $t$ denote the $(t+1)$-th step; e.g., the initial step is time 0. A time interval $I$ is an interval of non-negative integers that are interpreted as steps in the above sense.

Let $E(I)$ be the number of steps in which $c$ is enabled in the interval $I$. Note that $E(I)$ depends only on $\gamma$ and on the interval $I$ and is independent of the behavior of the injector.

**Claim 8:** $|\gamma \cdot |I|| \leq E(I) \leq [\gamma \cdot |I|]$ for any time interval $I$.

**Proof:** Let $q(t)$ denote the value of $q$ at time $t$. By our algorithm, $q(t) = q(t-1) - \gamma \cdot t$ and $E([0,t]) = [\gamma \cdot t]$. Let $0 \leq t_1 \leq t_2$ and $I = [t_1,t_2]$. Then:

$$E(I) = E([0,t_2]) - E([0,t_1]) = [\gamma \cdot t_2] - [\gamma \cdot t_1]$$

$$= [\gamma \cdot t_2] - [\gamma \cdot t_1].$$

By Claim 7,

$$[\gamma (t_2 - t_1)] \leq E(I) \leq [\gamma (t_2 - t_1)].$$

**Claim 9:** In the course of the SingleNode algorithm:
1. The traffic on $c$ is bounded by $a(a) + a(b)$.

2. The node never has more than three packets at the beginning of a step.

**Proof:** Claim 8 implies (1). Let $Y(I) (X(I))$ be the number of packets that entered (left) the node during interval $I$. Let $b(t)$ be the number of packets in the node at the beginning of step $t$. Fix a time $t$ and let $t' \leq \max \{ t \leq t' : b(t') = 0 \}$. As $b(0) = 0$, the above set is not empty. Let $I = [t', t]$. Since $b(t') = 0$ we have $b(t) = Y(I) - X(I)$. During the interval $I$, the node is empty only in $t'$. Therefore, $X(I) \geq E(I) - 1$. Claim 8 implies $E(I) \geq \lfloor \gamma / |I| \rfloor$. Thus

$$b(t) = Y(I) - X(I) \leq$$

$$\lfloor a(a) \cdot |I| \rfloor + \lfloor a(b) \cdot |I| \rfloor - \lfloor \gamma / |I| \rfloor + 1 <$$

$$< a(a) \cdot |I| + 1 + a(b) \cdot |I| + 1 - (\gamma / |I| - 1) + 1$$

$$= 4$$

5.2 The vertical routing algorithm

The above setup and transportation stages solve the Vertical-Routing-Problem defined in the beginning of this section. Hence:

**Lemma 2:** There is an algorithm that solves the Vertical-Routing-Problem with $T = O(n)$ and $Q = O(1)$. Moreover, if the $a(r_i)$ are restricted to the form of $k/n$ for some integer $k$, then $S = O(\log n)$ and $D = O(\log n)$.

6 The strongly-dimensional algorithm

In this section we show how to combine the linear array algorithms presented in sections 4 and 5 to construct a strongly-dimensional routing algorithm for the Mesh-to-Mesh Permutation-Routing-Problem.

Recall that we consider only packets whose directions are in the upper right quadrant. Let StronglyDimensional1 be the following 4-stage algorithm:

**Stage 1.** Each row of the horizontal mesh performs the horizontal setup stage. The vertical mesh is idle during this stage.

**Stage 2.** Each column of the vertical mesh performs the vertical setup stage. The horizontal mesh is idle during this stage.

**Stage 3.** In this stage the horizontal mesh and the vertical mesh work simultaneously. Each row of the horizontal mesh performs the horizontal transportation stage and each column of the vertical mesh performs the vertical transportation stage. Stage 3 terminates when the horizontal transportation stage terminates.

**Stage 4.** Each column in the horizontal mesh greedily routes its packets to their destinations.

The duration of each stage is $O(n)$ and depends only on $n$ (and not on the instance of the problem). Hence, each node needs only an $O(\log n)$-bit counter to tell when one stage ends and the other begins.

The discussion in sections 4 and 5, and the obvious reduction of the Permutation-Routing-Problem on the $n \times n$ mesh to the Mesh-to-Mesh Permutation-Routing-Problem on the $n \times n$ double-mesh imply:

**Theorem 1:** There is a strongly-dimensional routing algorithm for the Permutation-Routing-Problem on the $n \times n$ mesh, under the relaxed model, whose complexities are $T = O(n)$, $Q = O(1)$, $S = O(\log n)$ and $D = O(\log n)$.

7 Routing under the standard model

This section is devoted to routing under the standard model in which data may be sent only when accompanied by a packet.

Under this model we present two permutation routing algorithms for the $n \times n$ mesh. The first is strongly-dimensional (and thus also source-oblivious and minimal). Its complexities are $T = O(n \cdot \log n)$, $Q = O(1)$, $S = O(\log n)$ and $D = O(\log n)$. (Note that $T = O(n \cdot \log n)$ rather than $O(n)$.)

The second algorithm is oblivious and weakly-dimensional (hence also minimal, but is neither strongly-dimensional nor source-oblivious). Its complexities are $T = O(n)$, $Q = O(1)$, $S = O(\log n)$ and $D = O(\log n)$.

Our routing algorithms under the standard model mimic our routing algorithm under the relaxed model. Hence, we consider only packets whose directions are in the upper right quadrant, we work in the double-mesh rather than the mesh, the algorithms are composed of horizontal and vertical parts each mimicking the corresponding part under the relaxed model, etc.

The main difficulty is in mimicking the horizontal setup stage. The task accomplished there by moving data has to be accomplished here by moving packets which may move only toward their destinations. Due to this difficulty the resulting 2-dimensional algorithm solves only a restricted version of the routing problem. Successive applications of this algorithm to different parts of the mesh produce the final permutation routing algorithms.

We use the following notations: A *vertical* (horizontal) slice of a mesh $M$ is a nonempty subgraph of $M$ induced by nodes of consecutive columns (rows). The *height (width)* of a mesh is the number of rows (columns) in the mesh. For two sets of nodes, $X$ and $Y$, an $X$ to $Y$ packet is a packet originating in $X$ and destined for $Y$.

This section is organized as follows: In subsections 7.1, 7.2 and 7.3 we mimic the algorithms of the relaxed
Consider again the Horizontal-Routing Problem on the n-comb network of section 4. Under the standard model we know only to solve a variant of the problem, in which a gap is given between the sources and the destinations.

A segmented n-comb network is an n-comb network, whose n is a multiple of 8, that is partitioned to 3 vertical slices, denoted by \( S, G \) and \( D \), from left to right. The width of \( G \) is \( n/8 \) and the widths of \( S \) and \( D \) are multiples of \( n/8 \). See figure 5.

**Figure 5**: A segmented n-comb network.

### The Segmented Horizontal Routing Problem

In the following we define the version of the Horizontal-Routing-Problem solved under the standard model.

The **Segmented Horizontal Routing Problem** is a One-Many routing problem on a segmented n-comb network that is similar to the Horizontal-Routing-Problem of section 4. The initial conditions of the former are derived from the initial conditions of the latter by adding the following requirements:

1. All the packets are \( S \) to \( D \) packets.

2. Each packet \( p \) holds an integer \( r(p) \). If \( p \) and \( q \) are distinct packets having the same destination then \( r(p) \neq r(q) \).

(When using this problem for 2-dimensional routing, \( r(p) \) will be the row destination of the packets.)

A packet \( p \) is a **messenger packet** if \( r(p) \geq r(q) \) for all packets \( q \) having the same destination as \( p \).

The objective of our problem is twofold: The first task is to compute \( a(c) \) for each edge \( c \in C \), to store that value at the head of \( c \) and to move each messenger to its destination. The second task is to move the rest of the packets to their destinations along the shortest paths in such a way that every edge \( c \in C \) is bounded by \( a(c) \). The second task should begin only after the first one is completed.

As usual, this problem is solved by setup and transportation stages, that perform the first and the second tasks, respectively.

**The setup stage.** First we arrive at the following situation:

1. All packets are in \( G \), with at most 6 packets in a node.

2. Each packet destined for \( v \) carries the value \( a(c(v)) \).

3. Each packet knows whether or not it is a messenger.

This is achieved by routing the packets from \( S \) to \( G \) in such a way that each packet encounters all other packets. This enables each packet to resolve whether it is a messenger and to compute \( a(c(v)) \) for its destination node \( v \).

After this situation is reached, the messenger packets, together with their data, are greedily routed to their destinations. This setup is easily done in \( O(n) \) steps.

**The transportation stage.** At the end of the setup stage each packet destined to \( v \) carries \( a(c(v)) \); hence, the horizontal transportation stage of subsection 4.1 should do. Yet, there appears to be a problem. That algorithm transmits a data bit to distinguish between an empty cab and an absence of a cab and this bit is not accompanied by a packet. However, packets are picked by cabs only in \( G \), and delivered to destination only in \( D \); hence, cabs are never absent in \( G \) and they never pick up a packet in \( D \). Thus the above data bit is redundant.

### The Segmented Vertical Routing Problem

The **Segmented Vertical Routing Problem** is the following version of the Vertical-Routing-Problem solved under the standard model. The **Segmented-Vertical-Routing-Problem** is derived from the Vertical-Routing-Problem (of section 5) by adding the following restrictions to restrictions R1 to R5 of that problem.

**RM1.** Initially, each node \( v_i \) with \( a(r_i) > 0 \) has a packet destined for \( v_{j(i)} \) s.t. \( j(i) \geq i \).

**RM2.** A packet injected through \( r_i \) is destined for \( v_k \) where \( i \leq k \leq j(i) \).

The objective of our problem is to move the packets towards their destinations, while maintaining the queues below some constant which is independent of \( n \) and \( a \).

Denote the packets that are initially present in the network by initial packets. As usual, this problem is solved by a setup stage and a transportation stage.

**The setup stage.** The setup stage must complete in \( n \) steps, before packets are injected. Its objective is to compute \( a'(\epsilon_i) \), for each vertical edge \( \epsilon_i \), and to store this value in \( v_{i+1} \); note that this has to be done by the initial packets. Under the relaxed model, \( a' \) was defined by: \( a'(\epsilon_i) = \sum_{j \leq i} a(r_j) \). However, a minimal algorithm can not compute such an \( a' \) under the standard model. We therefore revised the definition of \( a' \). We introduce a fictitious edge \( \epsilon_0 \) and define \( a' \) recursively as follows:

\[
a'(\epsilon_i) = \begin{cases} 
0 & \text{no initial packet} \\
\alpha'(\epsilon_{i-1}) + a(r_i) & \text{is bound to } \epsilon_i
\end{cases}
\]

\( i > 0 \)
With the revised $a'$, the setup stage is easily done in $n$ steps.

The transportation stage. The objective of the transportation stage is to route packets toward their destinations while maintaining the queues below some constant. The vertical transportation stage here is identical to that under the relaxed model (section 5), except that the $a'$ functions are different. Induction on $i$, using Claim 9 and restriction RM2, establishes that the traffic on each edge $e_i$ is bounded by $a'(e_i)$ and that the queues are bounded.

7.3 The segmented 2-dimensional algorithm

A segmented $m \times n$ mesh is an $m \times n$ mesh, whose $n$ is a multiple of 8, that is partitioned to 3 vertical slices, denoted by $S$, $G$, and $D$, from left to right. The width of $G$ is $n/8$ and the widths of $S$ and $D$ are multiples of $n/8$. See figure 6.

![Figure 6: A segmented $m \times n$ mesh.](image)

The Segmented-2-Dimensional-Problem is to route a permutation of $S$ to $D$ packets on a segmented $m \times n$ mesh. It is solved by algorithm GapRouting1.

Algorithm GapRouting1. Algorithm GapRouting1 is obtained from the horizontal and vertical setup stages and the horizontal and vertical transportation stages of the standard model, essentially in the same way algorithm StronglyDimensional1 (of section 6) is obtained from the corresponding stages of the relaxed model.

In this construction we take care of the following details: In the Segmented-Horizontal-Routing-Problem, $r(p)$ is the row destination of the packet. The vertical setup stage follows the horizontal setup stage and thus the initial packets of the vertical algorithm are the messenger packets of the horizontal algorithm.

However, there is a difficulty with the restrictions on the injector in the Segmented-Vertical-Routing-Problem, as follows. By the definition of $a'$ (in section 4), $a(r(v)) \cdot n$ is the number of packets that are bound to $r(v)$. Since, at most, $m$ packets are destined to a column $V$, $\sum_{v \in V} a(r(v)) \leq m/n$. Hence, when $m$ is larger than $n$, restriction R2 (of section 5) may not hold. This can be remedied by changing the function $a$ computed in the horizontal setup stage. The revised $a$ is the original one multiplied by $n/m$. After this adjustment R2 holds and, by Claim 5, the horizontal transportation stage completes in $O(m)$ steps. The above discussion implies:

Lemma 3: The Segmented-2-Dimensional-Problem on the $m \times n$ mesh is solvable, under the standard model, by a strongly-dimensional routing algorithm with $T = O(n + m)$, $Q = O(1)$, $S = O(\log (m + n))$ and $D = O(\log (m + n))$.

7.4 A strongly-dimensional $O(n \cdot \log n)$-step routing algorithm

In this subsection, we use the results of subsection 7.3 to obtain an $O(n \log n)$-step strongly-dimensional permutation routing algorithm on the $n \times n$ mesh under the standard model.

For the sake of simplicity, in this subsection $n$ denotes an integer which is a power of 2. The extension of our result to other values of $n$ is straightforward.

For a packet $p$, denote by $d_x(p)$ ($d_y(p)$) the distance between the packet’s source and destination in the horizontal-axis (vertical-axis).

First we introduce two auxiliary routing algorithms. Both operate on an $m \times n$ mesh and take $O(n + m)$ steps.

The first, GapRouting2, solves a permutation routing problem s.t. $n/4 \leq d_x(p)$ for all packets $p$. It works as follows. The case of $n < 8$ is trivial. Otherwise, it invokes GapRouting1 on all the segmented $m \times n$ meshes. Note that there are 6 such segmented meshes.

The second auxiliary algorithm, GapRouting3 with parameter $i$, $2 \leq i \leq \log_2 n$, solves a Permutation-Routing-Problem s.t. $n/2^i \leq d_x(p) < 2n/2^i$ for all packets $p$. The algorithm has two phases:

In the first phase it partitions the mesh to vertical slices whose width is $4n/2^i$, and executes GapRouting2 on each of the slices in parallel. In the second phase it removes the first and the last $2n/2^i$ columns from the original mesh and applies the first phase on the resulting mesh.

The final algorithm: The final algorithm, StronglyDimensional2, sequentially invokes GapRouting3 with $i = 3, 4, \ldots, \log_2 n$. It then invokes GapRouting2 and, finally, it routes packets $p$ with $d_x(p) = 0$. This establishes the following theorem.

Theorem 2: There is a strongly-dimensional permutation routing algorithm on the $n \times n$ mesh under the standard model whose complexities are $T = O(n \cdot \log n)$, $Q = O(1)$, $S = O(\log n)$ and $D = O(\log n)$.

7.5 An oblivious, weakly-dimensional, $O(n)$-step algorithm

In this subsection we use the results of subsection 7.3 to obtain an oblivious and weakly-dimensional (and thus also minimal) permutation routing algorithm on the $n \times n$ mesh under the standard model with time complexity $O(n)$. Note that, in comparison to algorithm StronglyDimensional2, the time complexity is reduced from $O(n \cdot \log n)$ to $O(n)$, but the algorithm is neither strongly-dimensional nor source-oblivious.

The algorithm works in two phases: In the first phase it routes packets $p$ with $d_x(p) \geq d_y(p)$ and in the second
phase it routes the other packets. We present only the first phase; the second one is similar and is obtained by exchanging the roles of the vertical and horizontal axes. The first phase is strongly-dimensional but the combined algorithm is not strongly-dimensional, due to the exchange of the axes.

For the sake of simplicity, in this subsection $n$ denotes an integer which is a power of 2. The extension of our result to other values of $n$ is straightforward.

**Segmented routing on the $m \times n$ mesh in $O(n)$ steps.** Consider the problem derived from the Segmented-2-Dimensional-Problem by adding the requirement that $d_v(p) \geq d_y(p)$ for all packets $p$. The following two-phase algorithm, GapRouting, solves this problem. In the first phase it partitions the mesh to horizontal slices whose height is $2n$, except of the topmost slice whose height is at most $2n$, and it invokes GapRouting on each of these slices in parallel. In the second phase it removes the lowest $n$ rows of the mesh and applies the first phase on the resulting mesh.

Note that, in comparison with GapRouting, the time complexity is reduced from $O(n + m)$ to $O(n)$; i.e., it now depends only on the horizontal axis. Clearly, this is impossible without the $d_v(p) \geq d_y(p)$ requirement.

**The final algorithm.** Let $M$ be an $m \times n$ mesh and let $0 \leq r_1 < r_2 \leq 1$ be rational numbers s.t. $r_1 \cdot n$ and $r_2 \cdot n$ are integers. Then $M(r_1, r_2]$ denotes the vertical slice of $M$ consisting of columns $r_1 \cdot n + 1$ up to $r_2 \cdot n$. (The first column is column 1.)

The following algorithm, MinObl, solves a Permutation-Routing-Problem on an $m \times n$ mesh s.t. $d_v(p) \geq d_y(p)$ for all packets $p$.

**Theorem 3:** There exists an oblivious and weakly-dimensional permutation routing algorithm on the $n \times n$ mesh under the standard model whose complexities are $T = O(n)$, $Q = O(1)$, $S = O(\log n)$ and $D = O(\log n)$.

### Algorithm MinObl on an $m \times n$ mesh $M$

If $n < 8$

1. Greedily route the packets along strongly-dimensional paths.

Else

1. Route $M(0; \frac{3}{8})$ to $M(\frac{1}{2}; 1)$ packets via GapRouting.
2. Route $M(0; \frac{1}{2})$ to $M(\frac{5}{8}; 1)$ packets via GapRouting.
3. Invoke the algorithm recursively on $M(0; \frac{1}{2})$ and $M(\frac{5}{8}; 1)$, in parallel.
4. Invoke the algorithm recursively on $M(\frac{5}{8}; \frac{3}{8})$. 

See figure 7.

Standard calculations imply that the algorithm, when executed on the $n \times n$ mesh, takes $O(n)$ steps. This implies the following theorem:

**The Ignorant Vertical Routing Problem**

Let us return to the Ignorant-Vertical-Routing-Problem of section 5. We show in this section that this problem is unsolvable. In fact, we prove a much stronger result as follows. This problem is unsolvable even by a centralized algorithm that controls the entire network; moreover, it is unsolvable even by a strategy whose decisions are not required to be computable. In other words, this problem is unsolvable just because one can not predict the future.

**The Overflow Game.** We translate the Ignorant-Vertical-Routing-Problem into a complete information, two-player game: The Overflow Game with parameters $(n, c, k)$, denoted OFG($n, c, k$), is as follows: The game is played on the $D_n$ network (of section 5). The two players, the injector and the mover, take alternating turns.

The mover, in his turn, may move packets along the edges of the array; these movements are restricted as in the Store-and-Forward model – at most one packet may traverse an edge and at most one edge may be traversed by a packet.

The injector, in his turn, may inject packets into the array; all these packets are destined for the last node, $v_n$. In the initial state there are no packets in the network; the first turn is of the mover.

The aim of the injector is to overflow some node with $k$ (or more) packets while the aim of the mover is to prevent this. An unrestricted injector can clearly flood the array with packets. Hence, to win the game, the injector has to create some restrictions depending on $c$. 

Figure 7: The MinObl algorithm
A step of the game is composed of a turn of the mover followed by a turn of the injector and it corresponds to a step in our Store-and-Forward model.

Recall that $J$ is the set of edges from the injector to the network. The injector is bounded by a real number $c$ if there exists a function $a: J \to \mathbb{R}^+$, s.t. the injector is bounded by $a$, and $\sum_{e \in J} a(e) \leq c$. The injector is bounded by $c$ up to step $t$ if it is bounded by $c$ provided that no packets are injected after step $t$.

The injector wins the game in one of its turns iff:

**W1.** Some node has at least $k$ packets at the end of his turn.

**W2.** The injector is bounded by $c$ up to this step.

If the injector never wins, then the mover wins; the game is infinite in this case.

Clearly, a routing algorithm for the Ignorant-Vertical-Routing-Problem with queues bounded by $k$ implies a winning strategy of the mover for that $k$. This contradicts the following theorem.

**Theorem 4:** For any $k$ and $c > 0$, the injector has a winning strategy in $\text{OFG}(n, c, k)$ for all large enough $n$.

To prove the theorem we use a variant of the game as follows.

**The Segment-Overflow-Game.** A segment is a set of consecutive nodes. For a positive integer $m$, an $m$-segment is a segment of size $m$. The Segment-Overflow-Game with parameters $(n, c, k, m)$, denoted $\text{SOG}(n, c, k, m)$, is derived from $\text{OFG}(n, c, k)$ by replacing condition (W1) above with the following one:

**W1'.** Some $m$-segment has at least $m \cdot k / 2$ packets at the end of his turn.

That is, the objective of the injector here is to overflow an $m$-segment, rather than just a single node.

**Lemma 4:** For any $k$, $m$ and $c > 0$, the injector has a winning strategy in $\text{SOG}(n, c, k, m)$ for all large enough $n$.

Theorem 4 clearly follows from the above lemma: a winning strategy in $\text{SOG}(n, c, 2k, l)$ is a winning strategy in $\text{OFG}(n, c, k)$.

**Proof:** For a segment $s$ and a positive integer $t$ let $B(s, t)$ denote the average number of packets in the nodes of $s$ at the end of step $t$, and let $B'(s, t)$ be the average number of packets in the nodes of $s$ at the end of the turn of the mover in step $t$.

Consider the following assertion:

for any $m$ and $c > 0$, the injector has a winning strategy in $\text{SOG}(n, c, k, m)$ for all large enough $n$.

We prove, by induction on $k$, that the assertion is true for every $k$. The case $k = 0$ is trivial. Assume that the assertion is true for some $k$. Let $m' = 4m \cdot \lfloor 1/c \rfloor$, $c' = c / 2$ and let $n$ be large enough so that the injector has a winning strategy for $\text{SOG}(n, c', k, m')$. We prove that the following two-stage strategy is a winning strategy for the injector in $\text{SOG}(n, c, k + 1, m)$:

1. Apply the winning strategy for $\text{SOG}(n, c', k, m')$ until the injector wins, say at step $t$.

2. After step $t$ do the following. Let $l = 2m \cdot \lfloor 1/c \rfloor = m'/2$. Inject no packets until step $t + l$. In step $t + l$ choose an $m$-segment $s$ such that $B'(s, t + l)$ is maximal, and inject a packet into every node of $s$.

We now prove that the injector wins at step $t + l$ or earlier.

Since the injector wins $\text{SOG}(n, c', k, m')$ at step $t$, there is an $m'$-segment $s'$ with $B(s', t) \geq k / 2$. In $l$ consecutive time steps at most $l$ packets can exit a segment of the array, hence $B'(s', t + l) \geq k / 2 - 1/m' = (k - 1)/2$. As $m'$ is a multiple of $m$, there is an $m$-segment $r$ with $B(r, t + l) \geq (k - 1)/2$. Thus, from the criteria used to choose $s$ it follows that $B(s, t + l) \geq (k - 1)/2 + 1 = (k + 1)/2$.

It remains to show that, under this strategy, the injector is bounded by $c$. There is a function $a': J \to \mathbb{R}^+$ s.t. the injector is bounded by $a'$ up to step $t$ and $\sum_{e \in J} a'(e) \leq c'$.

Let $D$ be the set of edges from the injector to $s$. Define a function $a: J \to \mathbb{R}^+$ by

\[ a(e) = \begin{cases} \alpha'(e) & \text{if } e \notin D \\ \max \{\alpha'(e), 1/l\} & \text{if } e \in D \end{cases} \]

This $a$ satisfies:

\[ \sum_{e \in J} a(e) \leq \sum_{e \in J} a'(e) + \frac{m}{l} \leq \frac{c}{2} + m \cdot \frac{1}{2m} \cdot \frac{1}{\lfloor 1/c \rfloor} \leq c. \]

Thus, it remains to show that the traffic on any edge $e \in J$ is bounded by $a(e)$. This clearly holds for $e \notin D$. Let $e \in D$. Packets are injected through $e$ after step $t$ only once, at step $t + l$. Let $f(e, l)$ denote the number of packets traversing $e$ during interval $I$. We need to show that $f(e, l) \leq [a(e) \cdot l]$ for any interval $I$. The hard case is when $l = \lfloor t'', t + l \rfloor$ and $t'' \leq t$. In this case:

\[ f(e, l) = f(e, (t', t'')) + f(e, (t, t + l)) \leq [a'(e) \cdot (t - t')] + 1 = [1 + a(e) \cdot (t - t')] \]

\[ = [l/l + a(e) \cdot (t - t')] \leq [a(e) \cdot (t + l - t')] = \frac{1}{1/a(e)} \leq [a(e) \cdot (t + l - t')]. \]

The last inequality follows from $1/l \leq a(e)$. Thus the injector is bounded by $a$.

**Acknowledgment**

We wish to thank Adi Rosen for helpful discussions.
References


