

Simple Algorithms for Routing  
on Butterfly Networks with  
Bounded Queues

Bruce M Maggs Ramesh K Sitaraman

Presented by Bai

Apr. 13, 2000

## Definitions

- Greedy queuing protocol: At each step, each switch with one or more packets in its queue selects one packet and then sends it to the next level, unless the queue for the packet wishes to enter is already full. A switch is prohibited from sending more than one packet per step, provided that they use different edges.
- Non-predictive protocol: At each step, each switch selects one packet from its queue without examining the destinations of any of the packets in its queue and sends the packet to the next level, unless the queue for the packet wishes to enter is already full. If so, the switch selects the same packet at the next step.
- Network is lightly loaded: If each input of an N-input butterfly sends one packet.
- Network is fully loaded: If each input of an N-input butterfly sends  $\log_2 N$  packets.

## Results

- For any greedy queuing protocol, routing a problem on a lightly-loaded  $N$ -input butterfly  $O(\log N)$  steps, with high probability, provided queue size is a sufficiently-large fixed constant.
- For any deterministic non-predictive queuing there exists a one-to-one routing problem (per that requires  $O(q \log N)$  time to route, where  $q$  is maximum queue size.

## Results

- Routing algorithms that drop packets when there is contention:
  - For a random problem the number of packets that succeed in locking down paths from their origins to their destinations is  $\Theta(N/\log^q N)$ , with high probability, where  $q$  is the maximum number of packets that any wire can support.
  - For any fixed permutation the expected number of packets that reach their destinations is  $\Theta(N/\log^q N)$ .
- A simple, but non-greedy algorithm for routing a permutation problem on a fully-loaded  $N$ -input butterfly network with bounded-size queues in  $\Theta(\log N)$  steps, with high probability.

## Algorithms that drop packets

Two methods of routing:

1. Store-and-forward manner. At every time step, each switch examines the head of its input queue and forwards a packet to the appropriate output queue. If the queue receiving packets is full, then it drops packets.
2. Circuit-switching. Each packet tries to lock down a path between its source and destination. Each edge of the butterfly can sustain up to  $q$  such paths. If more than  $q$  packets want to use the same edge, only  $q$  can succeed, and the rest fails to lock down the path. Packets that succeed in locking down a path are then transmitted along these paths to their respective destinations.

## Algorithms that drop packets

- The idea of routing any fixed permutation is based on the idea of routing to random intermediate destinations.
- Two back-to-back butterflies: two butterflies with  $\log N$  nodes are identified.
- The source of the packet is level 0 of one of the butterflies and each packet has a destination at level  $\log N$  of the other butterfly.
- Two stages algorithm:
  1. Packets route from level 0 of the first butterfly;
  2. Packets route from level 0 of the second butterfly;

## Algorithms that drop packets

- In the first stage, at every step, a switch receives two packets, one from each of its edges. The switch selects a random outgoing edge for one of the packets and routes the other packet to the remaining outgoing edge. In this way, packets are routed to random but not independent destinations. Packets are dropped in this stage.
- In the second stage, packets are dropped according to different rules. Each packet picks a random rank  $r$  to  $r$ , where  $r \in [1, N]$ . When packets must be dropped, packets with the least rank are dropped in favor of packets with a higher rank.

## Algorithms that drop packets

Lemma 1. The probability that any  $k$  packets all pass through a node  $n$  in the  $i$ -th second butterfly is at most  $\frac{1}{2^{1k}}$ .

Proof. Let the sub-butterfly from the  $i$ -th first butterfly that contains  $n$  be  $B$ , where  $n$  is the node in  $B$  that corresponds to node  $n$ .

- $k$  packets pass through  $B$  if and only if all of them pass through some node of  $B$ .
- No two paths that are from the sources of  $k$  packets intersect. If so, then the probability of each packet hitting  $B$  is independent of the others and it is equal to  $\frac{1}{2^{1k}}$ .
- Thus the probability of all  $k$  packets hitting  $B$  is  $\frac{1}{2^{1k}}$ .

## Algorithms that drop packets

Theorem 1 The expected number of packets that reach their destination is  $O(N^{1/d})$ .

Proof Evaluate the probability of a particular packet  $p$  on a  $2^{\text{nd}}$  butterfly reaches its destination.

- The probability that  $p$  has the highest rank  $r$  is  $\frac{1}{r}$
- $p$  can be dropped only if there is a node on its path at least  $q$  (full) packets going through it, all of which have a higher rank than  $p$ .
- $E_q$ : expected number of packets incident on a node at level  $l$  of a  $2^{\text{nd}}$  butterfly  $E_q = \sum_{r=1}^l \binom{2^l}{q} \frac{1}{2^{ql}} \leq \frac{1}{q!}$ . (Lemma 1)
- Expected total number of packets incident on some node on the path is  $\log M/d$
- Expected number of collisions with all packets having rank  $r$  is at most  $\log M / (q! r) = 1/q!$ , since  $\frac{1}{q} \log M = \log$

## Algorithms that drop packets

- The probability that ~~no packet exists~~ ~~no packet exists~~ anywhere on the path of  $p$  is at least  $(1-1/q!)$ .
- $p$  can reach its destination with a probability  $(1-1/q!)(1/r)$ .
- Expected number of packets to reach their destination is at least  $(1-1/q!)(N/\log q)$ .

## A Simple Algorithm

- Result: With high probability, the algorithm routes packets in  $O(k + \log N)$  time to route packets with random destinations, where  $k$  is the number of packets that originate at each input.
- Algorithm:
  - Each input contributes one packet to each wave.
  - Waves of packets are separated by waves of tokens.
  - A token is placed between each pair of successive packets, and after the last packet.
  - The  $i$ th packet is assigned to wave  $2i$  and is assigned to wave  $2i+1$ .
  - For  $i < j$ , no packet or token leaves a switch before any packet or token in
  - $\theta$ -edge  $\langle 1, 0, c_1, \dots, c_{\log N-1} \rangle$  -----  $\langle 1, c_{\log N-1} \rangle$ .
  - $1$ -edge  $\langle 1, 0, c_1, \dots, c_{\log N-1} \rangle$  -----  $\langle 1, c_{\log N-1} \rangle$ .
  - By "forward" a packet or token, we mean send to the appropriate queue in the next level. If that queue is full, the switch tries again in the next step until it is not.

## A Simple Algorithm

- In 0-mode, a switch forwards packets in 0-queue until token is at head of 0-queue. Then it changes to 1-mode.
- In 1-mode, a switch forwards packets in 1-queue until token is at head of 1-queue.
- Switch waits until both queues at its outgoing room to receive a token and sends one token to them. Then it changes back to 0-mode.

Lemma 2. If some packet arrives at its destination  $\log N + d$  or later, then a  $(d, d)$ -delay sequence must have occurred, for some  $d \geq 0$ . Furthermore, no two tokens in the sequence belong to the same wave.

Proof This proof is very similar to our lectures.

- A  $(w, f)$ -delay sequence consists of four components:
  - 1. A path  $P$  from an output to an input, consisting of a sequence of  $w$ , not necessarily distinct switches which appear on  $P$ ;
  - 2. a sequence  $\mathcal{P}$  of  $w$  distinct packets and
  - 3. a non-increasing sequence of wave numbers  $r$

## A Simple Algorithm

- Forward edge path traces an edge from level  $l$  to level  $l+1$ .
- The length,  $L$ , of  $P$ :  $L = \log N + 2f$ , where  $f$  is number of forward edges in path  $P$ .
- A delay sequence occurs: packet belongs to  $h$  wave  $w_i$  and passes through switch  $s_i$  on level  $l$ .
- Lag of a switch  $s$  at time  $t$  is  $L$  on level  $l$ .
- Rank of a packet is a 2-tuple: (wave number, switch number). Each packet has a distinct rank.

First suppose some packet  $p$  arrives at its destination, at step  $t_1$ , where  $t_1 \geq \log N + d$ .  $p$  has lag at least  $d$  at step  $t_1$ . Then construct the delay sequence by spending lag  $d$ . Follow  $p$  back in time until the step at which it was delayed.

## A Simple Algorithm

Lemma 3. If a  $(d + (q - 2b)f)$ -delay sequence occurs, then a  $(bg, f)$  bunched delay sequence occurs, where

$$g = \left\lceil \frac{d + (q - 2b)f - bk - (b-1)\log N}{b} \right\rceil$$

Proof Packets are partitioned into bunches of size  $b$  such that all packets in each bunch pass through the same switch on the sequence and have the same wave number. We will find out a bunched delay sequence.

- Start at switch  $s$  in the sequence, form a bunch of  $b$  packets with wave number  $2(k-1)$ .
  - If successful, form another one.
  - If not, discard those packets and begin forming bunches with next smaller wave number if there are  $k(b-1)$  packets that pass through this case, discard at most  $k(b-1)$  packets; (2) Move on next switch if there are  $k(b-1)$  packets in this case, discard at most  $(\log N + 2f)(b-1)$  packets.
- The delay sequence contains at least  $k(b-2)$  packets and we discard at most  $k(b-1) + (\log N + 2f)(b-1)$  packets. So we can get  $g$  easily.

## A Simple Algorithm

**Theorem 2** For any  $\epsilon, c$  there exist constants  $q > 0$  such that the probability that any packet is delayed more than  $c_2(k + \log N)$  steps is at most  $\epsilon$ , where  $N$  is the number of packets per input of the butterfly.

**Proof** By Lemma 2 & 3, if some packet has delay  $(d + (q + f))$ -delay sequence, and a bunched delay sequence must occur.

- Number of different bunched delay sequence is

$$4^L \cdot \binom{L+g}{g} \cdot \binom{g+k}{g} \cdot \prod_{i=1}^g \binom{2^{d_i}}{b}$$

--There are  $N$  choices for  $i$  the output switch

--For each of  $L$  switches, there are at most

$\binom{L+g}{g}$ : choose  $g$  switches from  $L$ ;

$\binom{g+k}{g}$ : choose wave number for  $g$  bunches;

$\binom{2^{d_i}}{b}$ : for a switch with  $d_i$  choose  $b$  packets.

## A Simple Algorithm

- The probability that all of packets pass through corresponding switches, is  $\prod_{i=1}^g \frac{1}{2^{bd_i}}$ , since each of  $b$  packets in  $i$ th bunch has probability  $\frac{1}{2^{bd_i}}$  of passing through any particular switch on level  $i$ .
- So probability of any delay sequence occurs is:

$$\begin{aligned}
 & \cdot 4^L \cdot \binom{L+g}{g} \cdot \binom{g+k}{g} \cdot \prod_{i=1}^g \binom{2^{d_i}}{b} \cdot \prod_{i=1}^g \frac{1}{2^{bd_i}} \\
 & \leq 2^{3 \log N + 4f} \cdot (e^{L+g}/g!) \cdot (e^{g+k}/g!) \cdot (1/b!)^g \\
 & \leq (8e^2/b)^g \cdot 2^g
 \end{aligned}$$

By making  $g$  large enough, we can make this product less than  $1/N$  for any constant  $c$ .