4 Basic routing theory II – Adaptive routing

In this section we look at adaptive routing, i.e. the path taken by a packet may not only depend on its source and destination but also on the current situation in the network. Adaptive routing has several advantages over oblivious routing:

- It is possible to achieve good routing results with *deterministic* adaptive routing strategies, whereas we know from the Borodin-Hopcroft lower bound that deterministic oblivious routing strategies may perform poorly in certain situations.
- Many adaptive routing strategies can adapt quickly to faulty nodes or edges in the network, whereas in oblivious routing faulty nodes or edges may disconnect certain source-destination pairs and therefore may require the computation of new path system, which may be expensive.
- The congestion of adaptive routing strategies can be better than what can be achieved by oblivious routing strategies. (Note the lower bound of $\Omega(C^P_{\mathrm{OPT}} \log n)$ for oblivious routing on the mesh.)

However, adaptive routing usually also has some weaknesses:

- A high freedom of choosing a path for a packet may cause high delays for the packets, since it may take a while until the algorithm converges towards good paths.
- Adaptive routing may create additional communication (because many of them need control packets) and may have much higher demands on the hardware and software than oblivious routing (see the algorithms below).
- The analysis of adaptive routing is usually much harder than for oblivious routing. Therefore, we will mostly restrict ourselves to presenting algorithms in this section.

Adaptive routing usually works on a packet-based scale, and every packet may follow a different path. Therefore, it is often not possible to simplify the problem of routing information when using an adaptive algorithm to a problem of shipping flow through the network as we did it for oblivious routing. Instead, we will assume in the following that information is sent in the form of unit-size packets. Every node has so-called *buffers* in which packets in transit can be stored. Once a packet reaches its destination, it is absorbed. To keep our models simple, we assume that every edge can transmit only one packet in each direction in a time step.

Several adaptive routing approaches have been suggested in the past. Some of them are:

- *Adaptive routing based on path systems:* Here a path system similar to an oblivious path system may be given. However, instead of randomly selecting a path for a packet from its options, a path may be chosen based on the current situation in the network.
- *Adaptive routing based on edge costs:* In this case every edge is associated with a cost, and the goal is usually to select a path for a packet with minimal cost. This is the standard approach of routing protocols used in the Internet (such as RIP and OSPF).

- *Adaptive routing based on local information:* Here, no path system or edge costs are known. Each node only uses local information (such as the packets currently residing at a node) from itself and its direct neighbors when deciding which packet to transmit along an edge. This is certainly the hardest of the three cases, because no global guidance such as path systems or edge costs is available.

We will present an algorithm for each of these cases.

4.1 Adaptive routing based on path systems

Suppose we want to use the d-dimensional butterfly network as an indirect communication network with the nodes at layer d being the inputs and the nodes in layer 0 being the outputs. Then we know from the Borodin-Hopcroft lower bound that a permutation can be constructed so that $\Omega(\sqrt{2^d})$ paths go through a single node. One way to improve this is to allow more than one edge going to each half of the butterfly (see Figure 1). Then a system of many optional paths can be established for each source-destination pair. One may then ask whether this would suffice to get the congestion down to a level similar to Valiant's trick. However, one can show that using δ edges instead of one can still create a congestion of $\Omega(\sqrt{2^{d/\log 2\delta}})$. The question is whether adaptive strategies would do better in this situation, and we will demonstrate that under certain circumstances they do. For this we need some definitions.

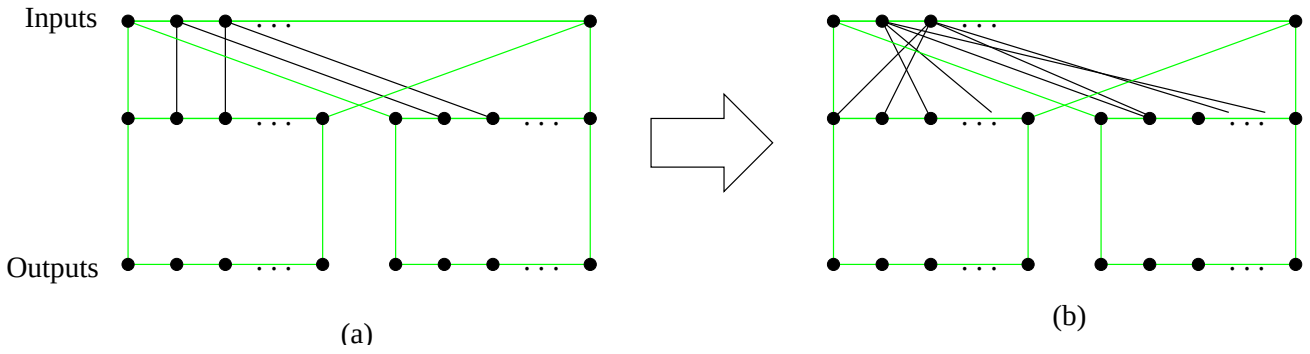

Figure 1: (a) shows a standard butterfly network and (b) shows a variant of it in which every node has several edges to each half.

Definition 4.1 *A bipartite graph $G = (V \cup W, E)$ is called an* $(\alpha, \beta, m, \delta)$-concentrator *if*

1. $|V| = m$ *and* $|W| = m/2$,
2. *the nodes in V have degree at most δ and the nodes in W have degree at most 2δ, and*
3. *for all $U \subseteq V$, $|U| \leq \alpha|V|$: $|\Gamma(U)| \geq \beta|U|$ (recall the definition of Γ in Section 1.1).*

Note that $\alpha \cdot \beta \leq 1/2$, because otherwise property (1) would be violated. Interestingly, there are constructions where for a small enough constant α one can get very close to this inequality [8, 6]. Concentrators allow to construct so-called splitters (see also Figure 2).

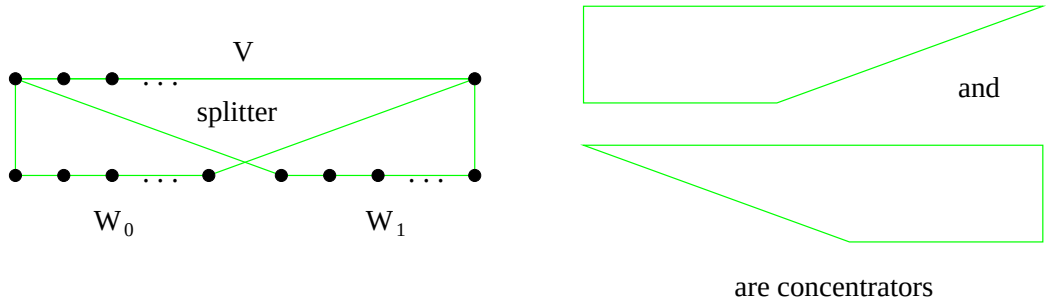

Figure 2: Design of a splitter.

Definition 4.2 *An* $(\alpha, \beta, m, \delta)$-splitter *is a bipartite graph* $G = (V \cup (W_0 \cup W_1), E_0 \cup E_1)$ *in which* $(V \cup W_0, E_0)$ *and* $(V \cup W_1, E_1)$ *represent* $(\alpha, \beta, m, \delta)$*-concentrators. Edges in* E_0 *are called* 0-edges*, and edges in* E_1 *are called* 1-edges*.*

With the help of the splitters we can construct a butterfly with multiple edges to each half, the so-called multibutterfly.

Definition 4.3 *The* d-dimensional multibutterfly $MBF(d, \alpha, \beta, \delta)$ *has* $N = (d+1) \cdot 2^d$ *nodes and degree at most* 4δ*. Figure 3 shows its recursive construction.*

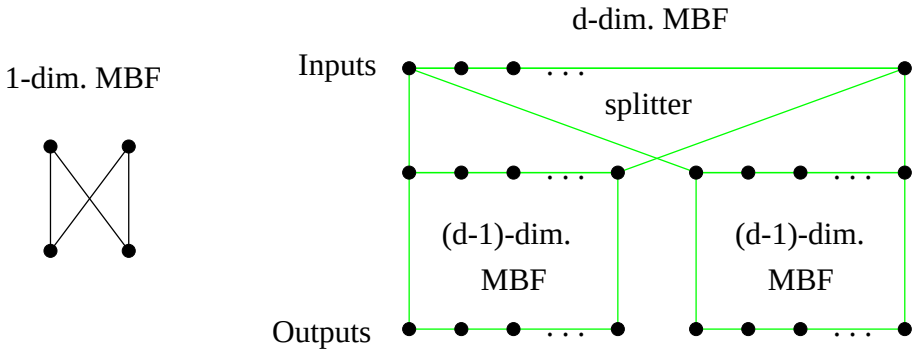

Figure 3: Recursive definition of a multibutterfly.

For our adaptive routing algorithm for the multibutterfly network we assume that every node can store at most one packet. A node that has one packet is called *blocked*. With the help of Hall's marriage theorem (see, e.g. [5]) one can show that every splitter can be partitioned into 2δ matchings $M_1, \ldots, M_{2\delta}$, where each matching M_i is a subgraph of the splitter in which every node has a degree of 1. The algorithm works in rounds. In each round, every node of the multibutterfly works as follows:

- for $i = 1$ to 2δ do:
  Every node that has a packet that has to traverse some s-edge with $s \in \{0, 1\}$ checks whether it has an s-edge in M_i leading to a non-blocked node at the next lower level. If so, it sends the packet to that node.

This protocol (or actually a slight variant of it) has been shown to have the following performance:

Theorem 4.4 ([8]) *For every $MBF(d, \alpha, \beta, \delta)$ with $\beta > 1$ and constant α and δ that are chosen so that there exists an $(\alpha, \beta, m, \delta)$-concentrator for any m it holds: Every permutation routing problem can be routed in $MBF(d, \alpha, \beta, \delta)$ in $O(d)$ steps.*

This is certainly optimal (up to a constant factor), because the distance between every source-destination pair in a d-dimensional multibutterfly is d.

Using simulation strategies of multibutterfly networks on general networks, one can also obtain the following result:

Theorem 4.5 ([2]) *Every constant degree network with unit-capacity edges and flow number F can route every permutation routing problem in time $O(F \log_F n)$ using only constant size packet buffers at every node.*

Note that for $F = n^\epsilon$ for some constant $\epsilon > 0$ (which is the case for all d-dimensional meshes with constant d) the time bound in the theorem reduces to a worst-case optimal $O(F)$ bound.

4.2 Adaptive routing based on edge costs

Here we assume that we have an infinite injection process of packets, and an adversary chooses the injection time, source, destination, and route for each packet. A sequence of injections is called (w, λ)-*admissible* for time windows of size w and injection rate $\lambda < 1$ if in any time interval I of size at least w the total number of packets injected into the network whose paths pass through any edge e is at most $|I| \cdot \lambda$. In this case, the corresponding paths are also called (w, λ)-*admissible*. The condition that $\lambda < 1$ is necessary if we want to avoid deleting packets, because we assumed that every edge can transmit only one packet per time step.

We say that a set of paths is *weakly (w, λ)-admissible* if we can partition the time into windows of length w such that for each window *in the partition* and each edge e, the number of paths that pass through e and correspond to packets injected during the window is at most $w\lambda$.

Suppose now that the adversary does not reveal the paths to the system. In this case, we need an adaptive routing algorithm that selects paths for the packets that hopefully achieve a congestion close to what the adversary can achieve. So, expressed in a formal way, the question is: Is it possible to ensure that if the injected packets have paths that are (w, λ)-admissible, then the adaptive routing algorithm ensures that the selected paths are (W, Λ)-admissible, possibly for a different window size W and a different $\Lambda < 1$?

Based on the ϵ-approximation algorithm by Garg and Könemann [4] for concurrent multicommodity flow problems, which in turn builds upon the work of Plotkin, Shmoys, and Tardos [7], Andrews et al. [1] suggested the following algorithm:

We assume that control information is communicated instantly. Whenever a source node chooses a route for a packet, this information is immediately transmitted to all the links on the route. Whenever the congestion on a link changes, this fact is instantaneously transmitted to all the source nodes. (Andrews et al. also show how to relax these assumptions, but for simplicity we will stick to them here.)

The algorithm by Andrews et al. uses parameters μ, δ, and t that are defined as follows. Let m be the number of links in the network. For any $\Lambda \in (\lambda, 1)$ of our choice, let

$$\mu = 1 - \left(\frac{\lambda}{\Lambda}\right)^{1/3} \tag{1}$$

$$\delta = \left(\frac{1-\lambda\mu}{m}\right)^{1/r\mu} \tag{2}$$

$$t = \lfloor \frac{1-\lambda\mu}{\lambda\mu} \ln \frac{1-\lambda\mu}{m\delta} \rfloor + 1 \tag{3}$$

In the routing algorithm by Andrews et al., every injected packet is routed along the path whose total congestion (i.e. the sum of the congestion of its edges) is the smallest under the current congestion function $c(\cdot)$, i.e. we route along shortest paths with respect to $c(\cdot)$. Initially, the congestion along every link is set to δ, where δ is defined in (2). Each time a route is selected for an injected packet, the congestion $c(e)$ of every link e along the chosen route is updated to $c(e)(1+\mu/w)$, where μ is defined in (1). We reset the congestion of every link to its initial value of δ at the beginning of each *phase*. A phase consists of t time windows of w steps, where t is defined in (3). In each phase, the following strategy is used:

- set $c(e) = \delta$ for all edges e
- for the ith window, $i = 1, \ldots, t$:
  - for each packet injected during the ith window, choose the least congested route p w.r.t. c
  - for all edges in p, set $c(e) = c(e)(1+\mu/w)$

Andrews et al. prove the following theorem about this algorithm.

Theorem 4.6 ([1]) *For all packets injected during one phase, at most $tw\Lambda$ of their routes chosen by the above procedure go through the same link for some $\Lambda < 1$. In other words, these routes are weakly (tw, Λ)-admissible.*

Furthermore, the following lemma implies that the selected paths are not only weakly (tw, Λ)-admissible, but also (w', λ')-admissible for some $w' \geq w$ and $\lambda' \in [\Lambda, 1)$.

Lemma 4.7 ([1]) *If a set of paths is weakly (w, λ)-admissible then it is also (w', λ')-admissible for some $w' \geq w$ and $\lambda' \in [\lambda, 1)$.*

Proof. Suppose the injections are weakly (w, r)-admissible. Then we show that they are (w', λ')-admissible for $\lambda' = (1+\lambda)/2$ and $w' = 4w\lambda/(1-\lambda)$. For any $T \geq w'$, let T be in the range of $[nw, (n+1)w)$ for some integer $n \geq 4\lambda/(1-\lambda)$. Due to weak admissibility and our choices of n, T and λ', the number of injections during T steps for any link e is at most $(n+2)\lambda w$. It holds that

$$(n+2)\lambda w \leq \frac{nw(1+\lambda)}{2} \quad \Leftrightarrow \quad 2\lambda w \leq \frac{nw(1+\lambda) - 2n\lambda w}{2} = n \cdot \frac{w(1-\lambda)}{2} \quad \Leftrightarrow \quad n \geq \frac{4\lambda}{1-\lambda},$$

which is true. Hence,

$$(n+2)\lambda w \leq \frac{nw(1+\lambda)}{2} \leq T \cdot \lambda' \ ,$$

which proves the lemma. ⊓

Hence, as long as it is guaranteed that all injected packets have (w, λ)-admissible routes, the algorithm by Andrews et al. can find (w', λ')-admissible routes for them.

Of course, in the Internet not all of the injected packets can be sent to their destinations. Thus, apart from being able to find good paths, also suitable rate and admission control algorithms have to be found, which was not addressed in [1].

4.3 Adaptive routing based on local information

Finally, we study the situation that nothing is known about the network apart from the information available at a node and its direct neighbors. Imagine now that we have an adversary that can inject an arbitrary number of packets in the system at any time step. In this case, only some of the injected packets may be able to reach their destination, even when using a best possible routing strategy. For each of the successful packets a schedule of the form $S = (t_0, (e_1, t_1), \ldots, (e_\ell, t_\ell))$ can be specified, where t_0 represents the time when the packet got injected and (e_i, t_i) represents the event that the packet crosses edge e_i at time t_i. Of course, schedules are not allowed to have a pair (e, t) in common (unless we allow edges to transport more than one packet at a time).

Each time a packet reaches one of its destinations, we count it as one *delivery*. The number of deliveries that is achieved by an algorithm is called its *throughput*. Since the adversary is allowed to inject an unbounded number of packets, we will allow routing algorithms to drop packets so that a high throughput can be achieved with a buffer size that is as small as possible.

In order to compare the performance of a best possible strategy with our strategy, we will use competitive analysis. Given any sequence of packet injections σ, let $\mathrm{OPT}_B(\sigma)$ be the maximum possible throughput (i.e. the maximum number of deliveries) when using a buffer size of B, and let $A_{B'}(\sigma)$ be the throughput achieved by some given routing algorithm A with buffer size B'. "Maximum possible" means here maximum possible given the same type of resources as used by the optimal algorithm. The only difference between the resources of our algorithm and a best possible algorithm we allow is the buffer size. We call A (c, s)-*competitive* if for all σ and all B,

$$A_{s \cdot B}(\sigma) \geq c \cdot \mathrm{OPT}_B(\sigma) - r$$

for some value $r \geq 0$ that is independent of $\mathrm{OPT}_B(\sigma)$. Note that $c \in [0, 1]$. c represents the fraction of the optimal throughput that can be achieved by A. If c can be brought arbitrarily close to 1, A is also called $s(\epsilon)$-*competitive* or simply *competitive*, where $s(\epsilon)$ reflects the relationship between s and ϵ with $c = 1 - \epsilon$. Obviously, it always holds that $s(\epsilon) \geq 1$, and the smaller $s(\epsilon)$, the better is the algorithm A.

We assume that every node has a buffer for every destination and that the maximum number of packets a buffer can hold is H. Let $q^t_{v,d}$ denote the size of the buffer for destination d in node v at step t. In every time step t the T-balancing algorithm performs the following operations in every node v:

1. For every edge $e = (v, w)$, determine the destination d with maximum $q^t_{v,d} - q^t_{w,d}$ and check whether $q^t_{v,d} - q^t_{w,d} > T$. If so, send a packet with destination d from v to w (otherwise do nothing).

2. Receive incoming packets and absorb all packets that reached the destination.

3. Receive all newly injected packets. If a packet cannot be stored in a node because its height is already H, then delete the new packet.

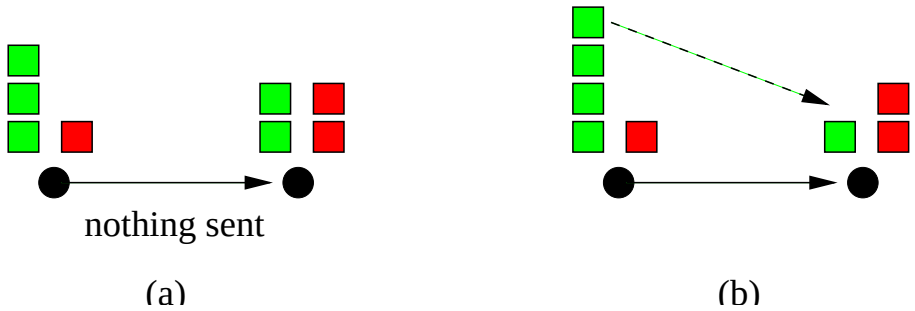

Figure 4: Illustration of T-balancing algorithm with $T = 2$. In (a) the maximum difference in queue size is below T and therefore nothing is sent, whereas in (b) the maximum difference in queue size is above T.

Let $L \leq n - 1$ denote an upper bound on the (best possible) average path length used by the successful packets in an optimal algorithm with buffer size B, and let δ denote the maximum degree of a node in the network. Then the algorithm achieves the following result.

Theorem 4.8 ([3]) *For any $\epsilon > 0$ and any $T \geq B + 2(\delta - 1)$, the T-balancing algorithm is $1 + (1 + (T + \delta)/B)L/\epsilon$-competitive.*

Hence, in the case that $B \geq \delta$, the T-balancing algorithm with $T = B + 2\delta$ is $O(L/\epsilon)$-competitive. Finally, we demonstrate that the analysis of the T-balancing algorithm is essentially tight, even when using just a single destination.

Theorem 4.9 *For any $\epsilon > 0$, $T > 0$, and $L \geq 1$, the T-balancing algorithm requires a buffer size of at least $T \cdot (L - 1)/\epsilon$ to achieve a more than $1 - \epsilon$ fraction of the best possible throughput.*

Proof. Consider a source node s that is connected to a destination d via two paths: one of length 1 and one of length $(L - 1)/\epsilon$. Further suppose packets are injected at s so that $1 - \epsilon$ of the injected packets have a schedule along the short path and ϵ of the packets have a schedule along the long path. Then the average path length is $1(1 - \epsilon) + ((L - 1)/\epsilon) \cdot \epsilon \leq L$. Since each time a packet is moved forward along a node its height must decrease by at least T, a packet can only reach the destination along the long path if s has a buffer of size $H \geq T \cdot (L - 1)/\epsilon$. Hence, such a buffer size is necessary to achieve a throughput of more than $1 - \epsilon$. ⊓

References

[1] M. Andrews, A. Fernández, A. Goel, and L. Zhang. Source routing and scheduling in packet networks. In *Proc. of the 42nd IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 168–177, 2001.

[2] F. M. auf der Heide and C. Scheideler. Deterministic routing with bounded buffers: Turning offline into online protocols. In *Proc. of the 37th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 370–379, 1996.

[3] B. Awerbuch, A. Brinkmann, and C. Scheideler. Anycasting and multicasting in adversarial systems. Unpublished manuscript, March 2002.

[4] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proc. of the 39th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 300–309, 1998.

[5] F. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays · Trees · $Hypercubes$*. Morgan Kaufmann Publishers (San Mateo, CA), 1992.

[6] M. Morgenstern. Natural bounded concentrators. *Combinatorica*, 15(1):111–122, 1995.

[7] S. Plotkin, D. Shmoys, and E. Tardos. Fast approximation algorithms for fractional packing and covering problems. In *Proc. of the 32nd IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 495–504, 1991.

[8] E. Upfal. An $O(\log n)$ deterministic packet routing scheme. *Journal of the ACM*, 39:55–70, 1992.