Networks on which hot-potato routing does not livelock [*]

Uriel Feige and Robert Krauthgamer
Department of Computer Science and Applied Math
Weizmann Institute of Science
Rehovot 76100, Israel
{feige,robi}@wisdom.weizmann.ac.il

September 13, 1999

Abstract

Hot-potato routing is a form of synchronous routing which makes no use of buffers at intermediate nodes. Packets must move at every time step, until they reach their destination. If contention prevents a packet from taking its preferred outgoing edge, it is deflected on a different edge. Two simple design principles for hot potato routing algorithms are *minimum advance*, that advances at least one packet towards its destination from every nonempty node (and possibly deflects all other packets), and *maximum advance*, that advances the maximum possible number of packets.

Livelock is a situation in which packets keep moving indefinitely in the network without any packet ever reaching its destination. It is known that even maximum advance algorithms might livelock on some networks. We show that minimum advance algorithms never livelock on tree networks, and that maximum advance algorithms never livelock on triangulated networks.

1 Introduction

A network of processors is modeled as a graph in which the processors are nodes and communication links are edges. We assume here that communication links are bidirectional, which gives rise to an undirected graph. Processors send messages to each other as packets. Packets follow a path through the network from their source node to their destination node. We assume that time proceeds in discrete time steps synchronized throughout the network, and that a packet can cross one edge per time-step. We further assume that each edge can carry at most one packet per time step in each direction.

A routing algorithm specifies for every node at every time step what to do with the packets currently at that node. That is, it determines for each packet, based on the local state of the node and on information contained in the header of the packets, whether to absorb it (we assume this happens if and only if the current node is the final destination of

[*] This is a preprint of an article accepted for publication in Distributed Computing© (copyright owned as specified in the journal).

the packet), whether to keep it buffered at the node for the current time step, or whether to send it out on an outgoing edge, respecting the limitations on edge capacity.

We study a form of packet routing algorithms known as *hot potato* routing (which is a special form of *deflection routing*). It uses no buffer space for storing delayed packets. Each packet must leave the node at the step following its arrival, unless it has arrived to its destination. Thus, packets keep moving in the network, giving rise to the term "hot-potato". In this form of routing some packets may have to be deflected away from their preferred direction due to congestion. This is different from store-and-forward routing, where a packet can be temporarily stored at a processor, and forwarded along the desired link once it becomes available.

We consider *shortest path* hot potato algorithms, where every packet attempts to follow a shortest path from its current location to its destination. Due to contention (two or more packets who wish to traverse the same edge at the same time), this may not be always possible. In the current paper we study the following contention resolution rules (similar to rules considered in [12, 7, 6]). These rules apply at every time step and every non-empty node.

- **Minimum Advance:** At least one packet advances towards its destination.
- **Maximum Advance:** The maximum possible number of packets advance.

The above rules provide a tradeoff between the progress guarantee that they give and the difficulty of implementing them. Minimum advance algorithms are easy to implement. Every node selects an arbitrary incoming packet and sends it out on its desired outgoing edge. All other packets are sent out on arbitrary edges, possibly deflected further away from their destinations. Maximum advance algorithms are more difficult to implement, as they require finding a maximum matching between the set of incoming packets and their desired outgoing edges. However, they do not deflect packets unless the deflection cannot be avoided, and hence are expected to make more progress at every time step.

Observe that even with maximum advance it is possible that at a certain time step more packets are being deflected than being advanced (e.g., if three packets want to cross the same link). This leads to the issue of livelock, discussed next.

1.1 Livelock

The *evacuation time* of a routing algorithm is the number of time steps it takes it to deliver all packets currently in the network to their respective destinations, assuming that no new packets are injected into the network. A routing algorithm may fail to evacuate a network for the following reasons:

- **Deadlock:** Packets in the network do not move because every packet waits for another packet to free network resources (typically, buffer space at nodes).
- **Livelock:** Packets do move in the network, but never reach their destination nodes.

Deadlock avoidance is a major issue for store-and-forward routing, when buffers at the nodes are of bounded size and nodes refuse to accept new packets unless they have sufficient buffer space for them. It becomes an even more acute danger for wormhole routing, where a single packet can occupy buffers of several nodes. See [11] and references therein for more details. However, for hot potato routing, deadlock cannot occur, because packets keep on moving all the time.

Livelock is usually not an issue for store-and-forward routing, because there it is often the case that a packet moves only if it makes progress towards its destination. However, livelock is a concern for hot potato routing, where packets might alternate between advancing towards their destination and being deflected away from it. Some simple examples of livelock are shown in [15] and [12].

There are certain contention resolution rules that are livelock-free on every network. For example, livelock can be avoided in any algorithm of the minimum advance family, by adding certain priority rules, e.g. priority given to packets that are closest to their destination, or fixed priority levels according to a global ordering of the destination of packets, cf. [10, 12].

Neither the minimum advance principle nor the maximum advance principle by themselves guarantee evacuation on every network. See Section 1.5 for more details. However, they are livelock-free on some networks. Identifying such networks is the motivation for the current paper.

Throughout, we focus on the *batch* (also called *static*) model, where all packets are injected to the network at time 0, as this more restricted model suffices for our purposes of identifying livelock-free networks. Indeed, we show in Section 1.6 that in this respect, the batch model is equivalent to the *continuous* model, in which packets may be generated at any time.

Remark: An important issue that is not addressed in this paper is that of *starvation*. Starvation occurs when some packets fail to reach their destinations indefinitely, whereas other packets do reach their destinations. For batch routing problems, as studied in this paper, either all packets are delivered, or a livelock occurs, making starvation a nonissue. However, if new packets are continuously generated and injected into the network before the older packets evacuate it, starvation may occur even on a livelock free network. A common approach for avoiding starvation in this case is by giving priority to older packets over newer ones.

1.2 Preliminaries

As mentioned above, a network is modeled as an undirected graph. We use the standard notion of a graph $G = (V, E)$ with a vertex set V and an edge set E, where the vertices represent the processors and the edges represent the communication links. In this work we consider only networks (graphs) that are finite, connected and simple.

A (simple) *cycle* of length l in a network is a subset of l distinct vertices $\{v_1, \ldots, v_l\}$ and corresponding edges $(v_1, v_2), (v_2, v_3), \ldots, (v_{l-1}, v_l), (v_l, v_1) \in E$. A *triangle* is a cycle of length 3. A *tree* network is one which contains no cycles. The *girth* of a graph is the length of the shortest cycle in it. Trees have infinite girth.

A *chord* in a cycle is an edge both whose endpoints belong to the cycle, though not adjacent on it. A network is *triangulated* (or *chordal*) if any cycle of length more than 3 has a chord. A good overview of the different aspects of triangulated graphs can be found in Golumbic's book [14], or in the exposition of Kloks [18, Chapter 2.1].

We consider the *batch* routing problem, where all packets are generated at time 0. We assume that the number of packets generated at a node is no more than its degree. Hence all packets can be sent out on outgoing edges at the first time step. There are no further restrictions on the routing problem (e.g. we allow many packets to have the same destination). The goal of the routing algorithm is to evacuate the network, i.e. deliver each of the packets to its destination.

We denote the number of vertices in the network as $n = |V|$, and the number of packets in the routing problem as k. A *configuration* of packets in a network is any placement (mapping) of the packets to the nodes of the network. Let us denote by Γ the set of all possible configurations of packets in the network. Then for k packets in a network with n nodes, $|\Gamma| \leq n^k$.

We note that in defining a configuration of a network, we abstract away past information about the network (e.g., on which link did a packet enter a node). In our context of studying the livelock avoidance properties of minimum/maximum advance algorithms, this is done without loss of generality, for the following reason. At every time step, we assume adversarial decisions: the routing algorithm, having complete knowledge of the location of all packets in the network, assigns packets to the worst possible outgoing edges, with the only constraint being that the minimum/maximum advance principle is respected. As the choice of worst possible outgoing edges is independent of the past, there is no reason for a configuration to encode past information. This also implies that if an algorithm does not livelock, the network is evacuated in no more than n^k steps, as no configuration can repeat itself (using a convention where a packet that reached its destination is "placed" in the configuration at its destination node).

Remark: For networks on which minimum advance algorithms cannot livelock, the discussion above implies that every minimum advance algorithm evacuates them within n^k steps. For other networks, some minimum advance algorithms might livelock, whereas other minimum advance algorithms that do not livelock might take more than n^k steps to evacuate the network. Long evacuation times may happen if routing decisions depend on the past, and hence a configuration (in the sense that we defined it) may repeat without implying livelock. For randomized routing algorithms (such as in [19]), it may happen that livelock avoidance is guaranteed only in a probabilistic sense, and then the expected number of steps it takes to evacuate the network need not be smaller than n^k.

1.3 Results

Our main results are the following:

Theorem 1 *Minimum advance algorithms cannot livelock on trees.*

Theorem 2 *Maximum advance algorithms cannot livelock on triangulated graphs.*

These theorems do not give a complete characterization of networks on which minimum/maximum advance algorithms do not livelock. For example, the minimum advance algorithm also evacuates every complete graph. However, we do give examples showing that our theorems cannot be extended to much larger classes of networks. Perhaps the major open question in this respect is whether the maximum advance algorithm can livelock on the two dimensional mesh.

The proofs of our theorems do not provide upper bounds on the evacuation time (except for the general upper bound of n^k mentioned above). The issue of upper bounds is touched upon in Section 3.1, but remains mostly unexplored.

1.4 Related work

Baran [1] is widely credited with having first proposed hot-potato routing. Borodin and Hopcroft [8] proposed an algorithm for hot-potato routing on the hypercube. Although they did not give a complete analysis of its behavior, they observed that "experimentally the algorithm appears promising". Numerous experimental results on hot-potato routing have been published, and several variants were implemented in massively parallel machines and in high-speed communication networks, especially in optical networks. For more details on experimental and practical issues, see the many references of [7].

Apparently, the first to consider worst case bounds for hot-potato routing was Hajek [15], whose solutions include giving priority to packets closer to their destination. His work was continued by Brassil and Cruz [10], who studied fixed priorities.

Several works consider design principles, i.e. families of algorithms. A certain class of "greedy" algorithms (stronger than minimum advance but weaker than maximum advance) was studied by Ben-Dor, Halevi and Schuster [7]. They give an upper bound for the time it takes any algorithm of this class to evacuate a mesh network. Feige [12] gives an upper bound for the time that an arbitrary maximum advance algorithm evacuates a tree network.

Nontrivial lower bounds on the worst case evacuation time are known for some hot-potato routing algorithms. Feige [12] shows examples in which a packet suffers $2^k - 1$ deflections when there are k other packets in the network, and the routing algorithm is maximum advance with fixed priorities. For the two dimensional mesh, Ben-Aroya, Chinn and Schuster [3] show examples where certain shortest path algorithms have evacuation times that are linear in the number of nodes.

Only few works consider hot-potato routing algorithms for general networks, cf. Feige [12] and, independently, Symvonis [23]. The store and forward case for general networks was addressed by Mansour and Patt-Shamir [20].

Most work on hot-potato routing suggest specific algorithms and analyze their worst case bounds for certain networks such as trees, hypercubes, meshes and tori. Some of these algorithms belong to the family of shortest path algorithms, cf. [4, 5, 7, 9, 12], while others are structured, i.e. use predefined paths or certain structures, cf. [2, 13, 17, 16, 21, 22]. For a recent discussion of these algorithms the reader is referred to [7] or [9].

1.5 An example of livelock

For completeness, we demonstrate a livelock on networks with girth 5 or more (recall that the *girth* of a graph is the length of the shortest cycle in it). Essentially, this example was given in [12] where a few more vertices were added to make the routing problem one-to-one (i.e. each node is the source and destination of at most one packet).

Proposition 1 *There are maximum advance algorithms that livelock on any cycle-network of length at least 5, and on any network with finite girth at least 5.*

Proof. Consider first a cycle of $g \geq 5$ nodes, and number the nodes on it from 0 to $g-1$. Let each node i be the source of two packets, whose destinations are $(i+1) \bmod g$ and $(i+2) \bmod g$. Now attempt to route the packets using a maximum advance routing algorithm, resolving contentions by giving priority to the packet which is furthest away from its destination. Observe, that in the next time step, each node i will have two packets, whose destinations are also $(i+1) \bmod g$ and $(i+2) \bmod g$. This configuration is exactly the initial one, leading to a livelock.

For arbitrary networks with finite girth $g \geq 5$, observe that the same routing embedded on a shortest cycle in the network is a maximum advance one. Indeed, the shortest path between a node i and a destination $(i+2) \bmod g$ must be unique, or otherwise the network would contain a cycle of length 4. Hence, the only edges which advance a packet towards its destination (either $(i+1) \bmod g$ or $(i+2) \bmod g$) are along the cycle edges, so the same routing is a maximum advance one and, of course, livelocks. □

1.6 Continuous Routing

In the *continuous* routing problem, packets may be generated at any time. Note that it might be impossible for a source node to inject the packet into the network if all its outgoing edges are occupied by other packets. We thus need to assume that a source node is able to store the packets it generates until it has a vacant outgoing edge.

We next show that for our purposes of identifying livelock-free networks, the batch and continuous models are equivalent.

Proposition 2 *A network is livelock-free for minimum (resp. maximum) advance algorithms in the batch model if and only if it is livelock-free for minimum (resp. maximum) advance algorithms in the continuous model.*

Proof. We show that a network might have a livelock in the batch model, if and only if it might have a livelock in the continuous model (with respect to the same class of algorithms). Clearly, livelock in the batch model implies livelock in the continuous model. Conversely, assume a livelock in the continuous model occurs, so no packets are delivered to their destination. Since the network is finite, at some point no more packets are injected to the network, and we remain in a batch scenario that livelocks. If the continuous algorithm satisfied the minimum (resp. maximum) advance principle, then so does the batch algorithm. Note that the implied batch algorithm might be nondeterministic even if the continuous algorithm was deterministic, because the continuous algorithm might have used additional information which is not available to the batch algorithm, such as the number of packets that wait to be injected to the network. However, our model of batch routing considers adversarial decisions at every time step, including nondeterministic choices (see Section 1.2). With an appropriate adversary, we get a livelock in the batch model. □

2 Maximum Advance Algorithms

A pair of adjacent edges (i.e. having a common endpoint) in a network is said to satisfy the *shortcut* property if it participates in a triangle of the network.

Proposition 3 *In a triangulated network, every cycle contains at least two pairs of edges with the shortcut property.*

Proof. If the length of the cycle is 3 (a triangle), then every pair of edges from it satisfy the shortcut property with the cycle itself being the required triangle. Otherwise, the length of the cycle is at least 4 so it must have at least one chord. This chord divides the edges of the cycle into two parts, each containing at least 2 edges. To complete the proof, it suffices to show that each of the two parts of the cycle contains at least one pair with the shortcut property.

Consider one part of the cycle, P, of $k \geq 2$ edges, and the dividing chord e. Together they form a cycle of length $k+1$. So if $k = 2$, then P itself is a pair of edges with the shortcut property, as claimed. If $k > 2$, then the formed cycle must have a chord e', and is thus also divided into two parts. Observe that the part not containing e, which we denote by P', is strictly contained in P, i.e. $P' \subset P$ and $2 \leq |P'| \leq |P| - 1$. Repeating this argument at most $k-2$ times, we end up with a subset of size 2 of P, i.e. a pair of edges, which satisfies the shortcut property, because the corresponding chord forms triangle, as required. □

We now prove Theorem 2, that maximum advance algorithms do not livelock on chordal graphs.

Proof. Assume to the contrary that there is a routing execution obeying the maximum advance principle which livelocks, and consider a sequence of advances A, constructed as follows. Start with some packet p advancing in an arbitrary j-th step. If p advances also in the next step (step $j+1$), then add this advance to A. Otherwise (p does not advance), all edges which would advance p are assigned to other packets p', which advance on it (or otherwise swapping between p and p' would contradict the maximum advance principle). In this case, choose one such edge and an advancing packet p' and add it to A. The rest of the advances are defined in the same manner.

Due to the livelock in a finite network, the sequence of advances A must repeat some node, closing a cycle in the network. Note that the cycle is simple if we consider the first time a node occurs twice in the sequence A, and its length is at least 3, by the way it was constructed.

Consider pairs of adjacent edges on the cycle. All of them but one (the pair of the last edge and the first edge in the construction of the cycle), participate in a shortest path of

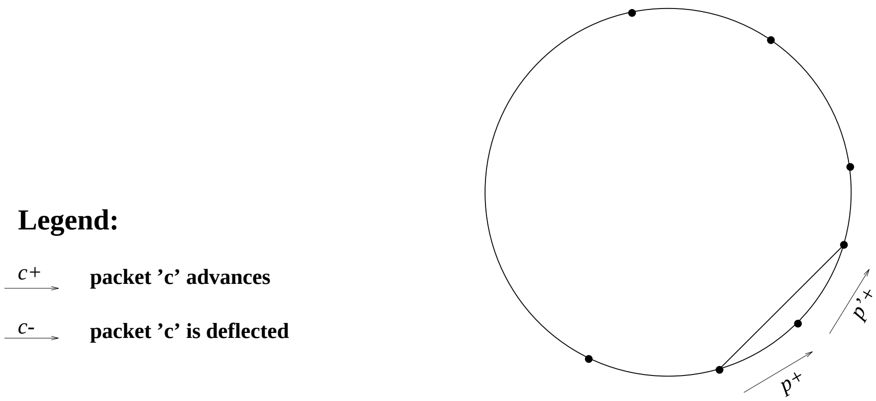

Figure 1: A minimum chord is actually a shortcut for a shortest path

some packet p to its destination, according to our construction of the cycle from the sequence of advances. By Proposition 3 there is a shortcut in the network for at least one of these pairs (see Figure 1). However, such a shortcut edge is in contradiction with the definition of a shortest path. $\square$

3 Minimum Advance Algorithms

We prove Theorem 1, that minimum advance algorithms do not livelock on trees.

Proof. Assume to the contrary that there is a routing execution obeying the minimum advance principle which livelocks. Since the number of possible configurations is finite, then some configuration must be repeated at some time. W.l.o.g. assume that after r time steps the network returns to its initial configuration.

Let p be an advancing packet in an arbitrary j-th time step. Notice that the edge (u, v) on which p advances (from u to v), is a bridge between two subtrees of the tree network, as shown in figure 2. We define the *leftover tree* $T_{p,j}$ to be the subtree which contains v (and also the destination of p).

Consider all leftover trees in the first r steps, and let $T_{\min} = T_{p',j'}$ be the tree of minimum size among them, defined as the leftover tree of a packet p' advancing at time step j'. If T_{min} contains only one node v then necessarily this node is the destination of p', which reaches its destination at this step, in contradiction to the assumption of a livelock.

So we can assume that T_{min} contains at least two nodes. Let $j'' > j'$ be the first step in which p' leaves the leftover tree T_{min}, namely traverses from v to u. There must be such a step because p' must return to u at some time by the livelock assumption. Packet p' is clearly deflected at this step (the edge (v, u) leads it further away from its destination). So by the minimum advance principle some other packet p'' must advance from v at this time step (see figure 3). However, the edge (v, u) is occupied by p', so p'' must advance further

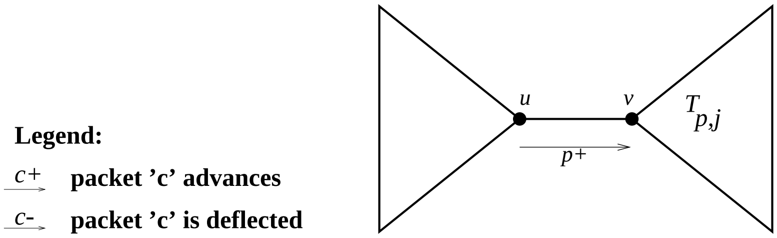

Figure 2: Packet p separates the tree and defines the leftover tree.

into T_{min}. Thus, p'' defines a strictly smaller leftover tree $T_{p'',j''} \subset T_{min}$, which contradicts the minimality of T_{min}. □

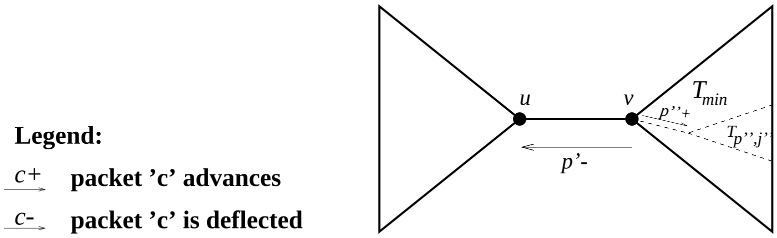

Figure 3: A smaller leftover tree $T_{p'',j''} \subset T_{min}$

Theorem 1 is nearly best possible, as the following proposition shows.

Proposition 4 *A minimum advance algorithm may enter a livelock on a network with one cycle of arbitrary length.*

Proof. For simplicity, we demonstrate a livelock on a network with one cycle of length 3. Examples with a larger cycle are similar. The network and a corresponding step sequence are described in figure 4. Circles denote packets destinations and arrows represent the traversal of a packet.

Observe that the routing satisfies the minimum advance principle but enters a livelock. □

3.1 Evacuation times

Both minimum advance and maximum advance algorithms evacuate any tree network. However, their evacuation time is different. For a packet p, let d_p be the distance between the packet's source and destination nodes, and let k denote the number of packets in the routing problem. It was shown in [12] that every maximum advance algorithm achieves

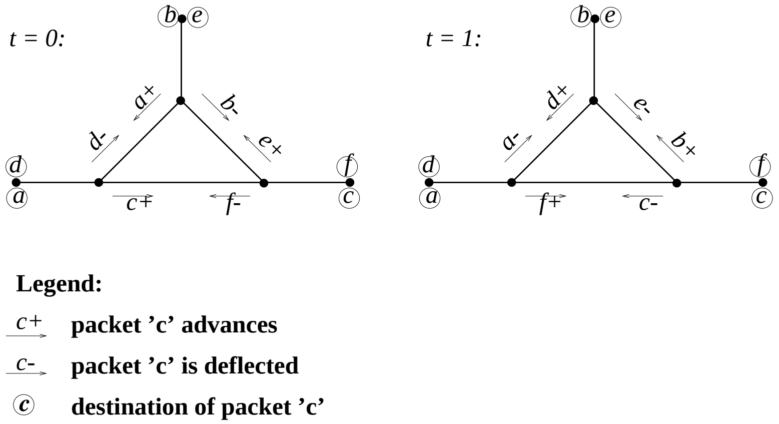

Figure 4: An example of minimum advance livelock on a network with one cycle

$t_p \leq d_p + 2(k-1)$ on trees, where t_p is the number of steps required for the packet to reach its destination. We show that this bound does not hold for some minimum advance algorithms.

Proposition 5 *There is a minimum advance algorithm on a tree network which does not deliver some packet p within $d_p + 2(k-1)$ steps.*

Proof. We show an example of routing 3 packets (denoted a, b and c) on a small tree network of 6 nodes. The network and the corresponding step sequence are described in figure 5. Circles denote packets destinations and arrows represent the traversal of a packet.

Observe that the routing satisfies the minimum advance principle, but although $k = 3$, packet a is deflected 3 times, and is thus delivered only after $d_p + 6 > d_p + 2(k-1)$ steps. □

References

[1] P. Baran. On distributed communication networks. *IEEE Transactions on Communications*, pages 1–9, 1964.

[2] A. Bar-Noy, P. Raghavan, B. Schieber, and H. Tamaki. Fast deflection routing for packets and worms (extended summary). In *Proceedings of the Twelth Annual ACM Symposium on Principles of Distributed Computing*, pages 75–86, Ithaca, New York, USA, 15–18 August 1993.

[3] I. Ben-Aroya, D. D. Chinn, and A. Schuster. A lower bound for nearly minimal adaptive and hot potato algorithms. *Algorithmica*, 21(4):347–376, 1998.

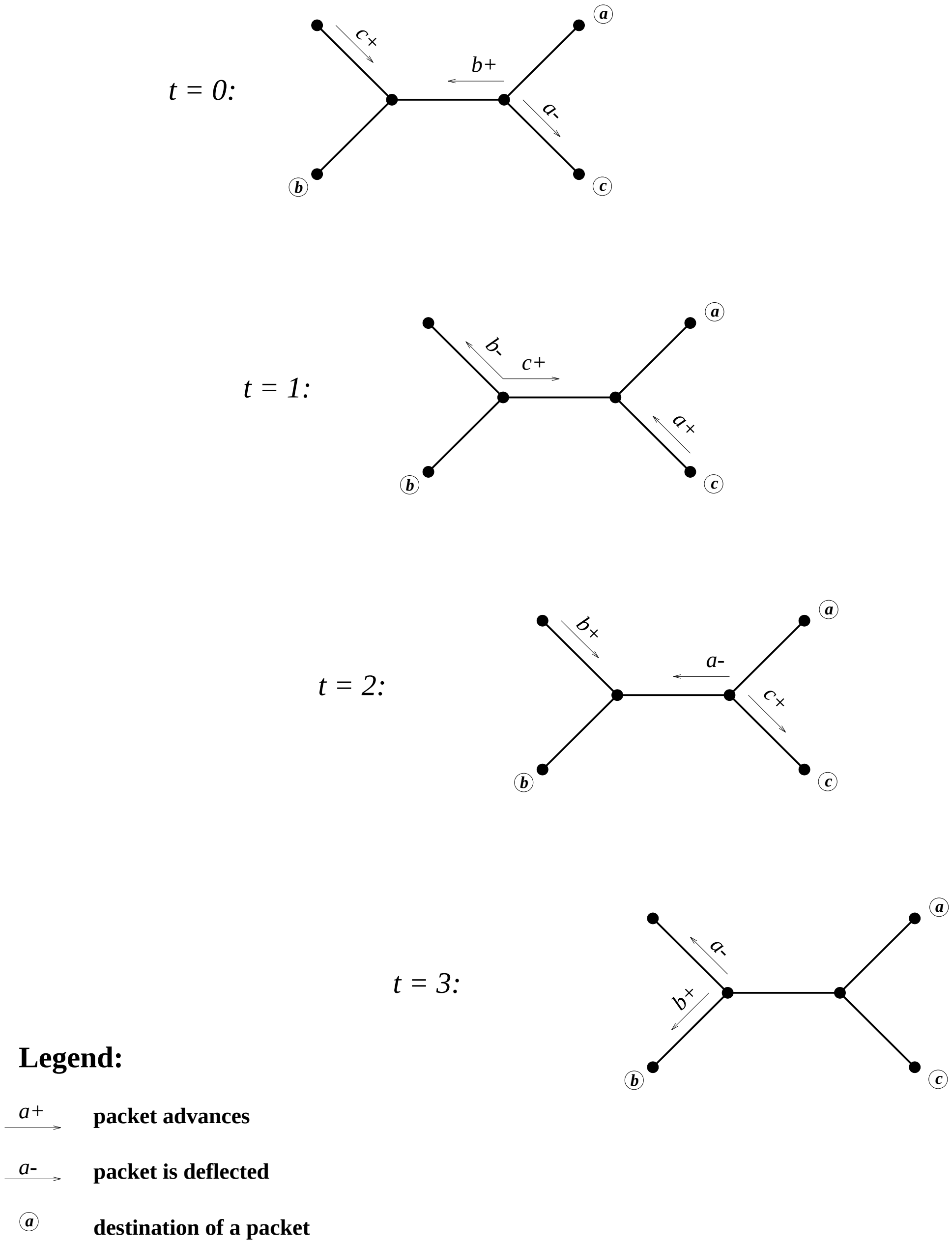

Figure 5: An example of minimum advance routing on a tree network, where delivery of packet a takes more than $d_p + 2(k-1)$.

[4] I. Ben-Aroya, T. Eilam, and A. Schuster. Greedy hot-potato routing on the two-dimensional mesh. *Distributed Computing*, 9(1):3–19, 1995.

[5] I. Ben-Aroya, I. Newman, and A. Schuster. Randomized single-target hot-potato routing. *J. Algorithms*, 23(1):101–120, 1997.

[6] I. Ben-Aroya and A. Schuster. Greedy hot-potato routing on the mesh. *ESA: Annual European Symposium on Algorithms*, 1994.

[7] A. Ben-Dor, S. Halevi, and A. Schuster. Potential function analysis of greedy hot-potato routing. *Theory Comput. Syst.*, 31(1):41–61, 1998.

[8] A. Borodin and J.E. Hopcroft. Routing, merging and sorting on parallel models of computation. *JCSS*, 30:130–145, 1985.

[9] A. Borodin, Y. Rabani, and B. Schieber. Deterministic many-to-many hot potato routing. *IEEE Trans. Parallel Distrib. Syst.*, 8(6):587–596, June 1997.

[10] J. Brassil and R. Cruz. Nonuniform traffic in the Manhattan street network. *Perform. Eval.*, 25(3):233–242, 1996.

[11] J. Duato, S. Yalamanchill, and L. Ni. *Interconnection Networks; An Engineering Approach.* IEEE Computer Society, Washington, 1997.

[12] U. Feige. Observations on hot potato routing. In *Third Israel Symposium on the Theory of Computing and Systems (Tel Aviv, 1995)*, pages 30–39. IEEE Comput. Soc. Press, Los Alamitos, CA, 1995.

[13] U. Feige and P. Raghavan. Exact analysis of hot-potato routing (extended abstract). In *33rd Annual Symposium on Foundations of Computer Science*, pages 553–562, Pittsburgh, Pennsylvania, 24–27 October 1992. IEEE.

[14] M. Golumbic. *Algorithmic graph theory and perfect graphs.* Academic Press [Harcourt Brace Jovanovich, Publishers], New York-London-Toronto, Ont., 1980.

[15] B. Hajek. Bounds on evacuation time for deflection routing. *Distributed Computing*, 5:1–6, 1991.

[16] M. Kaufmann, H. Lauer, and H. Schroder. Fast deterministic hot-potato routing on processor arrays. In *5th International Symposium on Algorithms and Computation (ISAAC)*, pages 333–341. Springer-Verlag, 1994.

[17] C. Kaklamanis, D. Krizanc, and S. Rao. Hot-potato routing on processor arrays. In *5th Symp. on Parallel Algorithms and Architecture (SPAA)*, pages 273–282. ACM, 1993.

[18] T. Kloks. *Treewidth. Computations and approximations*, volume 842 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin, 1994.

[19] S. Konstantinidou and L. Snyder. The chaos router. *IEEE Transactions of Computers*, 43(12):1386–1397, 1994.

[20] Y. Mansour and B. Patt-Shamir. Greedy packet scheduling on shortest paths. *J. Algorithms*, 14(3):449–465, 1993.

[21] F. Meyer auf der Heide and M. Westermann. Hot-potato routing on multi-dimensional tori. In *Graph-theoretic concepts in computer science (Aachen, 1995)*, pages 209–221. Springer, Berlin, 1995.

[22] I. Newman and A. Schuster. Hot-potato algorithms for permutation routing. *IEEE Transactions on Parallel and Distributed Systems*, 6(11):1168–1176, 1995.

[23] A. Symvonis. A note on deflection routing on undirected graphs. Technical Report TR 493, University of Sydney, Department of Computer Science, november 1994.