Baked Potato Routing*

(Extended Abstract)

Shlomi Dolev[†] Evangelos Kranakis[‡] Danny Krizanc[‡]

Abstract

This paper identifies the equivalence of deadlock prevention in store-and-forward communication networks and simultaneous arrival of packets to a switch of bufferless high-speed networks. Scheduling of packet transmission schemes, which we call baked-potato schemes, are used to avoid simultaneous arrival of packets to a switch. We present scheduling schemes for any capacity of links and switches. The schemes are evaluated by the maximal length of time between two successive schedulings of a processor. For the case of single capacity link and switch, our scheme is proved optimal by presenting a matching lower bound. Our baked-potato scheme does not assume any prior knowledge on the source destination demands and can be used for sending control packets and broadcasting.

1 Introduction

The infrastructure of high-speed networks that transfer hundreds of Gigabits per second is becoming a reality. Traditional algorithms designed for the point-to-point store-and-forward classical model of distributed networks may not fit high-speed networks. In particular the use of store-and-forward almost nullifies the speed advantage obtained by fiber-optics cables. In order to utilize the speed capability of the fiber-optics, high-speed networks avoid buffering as much as possible. A fast and primitive, possibly optical (e.g. [14]), switch is used to forward (on the fly, if possible) incoming packets[1] to outgoing links. This new network structure requires careful re-examination of classical basic tasks in communication networks, e.g. [15, 17, 32, 1, 19].

Deadlock prevention and lossless networks: Deadlock prevention is one of the most important issues in distributed computing, e.g., [29, 34, 16, 28, 32]. Store-and-forward deadlock is a situation in which there is a cycle of processors such that each processor is full with packets and has to transmit its packets to the next processor in the cycle. In such deadlock situation no processor can transmit or receive packets, therefore no progress is possible. Algorithms to prevent deadlock without losing packets are of a major interest for traditional communications networks. The following citation is taken from [21] "The buffer deadlock problem is one of several major problems which have to be solved if a data communication network is designed on the basis of store-and-forward communication". One obvious way to "solve" deadlock is by elimination of packets — a processor transmits packets once and if the destination of the packet cannot accept the packet (say, it has no free buffer for the incoming packet) the packet is *lost*. In other words, the data link layer is not obliged to forward the packets to the next processor. Unfortunately, the above "solution" may raise severe delivery problems and may require retransmissions and duplications control at higher levels of the communication protocol. For these reasons the above solution is not allowed (or even mentioned) in the literature on deadlock prevention.

In the contexts of bufferless high-speed networks, similar to the transmission of packet to a processor with full buffer, packets can arrive simultaneously to a (full) switch. The common assumption in such a case is that some of the simultaneously arriving packets are *lost* [4, 7]. Both the deadlock problem and the proposed "solution" in the traditional point-to-point network resembles the simultaneous arrival of packets to a switch and elimination of such packets. Packet loss has

*Research supported in part by NSERC (Natural Sciences and Engineering Research Council of Canada) grant.

[†]Department of Mathematics and Computer Science, Ben-Gurion University, Beer-Sheva, 84105, Israel. Partialy supported by the Israeli ministry of science and arts grant #6756196. Email: **dolev@cs.bgu.ac.il**.

[‡]School of Computer Science, Carleton University, Ottawa, Ontario, K1S 5B6, Canada. Email: **{kranakis, krizanc}@scs.carleton.ca**.

[1] In ATM networks the term *cells* is used instead of packets.

drawbacks in high-speed networks as well. One would like to ensure that at least the most important control packets are not discarded by the network due to simultaneous arrival. Although some capacity of the links and switches can be dedicated for control packets it is possible that too many control packets will arrive simultaneously to a single switch. These observations raise the following question: is there a lossless "deadlock prevention" protocol for (bufferless) high-speed networks?

Hot-potatoes: One approach to deal with bufferless (synchronous) networks is the "hot-potatoes" or "deflection routing" approach, e.g. [5, 26, 2, 11, 22, 20, 13, 24, 12]. A packet that is routed to a busy outgoing link is deflected from its route and is sent through another non-busy outgoing link. Note that when the number of incoming and outgoing links is the same it is always possible to forward a packet from an incoming link to some outgoing link — no buffers are required. Most of the results obtained so far are for specific networks such as the mesh and torus and therefore more applicable to multiprocessors than to communication networks. In [26] it is proposed to apply deflection routing to communication network with regular structure (the Manhattan street network). In a recent work [30] it is suggested to use scheduled hot-potatoes approach for a restricted type of high-speed networks. In [30] it is assumed that packets do not carry routing information and switches forward packets only according to there arrival time; the route of a packet is solely determined by its departure time. For given source-destination pairs they present heuristic for scheduling (the departure of) packets. Unfortunately, the solution suggested in [30] is a "one-shot" solution that has to be re-computed for every given source-destination demands. Since the source-destination demands might be frequently changed packets containing new information concerning demands should be distributed. Thus, beyond the need for frequent re-computations of schedules, it is not clear how the control packets that convey the demand information would be routed. Scheduling packets in a bufferless network is also proposed for parallel computations, [9, 8], however either a single prtocessor is assumed as a source (destination) of all messages [9] or a one-shot solution is proposed for a given set of messages (with specific source and destination) [8].

Source routing and synchronized clocks: In the current technology of high-speed networks the routing of packets is determined by the source and not by the switches along the packet path, e.g. [3, 15, 19]. This fact has influence on congestion control, as described in [4]: "The most common and effective approach is to control traffic flow at entry points to the network (i.e., the access nodes). This approach is especially effective in ATM networks because of its connection-oriented transport". In a bufferless network one can assume that the (small) transmission delay over a link or a switch is a known constant. Thus, the location of a packet at a particular time is solely determined by the source of the packet (by the route the source chooses and the time the packet is delivered to the network). Obviously, in a totally asynchronous network when the destinations of (the control) packets are not priori restricted, sources cannot guarantee that no simultaneous arrival of (overwhelming number of) packets to a single switch will occur. Current and future technology will support almost precise synchronization of clocks (see, e.g. [31, 27]). When clocks are (almost perfectly) synchronized it is possible to guarantee that only one packet arrives to a switch at a time. In fact in the extreme it is possible to schedule the processors transmissions in a way that ensures that there is at most one packet in transit in the entire network.

Scheduling Persistent Tasks: Distributed control of communication network involves packet sending that needs to be scheduled infinitely many times. One approach suggested in [6] is to present the scheduling restrictions of persistent tasks by a conflict graph. The vertices of the conflict graph are the tasks to be scheduled and the edges of the conflict graph define pairs of tasks that cannot be scheduled concurrently. The suggested conflict graph is not descriptive enough for refined dependences as the dependences in packets delivery scheduling. For example, the conflict graph does not describe a restriction that does not allow two processors to be scheduled one after the other. Nevertheless, we share the objectives of [6] in achieving a short scheduling period (short response time and no drift).

Contributions of this paper: In this work we present the first *baked-potato* routing scheme for control packets in high-speed networks. In our solution every processor is scheduled every T_p time units. When a processor is scheduled it may transmit a packet to *any* destination (or even execute a broadcast) and the packet transmitted is not lost (due to simultaneous arrival of packets to a switch). The *capacity of a link* is the number of packets the link can simultaneously transmit. Similarly, the *capacity of a switch* is the number of packets the switch can receive at a time. A *time unit* is the time that it takes for a packet to traverse a single switch and a single link[2]. Our first algorithm is designed for the case in which the switch capacity and link capacity is 1. The

[2] In high-speed networks it is assumed that the traversal time of a packet in the network is a single time unit (see, e.g. [15, 17, 19]) which is equivalent to (approximately) n time units of our model.

algorithm periodically activates the processors; every processor is scheduled for transmission every at most $3n-2$ time units. We present a tight lower bound that proves the optimality of the algorithm. Our second result assumes link capacity 2 where only two packets in opposite directions may simultaneously traverse a link. The second algorithm is designed to work on a (spanning) tree communication graph. If the tree has l leaves the second algorithm schedules every processor in at most $2n-l+4$ time units. We present a lower bound of $3(n-l)/2$ for the length of the schedule period. The algorithm is then extended to any capacity of links and switches.

The rest of the paper is organized as follows. The definition of the problem appears in Section 2. Tight upper and lower bound for the single capacity switches and links are presented in section 3. Section 4 presents an upper bound and a lower bound for double capacity links. Extensions of our algorithms appear in Section 5. The concluding remarks section is Section 6.

2 Definition of the Problem

The high-speed communication network is described by an undirected graph $G=(V,E)$. The nodes, $V=\{1,\ldots,n\}$, represent the processors $\{P_1,P_2,\ldots,P_n\}$ of the network. The edges of the graph represent communication channels between the processors. A communication link between two processors P_i and P_j can be viewed as two directed links, one (P_i,P_j) from P_i to P_j and the other (P_j,P_i) from P_j to P_i, where (P_i,P_j) is an *outgoing link* of P_i and (P_j,P_i) is an *incoming link* of P_i. Each processor consists of two components, a *switching subsystem* and a *node control unit* that are connected by a link. The switching subsystem, which we shortly call the *switch*, is a fast and simple hardware device that switches arriving packets to the appropriate communication link or to its node control unit. Within the switching subsystem each communication link is assigned a unique label[3]. When a packet p arrives at a switching subsystem and the header of p contains at least one label, l, then the switching subsystem removes the first label, l, and the shortened packet is sent on the link with the label l. One natural extension of the above is to mark a packet as a *copy packet* to sign the switches along the packet path to deliver a copy of the packet to their node control unit as well as forwarding the packet according to the link label. The *node control unit* of each processor contains the processing hardware and software necessary to extract the information content of packets (delivered in the packets), do internal computation, and generate packets to be forwarded to other nodes via the processor's switching subsystem.

[3]The ATM network allows the assignment of a set of labels to a link to support *virtual paths* and *virtual channels* [23] and by that to reduce the number of labels concatenated to a packet. Our results hold for the case of multiple labels for a link — however for the sake of simplicity we assume a single link label at this point.

The processors have synchronized clocks and the delay of a packet in a switch and in a communication link is known. (In Section 5 we show how to relax these assumptions.) The *capacity of a link* is the maximal number of packets that can traverse the link simultaneously. A link with capacity 1 can transmit a single packet at a time. A link with capacity $2x$ can transmit x packets in each direction at a time. The switch capacity is the number of packets that can be forwarded by a switch simultaneously. The header of every arriving packet contains the label of the outgoing link the packet should be forwarded to. Some of the packets can arrive at a switch from the same incoming link and be forwarded to (possibly) different outgoing links. A switch with capacity C can forward any combination of C arriving packets to the outgoing links they have to be forwarded to.

A *time unit* is the time that it takes for a packet to traverse a single switch and a single link. Processors (especially those in central junctions of the backbone of the high speed networks) occasionally need to send control packets to other processors in the networks. The scheduling solution we are seeking is one that allows each processor to transmit a packet every T_p time units *to any destination* in a predefined route; where T_p is the *period* of the schedule. Formally, a *schedule* is a tuple of integers and a function $S=(T_p, Sched(P_1), Sched(P_2), \ldots Sched(P_n), R)$ such that T_p is the length in time units of the schedule period, $Sched(P_i)$ is the time in this period in which P_i is scheduled to transmit a packet, and R is a function from source destination pairs that returns a path along which the packet is routed. (Note that the result of R is fixed and does not depend on the current time). P_i can transmit a packet to any destination at time t iff $t \pmod{T_p} = Sched(P_i)$. A schedule is *admissible* if it does not violate the capacity restrictions of the links and the switches.

A *configuration* c_t of the system is a (possibly empty) vector of pairs of packets and (directed) links $(p_1,e_1),(p_2,e_2),\cdots,(p_k,e_k)$ such that p_i is transmitted over e_i immediately following t. A configuration c_{t+1} is reached from c_t if the only new packets introduced to the network are sent by the processors that are scheduled at time $t+1$ and every packet of c_i is advanced one hop (towards its destination) or arrived to its destination and does not appear in c_{t+1}.

Evaluating Schedules: One trivial example of an admissible schedule for every communication network is $S = (n^2, 0, n, 2n, \ldots, n^2 - n, R)$. R returns the shortest path from a source to a destination. S schedules P_1 at time 0 (and then at time $n^2, 2n^2, 3n^2$ etc.) then schedule P_2 at time n (and then at $n^2 + n, 2n^2 + n$ etc.). In general P_i is scheduled at time $j \cdot n^2 + i \cdot n$ for $j = 0, 1, 2 \cdots$. The above schedule ensures that at most one packet is in transit in the entire network — and thus avoids any possibility of packet collision at a switch or a link. The utilization of the communication capabilities of the network in the above example is small. Thus, a schedule is evaluated by the **length of its period**, i.e. a schedule with smaller period implies better utilization.

Another aspect in the evaluation of a schedule is the function R that the schedule uses. For example for a communication network in a ring structure with switches and links of capacity 1, $S' = (n - 1, 0, 0, \cdots, 0, R')$ is an admissible schedule. Where R' is the clockwise route in the ring from a source to a destination. This schedule is optimal in the length of its period since the destination of all the packets can be the same processor and the switch of the destination can handle one packet at a time. However, as opposed to a routing function that uses the shortest path, R' is a more expensive solution since it implies a greater usage of bandwidth. The *bandwidth usage* can be formally defined as the sum over all the processors of the longest path to a destination. For instance for S' this sum is $n \cdot (n - 1)$. A schedule with smaller *bandwidth usage* is prefered. Therefore, the algorithms and lower bounds presented in this paper are for the case of a **shortest path routing**.

3 Single Capacity Switches and Links

In this section we present tight upper and lower bounds for a communication network with a single link and switch capacity. The obtained results are applicable to any communication network with such capacities and for scheduler with optimal bandwidth usage, namely R returns the shortest path to the destination.

The following pictorial description of the problem helped us in seeking for a schedule. Snow covers city roads in a way that allow only one car to pass any road and junction at a time (if two cars are approaching in opposite directions they cannot pass each other). To eliminate sliding cars drive from their source to the destination in a constant speed without stopping. The citizens are restricted to choose a shortest path to their (not priori known) destination. A solution that schedules all the citizens for departure time and minimize the waiting time is required.

3.1 Upper Bound

The scheduling algorithm may be described as a traversal of a token in the communication graph. Whenever the token reaches a node for the first time in the tour the node is scheduled. An invariant ensuring that the packets are always directed from the token to their destination is maintained. To maintain this invariant, the token is delayed letting packets be advanced towards their destination before a new neighboring processor is scheduled.

Step 1: Find a shortest tour, $\mathcal{T}$, in the graph.

Step 2: Perform a walk on the tour obtained in step 1. Maintain a *marking counter* that is initialized to zero and incremented in every step in the tour. The value of the marking counter is used to mark a visited node v, by $Sched(v)$, when v is visited for the first time (in the walk). Following a marking of a newly visited node the value of the marking counter is incremented by two. Otherwise, when the visited node is already marked the value of the marking counter is incremented by one. The walk ends when all the nodes were visited and the first node is reached. The value of the marking counter at the end of the walk is the schedule period denoted by T_p.

Figure 1: Scheduling in Single Capacity Links and Switches.

Note that the length of the shortest tour, $|\mathcal{T}|$, found in step 1 of Figure 1 is less than or equal to $2(n - 1)$ which is the length of an Euler tour on a spanning tree of the graph. Note also that a graph with Hamiltonian cycle has a tour of length n which is the shortest possible tour.

Definition 3.1 *Let dist(Q,R) be the number of links in a shortest path between Q and R.*

Lemma 3.1 *If Q is marked first by the algorithm then sched(R)−sched(Q) $>$ dist(Q,R) and sched(Q)−sched(R)* $\pmod{T_p}$ $>$ *dist(Q,R).*

Proof: After Q is marked and before R is marked the marking counter is incremented by two (just after Q

is marked) and then it is incremented by at least one every time a node is visited in the way from Q to R. Since the shortest path from Q to R includes $dist(Q,R)$ links there must be at least $dist(Q,R)-1$ intermediate nodes. Thus, the marking counter is incremented by at least $dist(Q,R)+1$ after Q is marked and before R is marked. This proves that $sched(R) - sched(Q) \geq 2 + dist(Q,R) - 1$. To prove that $sched(Q) - sched(R) \pmod{T_p} > dist(Q,R)$ it is enough to notice that we can start the marking walk from R and obtain an equivalent schedule. ■

Theorem 3.2 *The single link capacity algorithm produces a valid schedule.*

Proof: Assume towards contradiction that packets of two processors Q and R arrive at the same time to a node v. (Note that when the routing is a shortest path routing simultaneous arrival to a link can be reduced to arrival to a node by adding an intermediate node in the middle of the link). Q (and R) uses the shortest path to v. Let $dist(Q,R) = d$, $dist(Q,v) = x$ and $dist(R,v) = y$. Note that d, x and y are shortest paths thus $x + d \geq y$ and $y + d \geq x$.

If packets from Q and R arrive simultaneously to v at time t then Q (R) was scheduled at time $t - x$ ($t - y$, respectively). By Lemma 3.1 $t - y - (t - x) > d$ and $t - x - (t - y) \pmod{T_p} > d$. Thus, $x - y > d$ which contradicts $y + d \geq x$ and $y - x \pmod{T_p} > d$ which contradicts $x + d \geq y$. ■

Lemma 3.3 *The T_p produced by the single link capacity algorithm is $|\mathcal{T}| + n \leq 3n - 2$.*

Proof: Every step in the tour increments the marking counter by one or two. The number of steps in the tour is $|\mathcal{T}|$. The marking counter is incremented by two only due to a newly visited node. Thus, $T_p = |\mathcal{T}| + n$. The proof is completed since an Euler tour of a spanning tree of a graph yields $|\mathcal{T}| = 2n - 2$. ■

3.2 Lower Bound

The lower bound is for shortest path routing.

Lemma 3.4 *The time difference between any two packet send events is greater than the distance of the two packet sources.*

Proof: Assume towards contradiction that the time difference between two packet send events is less than the distance of the two packet sources. In such a case the packets can arrive at an intermediate node or link along a shortest path between the two sources at the same time. ■

Theorem 3.5 *The period produced by any scheduling algorithm is of length of at least $|\mathcal{T}| + n$.*

Proof: Let $p_1, p_2, \cdots, p_n$ be the order of the processors in the schedule. By Lemma 3.4 T_p is at least $dist(p_1,p_2) + dist(p_2,p_3) + \cdots + dist(p_n,p_1) + n$. Since we start with p_1 and finish with p_1 $dist(p_1,p_2) + dist(p_2,p_3) + \cdots + dist(p_n,p_1)$ is the length of the tour that uses the shortest paths between every p_i and p_j in $dist(p_i,p_j)$. The proof is completed since $\mathcal{T}$ is the shortest possible tour. ■

Note that for a tree $|\mathcal{T}| = 2n-2$ and thus the length of the period of any schedule for tree communication graph is at least $3n-2$.

4 Double Capacity Links

In this section we consider the case in which each link can transfer two packets at a time, a packet in each direction. This situation occurs in full-duplex links where communication can be transfered in both directions. The restriction on a single packet transmission in each direction is also applied to the link connecting the switching subsystem with the node control unit. This in turn implies that no two copy packets can reach a switch simultaneously since they are both routed to the node control unit. Since we are interested in broadcasting information (using copy packets) as well as sending individual packets we restrict the number of packets reaching a switch at a time to be one. The advantage in using double capacity links can be easily demonstrated by a two processors system. In the double capacity case both processor can transmit simultaneously every time unit, such a scheduling is not possible in the single capacity case.

We present a scheduling algorithm for a system with a tree communication network. The schedule is for shortest (simple) path packet routing on the tree. Note that the scheduling algorithm can be applied to a spanning tree of the communication graph. However, in such a case the routing is along the spanning tree and not necessarily along the shortest possible path.

4.1 Upper Bound

The algorithm uses two tokens that move in an Euler tour of the tree chasing each other. For ease of discussion we add the locations of the tokens to the configuration. A configuration c_t is a list of pairs of packets and links and two integers Tkn_1 and Tkn_2 that are the indices of the Euler tour of the tokens location immediately before time t. A token *moves* at time t if its location in c_t and c_{t+1} is not identical. A token waits a single time unit if its location in c_t and c_{t+1} is identical. The description of the algorithm appears in Figure 2; it uses the following definition.

Definition 4.1 *In a certain configuration the (single) path connecting the two tokens is the* token tree path.

Each edge on the token tree path of a specific configuration can be either *active* or *non-active*. As we show later packets that are transferred on the tokens tree path use only active edges. Every active (non-active) edge remains active for a single time unit and then becomes non-active (active, respectively) for a single time unit. Thus, the status of an edge on the token tree path alternates every time unit. The status of an edge that joins the token tree path is defined according to the status of a neighboring edge on the token tree path. Packets that are transfered on an edge that does not belong to the tokens tree path are always directed from a node on the path towards a leaf. The token tree path is empty when the two tokens reach the same node on the tree. At such an instance all packets are directed from the tokens location towards the leaves. In the sequel we prove that one time unit after both tokens are simultaneously at a certain node it holds that the token tree path consist of a single active edge.

In Figure 3 we demonstrate the execution of the algorithm that results with the schedule $S = (8, 0, 3, 6, 0, 1, 4, R)$ where $T_p = 8$, $Sched(P_1) = 0$, $Sched(P_2) = 3$, etc. and R returns the (shortest) path on the tree. P_4 is chosen as the home location for the second token since it is reached by the first token $3 + 3 + 3 = 9 \geq 6 + (6 - 1) - 3 = 8$ time units following its arrival to P_1 and it is at distance three from P_1. The location of the tokens is marked both on the tree and on the Euler tour. The arrows represent packets — since the source destination demands are not known in advance every possible path of a packet is considered. If a token fires at time t then its location is marked by $t = i$. The schedule repeats itself every 8 time units i.e. at time 8 both P_1 and P_4 may send packets.

Step 1: Choose a leaf to be the root and direct the tree from this root. Mark the Euler tour of the tree marking up edges and down edges differently. Note that the tour contains $n - 1$ down edges and $n - 1$ up edges. Define the weight of every down edge that is leading to a *non-leaf* node to be three and the weight of all the rest of the edges to be one.

Step 2: Mark the root as the first token starting point. Perform a walk through the Euler tour starting at the root. Use a *length accumulator* during the tour. In the beginning of the tour the value of the length accumulator is set to zero. Every traversal of an edge increments the length accumulator by the length of the traversed edge. Stop when the accumulator value is equal to or greater than $n + (n - 1) - l$, where l is the number of leaves in the tree. If the node reached is in odd distance from the root on the tree then mark the position reached on the Euler tour as the second token starting point. Otherwise, mark the next position in the Euler tour as the starting point of the second token.

Step 3: The tokens move along the Euler tour starting from their starting points in the Euler tour. We use a time counter to mark the nodes. The time counter is initiated to zero and is incremented by one each time unit. We say that the *token fires* when the node holding the token is marked by the time counter value. The first token fires at time zero i.e. marks its starting points by zero. The second token fires at time zero if its starting point is reached via a down edge of the Euler tour. In every time unit the location of the tokens is advanced according to the Euler tour. The only exception is when the token reaches its location via down edge and the node it is located in, is a non-leaf node. In such a case, the token waits two time units and then fires. In addition, the token fires every time it reaches a leaf node. The second token stops when it reaches the starting point position of the first token. The marking procedure is completed one time unit before the first token leaves the location in the Euler tour of the starting point of the second token. The value of the time counter at the completion of the marking procedure is assigned to T_p.

Figure 2: Scheduling with Double Capacity Links.

Lemma 4.1 *The following hold in every configuration:*

1. *A token has a neighboring active edge immediately after any of its moves unless the move places it with the other token.*
2. *If the two tokens are simultaneously at a node then none of them fires and one time later the token tree path consists of a single edge. This single edge is an active edge.*
3. *A token fires only when it is neighboring to an active edge.*
4. *There is no packet in a non-active edge.*
5. *There are at most two packets on an active edge. If two packets are transferred in a particular time through a particular edge then they are in opposite directions.*
6. *The direction of the packets that are not on the token tree path does not lead to a node on the tokens tree path. In other words, the packets on edges that do not belong to the token tree path are directed towards a leaf and not towards a node in the token tree path.*

The lemma is proved by induction. We assume the assertions of the lemma hold up to the configuration c_t and prove it for c_{t+1}. The details of the proof are omitted from this extended abstract.

The next theorem is implied by Lemma 4.1.

Theorem 4.2 *The algorithm for the double capacity links produces a valid schedule.*

Lemma 4.3 *The T_p produced by the double link capacity algorithm is less than or equal to $2n - l + 4$.*

Proof: In Step 2 of the algorithm the length accumulator can be greater by 2 than $n + (n-1) - l$ when it first exceed $n + (n-1) - l$. In order to guarantee odd tokens tree path the location of the second token starting points can be one node further. This yields $n + (n-1) - l + 2 + 3$. ■

As a corollary from Lemma 4.3, a star communication network can be scheduled in at most $n + 5$ time units. A full binary tree can be scheduled in at most $3n/2+3$ time units. For the case of a general graph (and routing on a spanning tree) the period of the schedule depends on the number of leaves in the spanning tree chosen. Finding the maximal leaf spanning tree is NP-complete. In [25] a polynomial algorithm is presented to construct a spanning tree whose number of leaves is at least $1/3$ the maximal. This algorithm can be used in choosing a spanning tree that yields short period of schedule.

4.2 Lower Bound

The lower bound is for a tree communication graph. The proofs are omitted from this extended abstract.

Claim 4.4 *In every time unit at most two processors can send packets simultaneously.*

Claim 4.5 *If two processors, Q and R, send packets simultaneously then any non leaf processor $P \notin \{Q, R\}$ cannot send a packet for at least two time units.*

Claim 4.6 *Two non-leaf processors, Q and R, cannot send simultaneously a packet one time unit after a non leaf processor $P \notin \{Q, R\}$ sent a packet.*

Claim 4.7 *If two non-leaf processors, Q and R, send packets one after the other in two consecutive time units then any non leaf processor, P, cannot send a packet for at least one time unit.*

Theorem 4.8 *The period produced by any scheduling algorithm for a tree graph with l leaves is at least $3(n-l)/2$.*

5 Extensions

In this section we describe a few natural extensions to our results. We first describe how our algorithm can be applied to a network with $2x$ links capacity. Then we show how our algorithms can broadcast packets, and at last discuss the effect of imprecise clocks.

For a $2x$ capacity of a link we can apply x versions of the double capacity algorithm starting their (first) tokens equally apart. The resulting period of the schedule will be roughly shorter than the period of the double capacity network by a factor of x.

To perform broadcast we allow multiple labels for links. A straightforward solution is one in which the labels of the links encode n shortest path trees, a tree $\mathcal{T}_Q$ for each source Q. Each link on a path on $\mathcal{T}_Q$ is labeled by l_Q. A copy packet sent with the label l_Q is sent through all the links labeled l_Q and to the node control unit.

When clocks are not synchronized our scheduling algorithms cannot avoid packet collision and loss. In certain cases when the clocks are partially synchronized there is still a possibility to schedule packet delivery while avoiding collisions. For example, when the maximal difference in the value of the clocks is bounded by y time units, our single capacity algorithm can delay the token for extra y time units after marking a node. Roughly speaking, this slower movement of the token ensures a "safety distance" and prevents collisions.

6 Concluding Remarks

In this paper we introduce a new approach for the design of control packet delivery in high-speed networks. Algorithms for scheduling packet delivery are suggested. The algorithms ensure that packets *are not lost*. The algorithm for the single capacity links is for general graphs and uses shortest path routing. The algorithms for the double capacity and $2x$ capacity links is for a communication graph in the form of a tree.

The framework we suggest is a base for future research for studying upper and lower bounds for special graphs and routing schemes (not necessarily shortest path). Such results can support the design of future high-speed networks. For example, scheduling of message transmissions (or similarly, phone calls) can overcome situations in which an external event (e.g. earthquick) causes simultaneous transmission and overloading a network component (e.g. a switch). When a processor identifies a communication problem the processor restricts its message transmission according to the schedule.

Next we demonstrate the variety of results we obtained. One example is a ring of odd number of processors. It is easy to verify that in such a ring, if the packets are routed according to the shortest path then no processor can be scheduled with or immediately after another processor. Thus, a lower bound of $2n$ time units for the period of the schedule is obtained. On the other hand, a non shortest path routing for the same ring implies a period of n — all the processors transmit simultaneously in a single direction.

Another aspect is the possibility to relax the requirements. Our definition of period is somewhat strict, requiring equal response time for each processor and no drift. A solution in which the average time between two activation is minimized instead of minimizing the maximal time between activation may be of interest. For instance, the approximately $3n$ upper bound for a single capacity chain can be improved to average of $2n$ time units between activations when allowing the maximal time between activation to be $4n$. Roughly speaking, when the token reaches the end of the chain it starts scheduling the processors in backwards order. Similarly, for the double capacity links the upper bound on the chain becomes $3n/2$. Note that the lower bound still holds and hence the algorithm is optimal for such settings. In addition in case the maximal time between activation is not restricted, the single line capacity algorithm can be easily modify to support long messages delivery: if k time units are required to transmit a particular message (instead of a single time unit) then the token is delayed for additinal $k-1$ time units before it is forwarded to the next node.

At last, we would like to mention that there are more efficient solutions for the destination non-oblivious case (e.g. [8]). Those solutions, require somewhat more complicated processing at the sending node. Moreover, a destination non-oblivious solution does not support simultaneous broadcast. In simultaneous broadcast, a processor can deduce, from the packet arrival time and the knowledge of the communication graph, the arrival of the packet to every other processor. A simple example of non-oblivious schedule is for the case of a chain where processors simultaneously send their packets to the right and n steps later to the left. Thus, this non-oblivious schedule results in a period of $2n$.

Acknowledgment: We thank Amotz Bar-Noy for useful comments on a written draft.

References

[1] H. Abu-Amara, B. Coan, S. Dolev, A. Kanevsky, and J. L. Welch, "A Fault-Tolerant Layered Approach to Fiber-Optic Networks," *High-Speed Networking and Multimedia Computing*, Proc. SPIE 2188 pp. 380-390, 1994.

[2] A.S. Acampora and S.I.A. Shah. "Multihop lightwave networks: a comparison of store-and-forward and hot-potato routing", *IEEE INFOCOM*, pp 10-19, 1991.

[3] The ATM Forum, *ATM, User-Network Interface Specification, version 3.0.* Prentice Hall, 1993.

[4] J. J. Bae and T. Suda, "Survey of Traffic Control Schemes and Protocols in ATM Networks," *Proceedings of the IEEE*, vol. 79, pp. 170-189, Feb. 1991.

[5] P. Baran. "On distributed communications networks", *IEEE Transactions on Communications"*, vol. 12, pp 1-9, 1964.

[6] A. Bar-Noy, A. Mayer, B. Schieber and M. Sudan, "Guaranteeing Fair Service to Persistent Dependent Tasks", SODA 1995, 1995.

[7] D. Bertsekas and R. Gallager, Data Networks, *Prentice-Hall*, pp. 138-139, 1992.

[8] S.N. Bhatt, G. Bilardi, G. Pucci, A.G. Ranade, A.L. Rosenberg, and E.J. Schwabe "On bufferless routing of variable-length messages in leveled networks", *1st European Symp. on Algorithms*, Bad Honnef, Germany (T. Lengauer, ed.) LNCS 726, pp. 49-60, 1993.

[9] S.N. Bhatt, G. Pucci, A.G. Ranade, A.L. Rosenberg "Scattering and gathering messages in networks of processors", *IEEE Trans. Comp.* 42, 938-949, 1993.

[10] B. J. Brachman and S. T. Chanson, "A hierarchical solution for application level store-and-forward deadlock prevention," *SIGCOM89, ACM*, pp. 25-32, 1989.

[11] J.T. Brassil and R.L. Cruz. "Bounds on maximum delay in networks with deflection routing", *IEEE Transaction on Parallel and Distributed Systems*, July, 1995.

[12] A. Ben-Dor, S. Halevi and A. Schuster. "Potential Function Analysis of Greedy Hot-Potato Routing", *Proc. of the 13th Annual ACM Symposium on Principles of Distributed Computing*, pp. 225-234, 1994.

[13] A. Bar-Noy, P. Raghavan, B. Schieber, and H. Tamaki. "Fast deflection routing for packets and worms", *Proc. 12th ACM Symposium on Principles of Distributed Computing*, 1993.

[14] C. Chen and S. Banerjee. "Optical Switch Configuration and Lightpath Assignment in Wavelength Routing Multihop Lightwave Networks," *IEEE INFOCOM*, pp. 1300-1307, 1995.

[15] I. Cidon and I. Gopal. "PARIS: An approach to private integrated networks," *Journal of Analog and Digital Cabled Systems* 1(2), pp. 77-86, 1988.

[16] I. Cidon, "An Efficient Knot Detection Algorithm", *IEEE Transactions on Software Engineering,* vol. 15, no.5, pp. 644-649, 1989.

[17] I. Cidon, I. Gopal and S. Kutten, "New models and algorithms for future networks," *Proc. of the 7th ACM Symposium on Principle of Distributed Computing*, pp. 75-89, 1989.

[18] S. Dolev, E. Kranakis and D. Krizanc. "Baked Potatoes: Deadlock Prevention Via Scheduling", Technical Report #95-07 Department of Mathematics and Computer Science Ben-Gurion University.

[19] S. Dolev, E. Kranakis, D. Krizanc and D. Peleg. "Bubbles: Adaptive Routing Scheme for High-Speed Dynamic Networks", *Proc. of the 27th Annual ACM Symposium on the Theory of Computing*, pp. 528-536, 1995.

[20] U. Feige and P. Raghavan. "Exact analysis of hot-potato routing", *Proc. of the IEEE Symposium on Foundations of Computer Science*, pp. 553-562, 1992.

[21] K. D. Gunther, "Prevention of deadlocks in packet-switched data transport systems", *IEEE Transactions on Communication*, Special Issue on Control in Computer Networks, pp. 512-524, 1981.

[22] B. Hajek. "Bounds on evacuation time for deflection routing", *Distributed Computing*, vol. 5, pp. 1-6, 1991.

[23] R. Handler and M.N. Hubler, "Integrated Broadband Networks: an introduction to ATM-based networks", *Addison-Wesley*, 1991.

[24] C. Kaklamanis, D. Krizanc and S. Rao. "Hot-potato routing on processor arrays", *Symposium of Parallel Algorithms and Architecture*, pp. 273-282, 1993.

[25] H.-I. Lu and R. Ravi, "The Power of Local Optimization: Approximation Algorithms for Maximum-Leaf Spanning Tree", *Proceedings of 1992 Allerton Conference.*

[26] N.F. Maxemchuk, "Comparison of deflection and store and forward techniques in the Manhattan street and shuffle exchange networks", *Proc. IEEE INFOCOM*, pp. 800-809, 1989.

[27] D. L. Mills, "Improved Algorithms for Synchronizing Computer Network Clocks", *IEEE/ACM Transactions on Networking*, vol. 3, no. 3, 1995.

[28] J. Misra and K. M. Chandy, "Distributed Graph Algorithms: Knot Detection, *ACM Transactions on Programming Languages and Systems,* vol. 4, no. 4, pp. 678-686, 1982.

[29] P. M. Merlin and P. J. Schweitzer, "Deadlock avoidance in store-and-forward networks — I: Store and forward deadlock," *IEEE Transactions on Communication*, vol. COM-28, pp. 345-352, 1980.

[30] S. Naor, Orda and R. Rom, "Scheduled Hot-Potato Routing", *IEEE INFOCOM*, pp. 579-586, 1995.

[31] Y. Ofek, "Generating a fault tolerant global clock using high-speed control signals or MetaNet architecture", *IEEE Transactions on Communication*, pp. 2179-2188, 1994.

[32] Y. Ofek and M. Yung, "Principles for high-speed network control: Loss-less and deadlock-freeness, self-routing and single buffer per link," *Proc. 9th ACM Symp. on Principles of Distributed Computing*, pp. 161-175, 1990.

[33] T. Szymanski, "An analysis of hot potato routing in a fiber optic packet switched hypercube", *Proc. IEEE INFOCOM*, pp. 918-926, 1990.

[34] S. Toueg and J. D. Ullman, "Deadlock-free packet switching networks," *SIAM Journal on Computing*, vol. 10, pp. 594-611, 1981.

[35] Z. Zhang and A.S. Acampora, "Performance analysis of multihop lightwave networks with hot potato routing and distance age priorities", *Proceedings IEEE INFOCOM*, pp. 1012-1021, 1991.

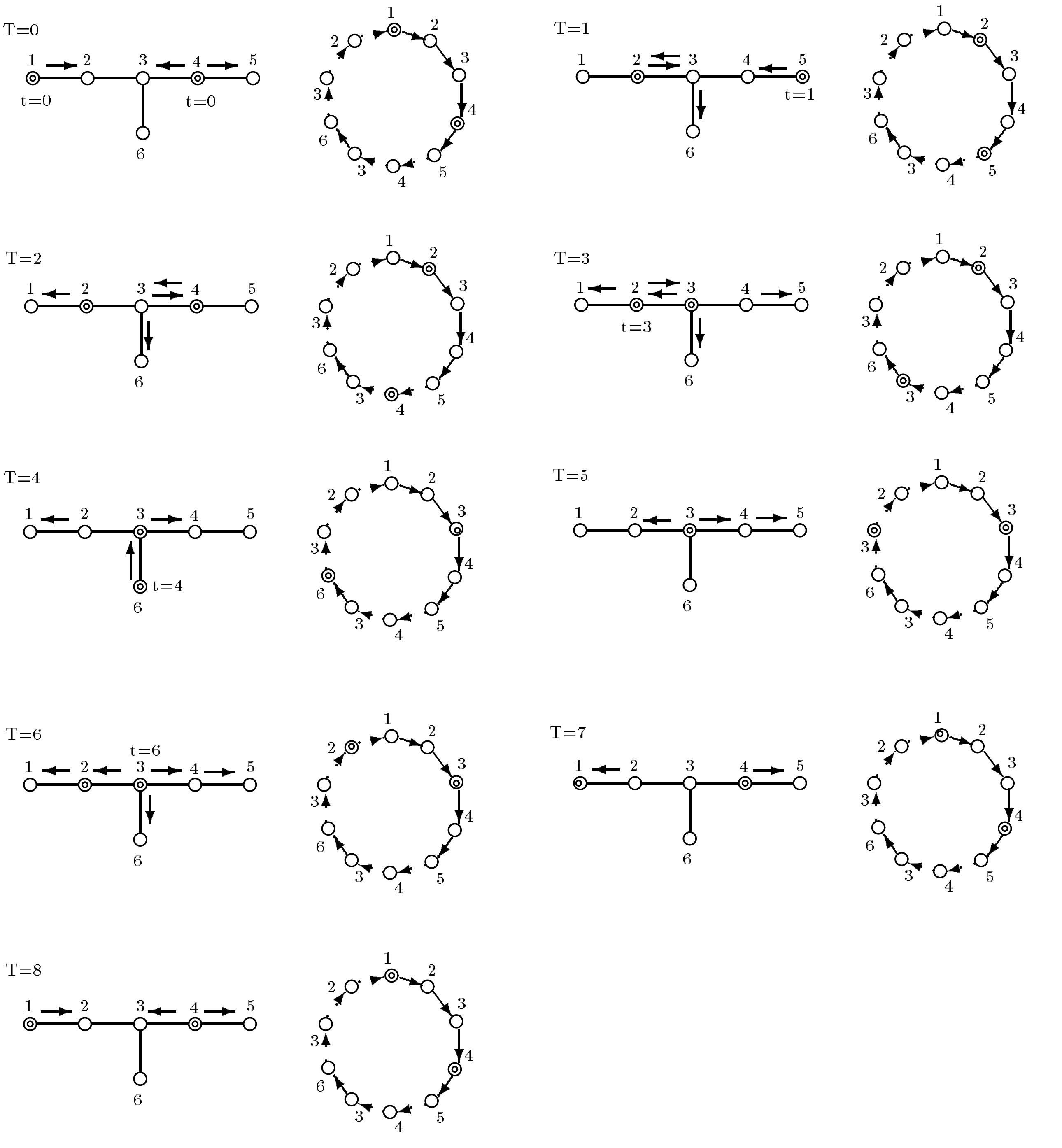

Figure 3: Double Capacity Links