Journal of Graph Algorithms and Applications
http://www.cs.brown.edu/publications/jgaa/
vol. 2, no. 4, pp. 1–20 (1998)

Scheduled Hot-Potato Routing

Joseph (Seffi) Naor
Department of Computer Science
Technion, Haifa 32000, Israel
http://www.cs.technion.ac.il/
naor@cs.technion.ac.il

Ariel Orda *Raphael Rom*
Department of Electrical Engineering
Technion, Haifa 32000, Israel
http://www.ee.technion.ac.il/
ariel@ee.technion.ac.il rom@ee.technion.ac.il

Abstract

This paper is concerned with fast, hot-potato routing, performed according to a predetermined schedule. At each time period each node selects an outgoing link, through which an incoming packet is sent. No buffers are used. We investigate first the problem of how to route a network-wide demand of packets, given the predetermined schedule. We show that certain versions of the problem have efficient solutions, while other versions are intractable. We then consider the problem of finding an optimal schedule given a network-wide demand of packets. We indicate that the problem is tractable for either a single source or single destination. However, for the multi-source multi-destination case we show that it is an NP-complete problem. We present an efficient heuristic for directed tree-networks, and adapt it to general topologies through a recursive scheme, for which an efficient performance bound is shown.

Communicated by S. Khuller: submitted November 1996; revised November 1997.

Raphael Rom is also with Sun Microsystems, Mountain View, CA, 94043.
A preliminary version of this paper appeared in the Proceedings of IEEE INFOCOM 1995, Boston, MA, USA, pp. 579-586. This work was supported by the Broadband Telecommunications R&D Consortium administered by the chief scientist of the Israeli Ministry of Industry and Trade.

1 Introduction

Due to the development of high-speed networks, there has recently been growing interest in efficient routing techniques that would allow a fast decision at the node regarding the edge through which an incoming packet should be routed. An efficient routing technique for a high-speed network is one that demands little "thinking" from the node. Ideally, a node would take an immediate routing decision (e.g., [4]). In addition, an efficient scheme for high-speed routing would avoid the use of buffering at intermediate nodes: ideally, a packet would flow throughout the network without queuing at nodes (e.g., [27]). Indeed, queuing means more delay and also a more complicated strategy from the node's standpoint.

A possible solution that addresses the above issues is that of *scheduled hot-potato routing*. Here, "scheduled" means that each node routes incoming packets to outgoing links according to a predetermined schedule, i.e., the identity of the outgoing link is a function solely of time. This means that the action taken by the node is immediate, and in particular it does not even need to look at the packet's header in order to perform the routing. Moreover, the routing decision becomes even more simple if we just let one outgoing link be "active" at every time instant. By "hot-potato" routing we mean that packets are not queued enroute; rather, an incoming packet is immediately transferred to a neighboring node, or, if this is impossible, it is discarded. Thus, with scheduled hot-potato routing we let packets move smoothly and quickly among nodes, through high-speed links, without encountering intra-nodal queuing, nor switching delays. Nonetheless, such a strategy demands careful planning, both of the predetermined schedule and of its actual use, once it has been set.

Hot-potato (also called deflection) routing has been given much attention in recent years, e.g., [7]–[20], mainly in the area of multiprocessing. Typically, however, in the above studies, hot-potato routing is applied to regular topologies and considers the identity (label) of the packet for making its routing decision. We note that deflection routing solutions are frequently plagued by deadlock and flow control problems [21]. Scheduled routing has also been the focus of several studies, e.g. [18]–[24], but these works considered the use of store-and-forward buffering.

In this work we consider the combination of both principles, namely scheduling and hot-potato routing, in order to achieve an efficient method for routing in high-speed networks. Modern high-speed networks are oriented towards the support of long duration sessions. In such networks, a session is set up in advance, determining both the routes and the rate in which data will subsequently flow [23]. Because the lifetime of a session is relatively long, there is typically enough time to gather the data that is necessary to make the routing decision. Similar ideas to those presented in this paper have been applied to practical pilot networks, e.g., Isochronets [27], in which network bandwidth is time-divided among routing trees in order to allocate "green-band" paths for traffic destined to a common root. However, until now, these ideas were tested only at the functional level, while the present work is the first to provide formal analysis.

Two typical problems related to routing through pre-established connections are the *design* problem and the *usage* problem. The first is concerned with the efficient design of the long-term routing plan, once some data on the source-destination demands is collected. The second is concerned with the best use of an existing routing plan, for traffic patterns that were not necessarily accounted for at the time that the routing plan was devised, e.g., low-priority traffic classes. We note that the usage problem follows the philosophy of the best-effort traffic class in ATM networks [23].

The present work is an attempt at providing an analytical framework for this novel method of routing. We establish the problems that should be addressed, provide optimal solutions to some while proving the intractability of others, for which efficient heuristics are described and analyzed. Our solution applies to general topologies and provides bounds on the delivery delay of packets. Naturally, with such bounds in effect, neither deadlock nor flow control is an issue. These bounds on the delay also make our scheme particularly adequate for real-time applications.

Specifically, we address the following problems. First, we consider a network for which a schedule has been predetermined, and investigate the usage problem, i.e., how to route a given demand of packets (for various source-destination pairs), through a pre-established schedule, in the most efficient way. We then consider the design problem, i.e., planning the best schedule for an expected demand of packets. We give an optimal solution for the first problem. We show that the second problem is intractable, even for very simple topologies. We indicate its tractability in some special cases and present efficient heuristics for the general case.

2 Model

The network is a directed graph $G(V, E)$, where V is the set of nodes and $E \subseteq V \times V$ is the set of edges, or links. For a node $i \in V$, let IN_i be the set of neighbors k such that $(k, i) \in E$, and OUT_i be the set of neighbors k such that $(i, k) \in E$.

We consider a discrete domain of time and a synchronous mode of operation. It is assumed that transmission delay on all links is of unit time, and refer to a *packet* as the amount of data that can be transmitted in a unit of time. Thus, at most one packet can be carried on a link at any time. A unit of time shall be referred to as a *slot*. Furthermore, it is assumed that nodes do not have buffers, and thus, an incoming packet has to be switched immediately to an outgoing link.

Considering a time period of P, we define the *nodal schedule* of a node i, S^i, to be a sequence $\{k_0^i, k_1^i, \cdots, k_\tau^i, \cdots, k_{P-1}^i\}$ such that, for $0 \leq \tau \leq P-1$,

$k^i_\tau \in OUT_i$, i.e., the nodal schedule of i assigns to each time-slot within $[0, P-1]$ a single neighbor of node i (routing is discussed next). Note that P is a given quantity independent of any other network parameter (notably of $|V|$). Typically, one can assume that P is at most linear in $|V|$. Given a nodal schedule for each node, we define the *network schedule* S to be the set comprising of all nodal schedules, i.e., $S = \{S^i | i \in V\}$.

We now discuss the *scheduled hot-potato routing* for a given network schedule S. Consider a node $i \in V$ and a time-slot t[1]. If exactly one packet arrives at i at the beginning of that time-slot, then it is sent (immediately) through link (i, k), where $k = k^i_\tau \in S^i$, and $\tau = t \bmod P$. If more than one packet arrives at the same time, then an arbitrarily chosen (single) packet is routed according to the schedule, and the rest of the packets are *eliminated*.

The network should support the flow of traffic between *connections* (or *sessions*), each identified by a pair of source and destination nodes. A *demand* D of size K on the network is a sequence of (not necessarily different) K source-destination pairs $D = \{(s_1, d_1), (s_2, d_2), \cdots (s_K, d_K)\}$; each pair belongs to a certain connection and designates one packet transmission of that connection. At times, we denote by p_j the packet associated with the j-th pair of D. In other words, $D = \{p_1, p_2, \cdots p_K\}$, where p_j is a packet to be routed from node s_j to node d_j. If, for all j, $s_j \equiv s$ then we say that D is a *single source* demand; similarly, if for all j $d_j \equiv d$ then we say that D is a *single destination* demand; if both hold, we say that D is a *single-source/single-destination* demand. For ease of presentation we shall assume that each node may be a source of at most one packet, i.e., $s_i \neq s_j$ for $i \neq j$ (this implies $K \leq |V|$). Our results are easily adapted to handle the more general case, by splitting a source s with K_s packets into a sequence of K_s independent nodes each connected to s. Denote by $M_{G,D}$ the diameter of graph G with respect to D, i.e., the maximal minimum-hop distance among all source-destination pairs of D.

Given a network demand D of size K, a *departure plan* Δ is a sequence of (integer) times $\Delta = \{\delta_1, \delta_2, \cdots \delta_K\}$ such that δ_j is the departure time of packet p_j i.e., p_j leaves the source node s_j (for the first time) at the beginning of time-slot δ_j.

A packet is routed between nodes until it arrives at its destination or it is eliminated. For given S and D, we say that a departure plan Δ is *proper* if all packets arrive at their destinations, i.e., no eliminations nor livelocks occur. For S, D and a proper Δ, the *arrival time* α_j of a packet p_j is the time at which it reaches the destination; if the packet does not reach its destination, then we set $\alpha_j = \infty$. For given S and D, the *termination time* of a proper departure plan Δ is the highest value of arrival times (possibly infinity).

We note that, even if p_j is the only packet in the network, for a given network schedule S, there may still be no departure time for p_j that would bring it to its destination. Namely, for every departure time the packet will loop in the

[1] All times are global relative to some starting time denoted by $t = 0$.

network forever. We say that S is *feasible* for packet p_j if there is some finite time t, such that if p_j departs at t, then it arrives at its destination in finite time; t is said to be a feasible departure time for p_j, given S. We say that a schedule S is feasible for demand D if it is feasible for each packet of D.

¿From the previous description it is not clear how packets are ever removed from the network. A useful way to view this is to assume that each node has a link emanating from it that terminates with the end users that are attached to that node. In other words, for a given node i, the set OUT_i contains a link to the attached end users. Thus, if a packet arrives at node i at time τ, and if schedule S^i indicates that k^i_τ is the edge that points to the end user, then we say that the packet is delivered to its destination. A variation on this approach is the one described in [4], where, by default, each packet, while being switched according to the nodal schedule, is also delivered to the end user attached to every node along the path, and is discarded by those nodes for which it is not intended (as is done in most LANs).

3 Use of a Given Schedule

In this section we consider the situation in which the network and its schedule are given and we look for ways to route packets through the network under various timing circumstances.

The first question is whether a proper departure plan of finite termination time exists at all. The following lemma gives a necessary and sufficient condition.

Lemma 1 *Given a network G, a schedule S and a demand D, there is a proper departure plan Δ with finite termination time if and only if S is feasible for D.*

Proof: Suppose that S is not feasible for D. Then, there is some packet p_j that will not arrive at its destination, and thus no Δ can have a finite termination time.

Conversely, suppose that S is feasible for D. Let $D = \{p_1, p_2, \cdots p_K\}$. Then there are times $t_1, t_2, \cdots, t_j, \cdots, t_K$ such that, for $1 \leq j \leq K$, if p_j departs at t_j, and does not encounter any other packet, then it arrives at its destination within finite time. Denote the (finite) arrival times by $\alpha_1, \alpha_2, \cdots \alpha_K$.

Denote by P the period of S. We construct a proper departure time as follows. Let p_1 depart at t_1. It thus arrives at α_1. Let $\widehat{\alpha}_1 = \lceil \frac{\alpha_1}{P} \rceil \cdot P$, i.e., we round up the arrival time of p_1 to the nearest end of a period. We set the departure time of p_2 at $\widehat{\alpha}_1 + t_2$. It is immediate that p_2 arrives at $\widehat{\alpha}_1 + \alpha_2$. In general, after fixing the departure time of the j-th packet, we calculate its (finite) arrival time, round it up to the nearest period, and add the value of t_{j+1}; this determines a feasible departure time for the $j+1$-st packet. Since $K < \infty$, this construction sets a departure plan that is proper and whose termination time is finite. □

Lemma 2 *Given a network G, a schedule S with period P and a demand D, S is feasible for D iff there is a proper departure plan Δ whose termination time is at most*

$$T_{\max} = 2K \cdot (|V| - 1) \cdot P + 2K \tag{1}$$

Proof: Suppose that S is feasible for D. Consider a packet p_j that leaves its source s_j at a feasible departure time t_j, and assume that it is the only packet traversing the network. Suppose that by time $t_j + (|V|-1) \cdot P + 1$ the packet has not reached its destination d_j. At all times t, $t_j \leq t \leq t_j + (|V|-1) \cdot P + 1$, define the state of the packet to be the pair (v_j^t, τ^t), where $v_j^t \in V$ is the identity of the current node and $\tau^t = t \mod P$. Since there are only $(|V|-1) \cdot P$ different states for which $v_j^t \neq d_j$, there are times θ_1, θ_2, where $t_j \leq \theta_1 < \theta_2 \leq t_j + (|V|-1) \cdot P + 1$, such that $v_j^{\theta_1} = v_j^{\theta_2}$ and $\tau^{\theta_1} = \tau^{\theta_2}$. Clearly, packet p_j returns to node $v_j^{\theta_2}$ in times $\tau^{\theta_1 + i(\theta_2 - \theta_1)}$, for all integral $i \geq 1$, and it does not visit node d_j in between consecutive visits to $v_j^{\theta_2}$. This means that p_j never reaches its destination (for any choice of the departure time t_j), thus contradicting the feasibility of S with respect to D. Note that our assumption, that p_j traverses the network alone, does not limit the generality of the contradiction, since a colliding message that is eliminated never reaches its destination. We conclude that a packet p_j that leaves its source at a feasible departure time t_j and traverses the network alone reaches its destination by time $t_j + (|V|-1) \cdot P + 1$. By a similar argument, we can also conclude that $t_j \leq (|V|-1) \cdot P + 1$. Thus, by choosing a departure time $\delta_j \leq ((|V|-1) \cdot P + 1) \cdot (2j-1)$ for each packet p_j, $1 \leq j \leq K$, we guarantee that it arrives at its destination no later than at time $t_j + ((|V|-1) \cdot P + 1) \cdot 2j$. We conclude that $\Delta = \{\delta_1, \delta_2, \cdots \delta_K\}$ is a proper departure plan whose termination time is at most $2K \cdot (|V|-1) \cdot P + 2K$.

In the other direction, if there is a proper departure time with finite termination time, then the schedule S is feasible for the demand D. □

The above lemma enables us to consider a finite domain of time, whose size $T_{\max}$ is polynomial in $|V|$, K, and P.

It is clear that if packets are allowed to enter the network freely upon arrival, some of them may be eliminated. In an attempt to maximize throughput, it would be useful to exert some admission control, i.e., forbid the transmission of some of the packets so that throughput is enhanced. More formally, consider the following problem.

Throughput Maximization (TM) Problem: Given a network G, a schedule S, a demand D, and a departure plan Δ, find a maximal set of packets that can be delivered to their respective destinations.

To attack this problem, we observe that the dynamic behavior of the network G with schedule S over a finite domain of time T can be represented by an equivalent static network, called the *Timed-Exploded (TE)* version of G. Note that T is implicitly determined by the other parameters of the TM problem and is bounded by the value $T_{\max}$ as computed in Lemma 2. Specifically, a time-exploded version $TE_{G,S,T}$ of G and S, and of size $T \leq T_{\max}$, is obtained in the following way.

The graph $TE_{G,S,T}$ consists of $T+1$ "layers" $0, \cdots, T$, where each contains $|V|$ nodes, such that there is a $1-1$ correspondence between the nodes of each layer and the nodes in V; for each $v \in V$, and $0 \leq t \leq T$, we denote by v^t the node at level t that corresponds to node v. v^t is said to be a *descendant* of v. The set of edges of $TE_{G,S,T}$ is defined as follows. First, for each $v \in V$, and $1 \leq t < T$, we insert an edge from v^t to w^{t+1}, where w is the next node from v at time t according to schedule S, i.e., $w = k_t^v$. Then, for each packet $p_j = (s_j, d_j) \in D$ we insert an edge from s_j^0 to $s_j^{t_j}$, where t_j is the departure time of p_j for the departure plan Δ. If $t_j = 0$, then we insert an edge from s_j^0 to w^1, where w^1 is the next node from s_j at time 0 according to schedule S.

We remark that the notion of a Timed-Exploded graph was independently discovered by Symvonis and Tidswell [25]. They use a method which they call the multistage method and their graphs are called "multistage graphs".

Consider a path in $TE_{G,S,T}$ that starts at the 0th-level descendant of the source node in G of some packet $p_j \in D$, and ends at a descendant of the destination node of p_j. Such a path in $TE_{G,S,T}$ corresponds to the unique route in G that conforms to the schedule S, departure plan Δ, and upper bound T on the arrival time, and is referred to as a *relevant path*. The graph $TE_{G,S,T}$ has the additional following properties, that can easily be verified:

- $TE_{G,S,T}$ is the union of at most $|V|$ directed relevant paths.
- Two relevant paths in $TE_{G,S,T}$ that meet at some node v^t, continue from v^t together until one of them (or both) reaches its destination. Notice that two such paths correspond to the routes of two packets that collide at node v at time t (causing one of them to be eliminated).

Given these properties, solving problem TM amounts to answering the following question: what is the maximum number of disjoint relevant paths in $TE_{G,S,T}$?

In order to answer this question, we construct the corresponding (undirected) *path graph* of $TE_{G,S,T}$ ($PG(TE_{G,S,T})$) as follows. In $PG(TE_{G,S,T})$, a node corresponds to every relevant path in $TE_{G,S,T}$; there is an edge between every two nodes that correspond to two (relevant) paths in $TE_{G,S,T}$ that are not edge-disjoint.

Clearly, finding the maximum number of relevant edge-disjoint paths in $TE_{G,S,T}$ is equivalent to finding a maximum independent set in $PG(TE_{G,S,T})$, i.e., a maximum set of nodes such that any two nodes in the set are not adjacent. In general, both the problem of finding edge disjoint paths (between various source-destination pairs), and that of finding a maximum independent set, are known to be NP-Complete [8]. Nonetheless, we now show that in our case the problems are tractable, since the graph $PG(TE_{G,S,T})$ is chordal. A graph is called *chordal* if every cycle of length 4 or more contains a chord.

Chordal graphs constitute a graph family for which it is well known [10] that many NP-complete problems become tractable when restricted to it. In particular, a maximum independent set can be computed in linear time in a chordal graph [9, 10].

Lemma 3 *$PG(TE_{G,S,T})$ is a chordal graph.*

Proof: To prove the lemma, we must show that $PG(TE_{G,S,T})$ does not contain a chordless cycle of length 4 or more. Our proof is based on the following observation regarding the path graph. If there is an edge between two nodes π_1 and π_2 of $PG(TE_{G,S,T})$, then there exists a time t' such that the routes corresponding to π_1 and π_2 coincide for all $t > t'$ (until the end of the shorter route).

Assume, to the contrary, that there exists a chordless cycle of length four whose vertices (numbered clockwise) are $\pi_1, \pi_2, \pi_3, \pi_4$. By the above observation, there exist times t_1, t_2, t_3, t_4, one for each edge on the cycle, such that from that time on, the two paths (corresponding to the two vertices on the edge) coincide. Assume, without loss of generality, that t_1, denoting the coincidence time of π_1 and π_2, is the earliest of them. Since π_3 coincides with π_2 (at $t_2 > t_1$), and does not coincide with π_1, we conclude that π_1 terminates before t_2. Consider now π_4. It coincides with π_1 at t_4, and thus $t_1 < t_4 < t_2$, which implies that π_4 coincides with π_2, contrary to the assumption.

In a similar manner, it can be shown that the property holds for cycles of length larger than four. □

Theorem 1 *Problem TM can be solved by an $O(|V| \cdot T)$ algorithm.*

Proof: Since $TE_{G,S,T}$ is the union of at most $|V|$ directed relevant paths, the graph $PG(TE_{G,S,T})$ contains at most $|V|$ vertices. Every relevant path can intersect at most T other relevant paths (based on its time of departure), or at most $|V|$ relevant paths (based on the number of vertices). In other words, every relevant path can intersect at most $\min\{|V|, T\}$ other relevant paths, so that the number of edges in $PG(TE_{G,S,T})$ is at most $O(\min\{|V|^2, |V|T\})$. Therefore, $PG(TE_{G,S,T})$ can be constructed in $O(\min\{|V|^2, |V|T\})$ time. A maximum independent set in $PG(TE_{G,S,T})$ can be computed in linear time, i.e., in $O(\min\{|V|^2, |V|T\})$ time, yielding the theorem. □

Since $T \leq T_{\max} = O(K \cdot |V| \cdot P)$ and $K = O(|V|)$, we conclude that the above solution is polynomial in $|V|$ and P.

A natural question that arises in this context is that of finding a better use of a given schedule, i.e., assuming all packets in D are available at time $t = 0$, when should each depart. More formally:

Schedule Usage (SU) Problem: Given a network G, a network schedule S, and a demand D, find a proper departure plan Δ of minimum termination time.

The decision version of Problem SU is defined as follows. Given a network G, a network schedule S, and a demand D, does there exist a proper departure plan Δ which terminates in time τ? Unfortunately, this problem is intractable. This can be proven using a reduction from 3-SAT. We outline the reduction.

Consider a 3-SAT instance with variables $x_1, \ldots, x_n$ and clauses $C_1, .., C_m$. We construct a graph where we associate a source-sink pair with each variable and each clause. In addition, we have a node z such that each source has a direct edge to z. Let the source (sink) associated with variable x_i be denoted by s_{x_i} (t_{x_i}), and let the source (sink) associated with clause C_j be denoted by s_{C_j} (t_{C_j}). Source s_{x_i} has two outgoing edges, e_{x_i} and $e_{\bar{x}_i}$, (in addition to the edge going to z), where each starts a path ending at sink t_{x_i}. The schedule is such that a packet is routed through e_{x_i} $(e_{\bar{x}_i})$ at time-slot $2i-1$ $(2i)$. At all other time-slots, a packet is routed to z. The idea is that if the packet of s_{x_i} is routed through edge e_{x_i}, the variable x_i is set to "true", else if it is routed through edge $e_{\bar{x}_i}$, then variable x_i is set to "false".

Let clause C_j contain literals x_{j_1}, x_{j_2}, and x_{j_3}. For simplicity of presentation, we assume here that all three literals of C_j are positive. Each source s_{C_j} has three outgoing edges, e_1^j, e_2^j, and e_3^j (in addition to the edge going to z), where each starts a path ending at sink t_{C_j}. The schedule is such that a packet is routed through e_1^j (e_2^j, e_3^j) at time-slot $2j_1 - 1$ $(2j_2 - 1,\ 2j_3 - 1)$. At all other time-slots, a packet is routed to z. The idea is that a packet leaving s_{C_j} at time-slot $2j_k - 1$ $(k = 1, 2, 3)$ collides with a packet leaving source $s_{x_{j_k}}$ at time-slot $2j_k$, i.e., through the edge associated with the setting of x_{j_k} to "false". This can be easily achieved for all clauses by adding dummy nodes and edges, such that paths would intersect at the desired time-slots.

It is not hard to verify that by choosing a large enough period (yet linear in $n+m$), there is a proper departure plan that terminates in less than one period, if and only if there is a satisfying assignment to the 3-SAT instance.

Given that Problem SU is intractable, we describe the following heuristic algorithm (Algorithm HSU) to find a good departure plan, which is based on a greedy approach. The main idea is to start at time $t = 0$, and schedule the maximal number of departures for that time. We then attempt to schedule the maximal number of departures for each successive time slot, using a residual time exploded graph, to avoid elimination by previously scheduled packets. We note that a similar heuristic was used by Symvonis and Tidswell [25].

Algorithm HSU:

1. Initialization: $D' = D$, $TE' = TE_{G,S,T_{\max}}$, $t = 0$.
2. Solve problem TM for D', TE' and a departure plan Δ' such that $t_j = t$ for all packets $p_j \in D'$. Let $\widehat{D}$ be the resulting maximal set.
3. Set $\delta_j = t$ for all packets $p_j \in \widehat{D}$.
4. Remove from TE' all the edges of paths that correspond to the packets of $\widehat{D}$.

5. Set $D' = D' \backslash \widehat{D}$; $t = t + 1$.

6. If $D' \neq \emptyset$ then repeat Step 2, otherwise Stop: the departure plan is $\Delta = \{\delta_1, \ldots, \delta_K\}$.

Algorithm HSU terminates in at most $T_{\max}$ iterations, since at each iteration, at least one path is selected and scheduled individually (Lemma 2 guarantees that even if the packets are routed one-by-one, the number of iterations is at most $T_{\max}$). Note also that algorithm HSU is not restricted to a single packet per source; it will also work if D includes a separate element for each (duplicate) packet.

4 Designing an Optimal Schedule

In this section we consider the complementary problem to that considered in the previous section, namely, how to design a schedule rather than use a given one.

Given demand D and schedule S, call the termination time, corresponding to the solution of Problem SU, the *optimal termination time* of S with respect to D. We refer to the *optimal termination time* for D as the minimum, among all optimal termination times, taken over all possible schedules. A schedule S that induces an optimal termination time (with respect to D) is called an *optimal schedule*. Note that by Lemma 2, the optimal termination time is bounded by equation (1).

We are now ready to present our problem formally:

Schedule Design (SD) Problem: Given a network G and a demand D, find an optimal schedule with respect to D.

In other words, our aim is to find a schedule, with respect to demand D, that achieves the best possible termination time (through the solution of problem SU). Due to the finiteness of the optimal termination time (see equation (1)), problem SD can be transformed into the following decision problem.

T-constrained Schedule Design (TSD) Problem: Given a network G and a demand D, is there a schedule S for which there is a departure plan Δ with termination time at most T?

We note that Problem SD can be optimally solved via Problem TSD, by performing a binary search over the feasible range of T.

4.1 Intractability

Unfortunately, problem TSD (and thus also problem SD) is intractable, as shown by the following theorem.

Theorem 2 *Problem TSD is NP-Hard.*

Proof: We prove the theorem by a reduction to the $3SAT$ problem, which is known to be NP-Complete [8]. We adapt Karp's proof [16] for the intractability of the Disjoint Paths Problem. Consider a version of the $3SAT$ problem, where each literal appears exactly κ times for some constant κ. This is the κ-$3SAT$ problem which is known to be NP-Complete [14]. We show how to transform, in polynomial time, an instance of κ-$3SAT$ into an equivalent instance of TSD. The source-destination pairs in the constructed problem will be in one-to-one correspondence with the clauses in κ-$3SAT$, i.e., pair (s_i, d_i) will correspond to clause C_i.

The graph G of TSD is obtained by joining together a number of subgraphs, each corresponding to a variable. If variable A occurs in clauses $i_1, i_2, \cdots i_\kappa$, and $\bar{A}$ occurs in $j_1, j_2, \cdots j_\kappa$, then the subgraph corresponding to A is as shown in Figure 1 (for simplicity, the figure is for $\kappa = 3$). I.e., for $1 \leq l \leq \kappa$, the "column" of s_{j_l} meets the "row" of S_{i_l} on the leftmost node, then it meets the row of $S_{i_{l \bmod \kappa+1}}$ on the next to the leftmost node, and so on, until at last it meets the row of $s_{i_{l \bmod \kappa-1}}$ on the rightmost node. (If $l \bmod \kappa = 1$, then it meets the row of s_{i_κ} on the rightmost node.) The overall graph is simply obtained by identifying, as a single node, all the occurrences of each s_i (or d_i) in the various subgraphs.

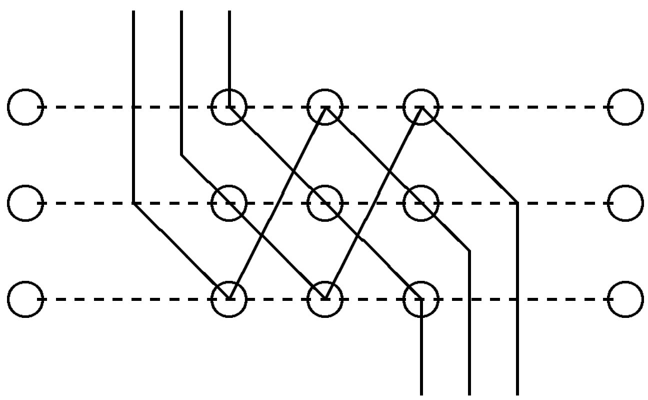

Figure 1: Reduction from 3SAT to TSD

For the above graph G, consider the TSD problem for which each (s_i, d_i) corresponds to a source-destination pair, and $T = \kappa + 1$. It is easy to verify that, for this TSD problem to have an affirmative answer, all packets must depart at $t = 0$. Moreover, it is immediate that routing a packet through a row eliminates the possibility of routing packets in any of the columns of the corresponding subgraph, and vice-versa. Thus, the given conjunctive normal form (of κ-3SAT) is satisfiable, if and only if the TSD problem has a positive answer: if variable x is assigned "true", select the horizontal paths in the subgraph for x, otherwise select the vertical paths. Thus, if either x or $\bar{x}$ occur in clause C_i and x is true in the assignment, then s_i and d_i will be joined in the subgraph for x. In the other direction, it is straightforward that an affirmative answer for the TSD problem implies the satisfiability of the corresponding κ-3SAT problem. Thus, we established a reduction of κ-3SAT to TSD. Since TSD is clearly in NP, this problem is NP-Hard. $\square$

Consider a network topology in which nodes are arranged on a line, such that each pair of adjacent nodes is connected by a bidirectional edge. Let D be such that traffic flows in both ways, i.e., some destinations are located to the right of their sources, while others are located to the left. Dinitz has shown [6] that even in this setting problem SD is intractable.

We note, though, that a simple but efficient heuristic exists for the linear topology: first, all sources that have destinations to their right send (at time 0) their packets. Once these packets arrive, the other sources send (all at once) their packets. It is clear that by using this heuristic packets never collide and the termination time is at most twice that of the optimal termination time.

4.2 Tractable Special Cases

Consider the single destination case, i.e., where all packets have the same destination. Then, problem TSD can be solved in polynomial time in the following way. We construct $TE_{G,T}$ similar to the construction of $TE_{G,S,T}$, as follows. As in graph $TE_{G,S,T}$, $TE_{G,T}$ consists of $T+1$ "layers" $0, \cdots, T$, where each contains $|V|$ nodes, such that there is a $1:1$ correspondence between the nodes of each layer and nodes in V. The edge set of $TE_{G,T}$ is defined differently. Let v, w be vertices in G; there is an edge between every two nodes $v_t, w_{t+1} \in TE_{G,T}$, for $0 \leq t \leq T-1$, iff $(v, w) \in E$.

Thus, for every path in $TE_{G,T}$, a schedule S can be chosen, such that this path can be realized as a relevant path. Notice, however, that there exists a schedule that realizes two given paths in $TE_{G,T}$ (as relevant paths), if and only if the two paths are *disjoint*. This implies that the problem of satisfying a given demand D is equivalent to that of finding a set of disjoint paths, from the sources to the destinations, in the graph $TE_{G,S,T}$. This problem can be cast as a $\{0,1\}$ multicommodity flow problem in $TE_{G,T}$, where edges and vertices of $TE_{G,T}$ are assumed to have unit capacity.

The general discrete multicommodity flow problem is intractable, even in its $\{0,1\}$ version; this is not surprising, since the intractability of TSD was essentially established by adapting Karp's proof of the intractability of the disjoint paths problem. However, in the common destination case, the multicommodity flow problem can be transformed into a single source/single destination max-flow problem, by connecting all sources to a (fictitious) common source (usually referred to as super-source). Hence, for the case of a single destination, TSD can be solved in polynomial time. A similar argument can be made for the single source case.

The above implies that SD can also be solved in polynomial time for the case of a single destination or source. This is achieved by performing a binary search on the value of T. Since the arrival time of each packet, when transmitted alone, is bounded by $M_{G,D}$ (the diameter of the graph with respect to D, see Section 2), it follows that $K \cdot M_{G,D}$ is an upper bound on the value of T. This increases the complexity of the solution by a multiplicative factor of $O(\log(K \cdot M_{G,D})) = O(\log(|V|))$.

4.3 Heuristics

Consider first the special case of directed tree networks where edges are directed from children to parents. The only schedule that can be considered is the trivial one where each node always chooses its unique outgoing edge. We now devise a departure plan, which is not necessarily optimal, but for which we can bound the termination time. This plan will serve us in building a heuristic for general graph topologies.

If the problem has at all a feasible solution, then the destination of each source-destination pair must be on the path between the source and the (common) root of the tree. Consider first a modified problem in which the root of the tree is the destination for all the packets in D. This is a common destination problem for which an optimal departure plan can be found. We use this departure plan for our (unmodified) problem, but route the packets only to their destination (instead of the root). It is not hard to see that this is a feasible departure plan which we now analyze.

Lemma 4 *For common destination directed tree networks (where the destination is the root of the tree), there is a departure plan whose termination time is bounded by* $K + M_{G,D}$.

Proof: We claim that departure times can be scheduled such that packets arrive at the root, one packet per time-slot, after some initialization time which is not greater than $M_{G,D}$. We prove this by induction on the number of vertices in the tree. It clearly holds for a tree containing two vertices. Let r be the root of the tree, let $r_1, r_2, \ldots, r_\ell$ be its neighbors in the tree, and let $T_1, \ldots, T_\ell$ be subtrees such that r_i is the root of T_i. For each subtree T_i, we denote by D_i the portion of the demand D originating within T_i. If the destination of a packet is the root r, then its destination in D_i will be r_i. By the inductive hypothesis, for each subtree T_i, there exists a departure plan with the above property. Since the subtrees interact only in the root r, it is clear that by performing "sequentially" the departure plans computed for each of the subtrees, it is possible to obtain a departure plan in which packets arrive at the root r, one packet per time-slot. Clearly, in the worst case, the first packet will arrive at r after $M_{G,D}$ time slots. □

We refer to the algorithm implied by this proof as the *Train Algorithm*. We note that the above upper-bound also applies to the inverse case, where edges are directed from the root to the leaves, and each destination is on a directed path from the corresponding source. In order to see that, reverse the direction of edges (so that now the root is a sink on the tree), and reverse the roles of sources and destinations. We thus have:

Corollary 1 *If G is a directed tree, in which the orientation is either towards the root or away from the root, and if there is a directed path from each source to each destination, then the Train Algorithm provides a departure plan with termination time of at most $K + M_{G,D}$.*

Consider now the case of an undirected tree. The problem is that since there is no single orientation, source-destination pairs may need conflicting orientations. However, we show how the Train Algorithm can be used in this case such that termination times are still within a reasonable bound. Similar ideas have been used in Up/Down Routing [22].

We begin by choosing a node in the above tree, which we refer to as the *separator*. (A good way to choose a separator is discussed following Lemma 5). When the separator is removed, the tree is separated into components where each is a tree; denote by $C_1, C_2, \cdots, C_m$ these components with the separator attached to each. Denote by k_{ij} $(i \neq j)$ the number of packets whose source belongs to C_i while its destination belongs to C_j. The total number of inter-component packets is therefore $f = \sum_{ij} k_{ij}$. We are interested in bounding the time required to deliver this intercomponent traffic, which we do using the following technical lemma.

Lemma 5 *The entire intercomponent traffic f for a given set of components $C_1, \ldots, C_m$ can be delivered within $f + 2M_{G,D} \log f$ time steps.*

Proof: Let $f_1 = f$. We partition the components into two sets A_1 and B_1, and let n_1 be the number of packets to be routed between these two sets. The partition into A_1 and B_1 is made such that at least half of the intercomponent packets flows between them, or, in other words, $n_1 \geq f_1/2$. (The standard 2-approximation algorithm for the MAX-CUT problem achieves this.) In the first phase, first direct all the edges in the components of A_1 towards the separator, and the edges in the components of B_1 away from the separator, and send all the packets from A_1 to B_1. Then, reverse the edge orientation and send all the packets from B_1 to A_1. By corollary 1 this operation will take at most $n_1 + 2M_{G,D}$ time steps.

In the next phase we need to transfer the traffic among components of A_1 and among those of B_1. The number of packets to be dealt with at this phase is $f_2 = f_1 - n_1 \leq f_1/2$. This cannot be done completely in parallel, since the separator node is common to all of these components, and therefore becomes a bottleneck. Our approach is to keep the separator as busy as possible. Partition A_1 into two subsets, A_2 and B_2, in the same manner as before, i.e., such that at least half of the packets to be routed among the components of A_1 is carried across the boundary between A_2 and B_2. Similarly divide B_1. If n_2 denotes the total amount of this traffic, then $n_2 \geq f_2/2$. In the way described in the proof of Lemma 4, this phase can be completed in $n_2 + 2M_{G,D}$ time steps.

Continuing in this manner, there are at most $\log f$ phases, so the entire process is done within

$$\sum_i (n_i + 2M_{G,D}) \leq f + 2M_{G,D} \log f$$

which completes the proof. □

We now proceed to provide a good solution for the case of an undirected tree by a recursive application of the Train Algorithm. It is known that every tree has a separator, that if removed, separates the tree into components, where each component is a tree containing at most half of the nodes of the original tree [17]. In a similar way it can be shown that every tree, some of whose nodes are sources, has a separator such that if removed, separates the tree into components, such that each component is a tree containing at most half of the sources of the original tree. We choose such a node as our separator.

We first send the intercomponent packets in the manner described in Lemma 5 and then apply the whole procedure, recursively, to the (undirected) tree defined by each component. Note that, in this case, different components do not share nodes, so the recursion can be applied concurrently to all components, i.e., packets will be routed concurrently as part of the recursive solution. Call this scheme the *Undirected Train Algorithm.*

Lemma 6 *The Undirected Train Algorithm guarantees a termination time of*

$$K + 2M_{G,D} \log^2 K.$$

Proof: Let $\Gamma(k)$ be the execution time of the Undirected Train Algorithm for the given tree topology and for k packets. By Lemma 5 the intercomponent packets can be delivered within

$$\sum_{ij} k_{ij} + 2M_{G,D} \log(\sum_{ij} k_{ij}) \leq \sum_{ij} k_{ij} + 2M_{G,D} \log k$$

Due to the manner in which the separator is chosen, each component has at most $k/2$ packets. Hence, the recursive structure of the algorithm gives

$$\Gamma(k) \leq \Gamma\left(\frac{k}{2}\right) + \sum_{i,j} k_{ij} + 2M_{G,D}\log k$$

We note that the summation of the term k_{ij} in the recursive expression cannot exceed k. Also, since there are at most $\log k$ stages to the recursion, the third term appears at most $\log k$ times. Thus $\Gamma(k)$ is bounded by $k + 2\log^2(k)M_{G,D}$. For the entire graph, $k = K$, which yields the required result. □

Lemma 7 *The Undirected Train Algorithm terminates in $O(|V|^2 \log^2 K)$ steps.*

Proof: Consider the selection of a separator as the initiation of a new iteration of the Undirected Train Algorithm. At the beginning of an iteration, a separator should be chosen, and this can be done in $O(|V|)$ steps. Next, the number of intercomponent packets (i.e., the k_{ij}'s), should be computed, consuming $O(K)$ steps. Next, the iteration breaks, recursively, into $O(\log K)$ sub-iterations. At each sub-iteration a group of $O(|V|)$ components has to be partitioned into two sets, and it is easy to see that this can be done in $O(|V|^2)$ steps. The sub-iteration concludes by executing the (directed) Train Algorithm on the directed tree implied by the chosen sets. It is easy to see that an execution of the Train Algorithm on a directed tree with $O(|V|)$ nodes and packets can be performed in $O(|V|)$ steps. Thus, each sub-iteration consists of $O(|V|^2)$ steps. Since there are $O(\log K)$ sub-iterations in an iteration, we conclude that each iteration terminates in $O(|V|^2 \log K)$ steps. Finally, by the recursive structure of the Undirected Train Algorithm, it has $O(\log K)$ iterations, and the result follows. □

The Undirected Train Algorithm suggests a heuristic algorithm (HSD) for any general topology: compute a spanning tree Θ for G, and then run the Undirected Train Algorithm on Θ. This gives us:

Theorem 3 *For any graph G and demand D, algorithm HSD terminates in $O(|V|^2 \log^2 K)$ steps and provides a departure plan whose termination time is bounded by $K + 2|V| \log^2 K$.*

Proof: The first claim follows from Lemma 7. The second claim follows from Lemma 6, since $M_{\Theta,D} < |V|$. □

We note that, on a general graph, algorithm HSD makes use of only $|V| - 1$ edges. While the above analysis provides a guaranteed performance on general graphs, further improvements can be achieved by perturbing the output of HSD, in an attempt to make use of other, non-tree edges.

5 Conclusion

In this paper we investigated efficient schemes for high-speed routing, that rely on techniques of both hot-potato and scheduled routing. Such schemes are advantageous because they require neither packet buffering nor routing decisions based on packet contents. Our analysis was carried in two directions. First, we indicated how routing should be planned after a schedule has been fixed, and then analyzed the problem of designing an efficient schedule. For some of the problems, tractable solutions that are optimal have been found. Other problems were shown to be intractable. For these problems we presented heuristic solutions. In particular, for the schedule design problem we obtained a recursive algorithm for which an efficient performance bound has been proved.

Several problems are still left open for further research. Additional versions of the general problem are the most obvious. Obtaining better heuristics is another one. For example, the heuristic algorithm presented for the design problem on a general topology currently achieves a bound of $K + 2|V|\log^2 K$. We would like to improve this bound to that of $O(K + M_{G,D}\log^2 K)$ which requires that $M_{\Theta,D} = O(M_{G,D})$. Whether such a spanning tree Θ exists and whether it can be constructed seem to be non-trivial problems.

Acknowledgement

We would like to thank Efim Dinitz for several useful discussions.

References

[1] A. Acampora and S. Shah. Multihop lightwave networks: a comparison of store-and-forward and hot-potato routing. In *Proceedings of IEEE INFO-COM*, pages 10–19, 1991.

[2] A. Barnoy, P. Raghavan, B. Schieber, and H. Tamaki. Fast deflection routing for packets and worms. In *Proceedings of the 12th ACM Symposium on Principles of Distributed Computing*, pages 75–86, 1993.

[3] A. Ben-Dor, S. Halevi, and A. Schuster. Potential function analysis of greedy hot-potato routing. To appear in *Mathematical Systems Theory*, 1997. Preliminary version in *Proceedings of the 13th Symposium on Princi ples of Distributed Computing*, 1994, pages 225–234.

[4] I. Cidon, I. Gopal, and S. Kutten. New models and algorithms for future networks. In *Proceedings of the 7th ACM Symposium on Principles of Distributed Computing*, pages 74–89, 1988.

[5] I. Cidon, S. Kutten, Y. Mansour, and D. Peleg. Greedy packet scheduling. *SIAM Journal on Computing*, 24:148–157, 1995.

[6] E. Dinitz. Private communication, May 1993.

[7] U. Feige and P. Raghavan. Exact analysis of hot-potato routing. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, pages 553–562, November 1992.

[8] M. Garey and D. Johnson. *Computers and Intractability.* Freeman, San Francisco, 1979.

[9] F. Gavril. Algorithms for minimum coloring. *Siam J. Computing*, 1:180–187, 1972.

[10] M. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, New York, 1980.

[11] A. Greenberg and J. Goodman. Sharp approximate models of deflection routing in mesh networks. In *Proceedings of IEEE INFOCOM*, pages 307–318, 1983.

[12] A. Greenberg and B. Hajek. Deflection routing in hypercube networks. *IEEE Transactions on Communications*, 40:1070–1081, 1992.

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

[14] A. Itai. Two-commodity flow. *Journal of the ACM*, 25:596–611, 1978.

[15] C. Kaklamanis, D. Krizanc, and S. Rao. Hot potato routing on processor arrays. In *Proceedings of the 5th ACM Symposium on Parallel Algorithms and Architectures*, pages 273–282, 1993.

[16] R. Karp. On the complexity of combinatorial problems. *Networks*, 5:45–68, 1975.

[17] B. Korte, L. Lovasz, and H. Promel. *Paths, Flow, and VLSI-Layout.* Springer-Verlag, Berlin New-York, 1990.

[18] Y. Mansour and B. Patt-Shamir. Greedy packet scheduling on shortest paths. *Journal of Algorithms*, 14:449–465, 1993.

[19] N. Maxemchuck. Comparison of deflection and store and forward techniques in the manhattan street and shuffle exchange networks. In *Proceedings of IEEE INFOCOM*, pages 800–809, 1989.

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

[21] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *IEEE Computer*, 26(2):62–75, February 1993.

[22] P. Palnati, E. Leonardi, and M. Gerla. Deadlock-free routing in an optical interconnect for high-speed wormhole routing networks. In *Proceedings of the 1996 International Conference on Parallel and Distributed Systems*, Tokyo, June 1996.

[23] C. Partridge. *Gigabit Networking.* Addison-Wesley, 1994.

[24] P. Rivera-Vega, R. Varadarajan, and S. Navathe. Scheduling data redistribution in distributed databases. In *Proceedings of the 6th International Conference on Data Engineering*, pages 166–173, Los Angeles, February 1990.

[25] A. Symvonis and J. Tidswell. An empirical study of off-line permutation packet routing on two-dimensional meshes based on the multistage routing method. *IEEE Transactions on Computers*, 45(5):619–625, May 1996.

[26] T. Szymanski. An analysis of hot potato routing in a fiber optic packet switched hypercube. In *Proceedings of IEEE INFOCOM*, pages 918–925, 1990.

[27] Y. Yemini and D. Florissi. Isochronets: A high-speed network switching architecture. In *Proceedings of IEEE INFOCOM*, pages 740–747, 1993.

[28] Z. Zhang and A. Acampora. Performance analysis of multihop lightwave networks with hot potato routing and distance age priorities. In *Proceedings of IEEE INFOCOM*, pages 1012–1021, 1991.