The Travelling Miser Problem

David Breitgand, Danny Raz, Yuval Shavitt

Abstract— **Various monitoring and performance evaluation tools generate considerable amount of low priority traffic. This information is not always needed in real time, and thus could often be delayed by the network without hurting functionality. This paper proposes a new framework to handle this low priority, but resource consuming traffic in such a way that it will incur a minimal interference with the higher priority traffic, and thus improve the network goodput. The key idea is to allow the network nodes to delay data by locally storing it. This can be done, for example, in the *Active Network* paradigm.**

In this paper we show that the active network paradigm can improve the network's goodput dramatically even if very simple scheduling algorithm is used. To obtain minimal cost schedules we define an optimization problem that we call the *travelling miser problem*. We study primarily the *on-line* variant of this problem, which is of greater practical interest. For this problem we develop an enhanced scheduling strategy, study its characteristics in a restricted case, and evaluate its performance through a rigorous simulation study.

I. Introduction and Motivation

In some important management applications, such as large scale monitoring for the purposes of accounting, planning, performance evaluation *etc.*, considerable traffic is generated. This traffic shares the same network infrastructure with user applications. From the point of view of these other services this traffic is an overhead since it is of no immediate interest at the user level.

Although there exist various kinds of high priority network management traffic that may have quite stringent timing requirements, (*e.g.*, in pro-active management applications), in many other cases the management traffic may be much less demanding in terms of timing. Examples include various logging facilities that subsequently transfer the accumulated data over the network, software distribution facilities, distributed backups, accounting and billing, long term monitoring of the large-scale systems *etc.* In all these cases the applications have a considerable maneuvering space when actually to use the network.

Since in most cases the end-users are oblivious to the services described above, and thus their impact on the user-visible network services should be minimized. For instance, it would be beneficial to preempt some management traffic at the time of network congestion and schedule it for later transmission when the transient congestion condition is over.

David Breitgand is with the School of Engineering and Computer Science of the Hebrew University, Jerusalem, Israel. Email: davb@cs.huji.ac.il

Danny Raz is with the Computer Science Department, Technion, Haifa, Israel. Email: danny@cs.technion.ac.il. His work was supported, in part, by the fund for the promotion of the research at the technion.

Yuval Shavitt is with the Department of Electrical Engineering — Systems, Tel-Aviv University, Tel-Aviv, Israel. Email: shavitt@eng.tau.ac.il

This work was done in part when the authors were at Bell Labs, Lucent Technologies, Homldel, USA.

Thus, it is useful to differentiate between the higher priority user-visible traffic and the lower-priority management traffic. It is important to stress, however, that the terms high-priority and low-priority refer both to best effort traffic with different timing constraints. The user traffic, say HTTP packets, has to arrive at its destination within a short period of time (2-3 seconds for the HTTP example) while the low-priority management traffic can be delayed much longer. It is important to note that both the high-priority and the low-priority traffic compete for the same limited amount of network resources.

Given this model, it is our objective to improve a *goodput* of the network by preferring the high priority (user) traffic over the low priority (management) traffic at times of high load, if timing constraints of the management traffic are flexible enough to allow this. In other words, if there are plenty of resources in the network there is no difference between the low and high priority traffic. However, when the resources become scarce the user traffic is given a priority over the management traffic. In this work the resource that interests us is bandwidth.

We characterize every management packet originating from a management agent, the *source*, by a single parameter: *deadline*. This parameter defines when the packet should arrive at the manager station, the *destination*. In this work we are not concerned with the way management data is obtained. We simply view management applications as producers of data being a subject to the deadline constraint.

Based on this constraint we attempt to create an individual *itinerary* for every low-priority packet, along the existing routing path between the source and the destination in such a way that the packet would meet its deadline and at the same time would incur minimal additional load on the routing elements and links along the path. This, of course, can be done only when we allow the packet to "park" at internal nodes for some time.

One framework in which this is possible is the *active networks* paradigm. This paradigm allows to equip each packet with an "autonomous intelligence" enabling the packets to react to the network conditions much like the rational car drivers do when they react to the road traffic conditions. In particular, an active packet is capable of detaining itself in the network in case of unfavorable conditions, scheduling its transition for a later time. Storing a packet in the network using the switching elements themselves is, obviously infeasible. Fortunately, this need not to be the case since active packets may be diverted to a special purpose machine, that is decoupled from the switching element itself, e.g., the *active engine* in [1].

Figure 1 shows a very simple algorithm that can be used by packets in active network to achieve the above objectives.

A link is considered congested when its utilization crosses some threshold value. However, it is important to notice that

```
while next link is congested
      wait for time interval t and
      check again
proceed to next hop.
```

Fig. 1. Simple Algorithm using the Intermediate Parking Ability of Active Networks.

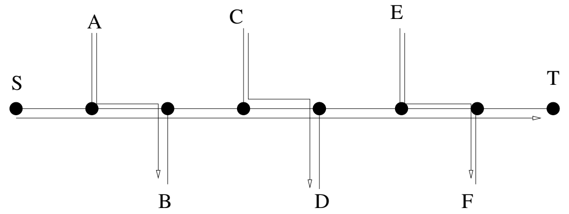

Fig. 2. A simple cross traffic scenario

different threshold values may be configured for low-priority and high priority traffic. Thus, these administratively defined threshold values can serve as a network management mechanism that forces the low priority traffic to "give way" to the high priority traffic when the competition over the resources becomes acute. When this mechanism for prioritizing the traffic is used, an additional advantage of the active network paradigm would be a higher reliability for the low-priority traffic.

Consider the scenario described in Figure 2. The source S is trying to send some monitoring information to the target T. There is cross traffic between A and B, and E and F respectively, which alternates, *i.e.*, whenever the cross traffic from A to B stops, the traffic from E to F starts. In this case every attempt of S to send the information to T will fail[1] as the packets will experience loss due to the cross traffic that raises utilization of the links over the low-priority threshold. However, our simple algorithm will just park at the intermediate node waiting for the cross traffic from E to F to abate, and then will continue to T. If delays introduced by waiting is not too large, the low-priority information will arrive in time for the management application to be used, and the user traffic is unaffected. This simple example shows that indeed the extra functionality introduced by the active network paradigm, adds an extra value to the network.

It is reasonable to assume that the timing constraints of the management traffic will permit a practical usage of the above scheme because while the round trip time is typically less than a second, the typical management traffic deadlines for non-real time traffic is in the range of minutes to hours. This provides the required maneuvering space for the active packets.

We formulate the problem of creating the low-cost parking schedules for the low-priority traffic in abstract terms as a graph theoretic problem and study its various variants. We call this problem the *traveling miser problem* because the behavior of packets resemble a strategy of a traveler that is allowed to make stops in hotels offering different time-dependent prices along some route, trying to keep the total cost of the trip low.

[1]E.g., due to application level time out, after too many retransmission attempts.

We distinguish between the *on-line* variant, in which decisions should be made based on the partial information available locally at a given time, and the *off-line* variant in which all information about the network's behavior is known in advance.

Although solutions for both variants of the problem are applicable to the management applications, the on-line variant is of greater practical interest. For the off-line variant we explain its connection with other work done on the minimal weight path problem in time-dependent networks. For the on-line variant we present a scheduling algorithm that, not only minimizes the interference of the management traffic with the user traffic (which is already achieved by the simple strategy above), but also minimizes the overhead on the active network itself. We prove that for the restricted case in which only one link is congested at a time, the cost our algorithm incurs on the active nodes is logarithmically competitive with respect to an optimal off-line algorithm. We farther demonstrate our algorithm usefulness by studying its performance under other, more complex, network conditions and comparing it to to the simple active algorithm of Fig. 1.

The main contributions of this work are as follows.

- Lower priority, "overhead" traffic, is under scrutiny, in contrast to the usual case when the higher priority traffic is in the center of the problem.
- We show that additional internal node capabilities (such as in the Active Network paradigm) indeed increase the network functionality.
- We formally define the relevant optimization problem, study it in the on-line case, and present a $O(\log B)$ competitive on-line algorithm for a restricted variant of it.
- We evaluate the performance of our algorithms under realistic scenarios, and identify through an extensive simulation study the most significant network parameters in this case.

The rest of the paper is organized as follows. In the next section we formally define the model. In Section III we formulate the problem, and then in Section IV we describe and analyze the on-line and off-line algorithms. In section V we present the result of our simulation study, then we survey the related results and provide some conclusions.

II. Model

The motivation of the model presented in this section is to allow reasoning about the load of active nodes and the cost associated with it. This model enables the definition, creation, and analysis of an optimization heuristics (see Section IV) that would lower the overhead imposed on network nodes by accommodating the detained low-priority traffic.

Following [2] we consider a bi-directional *time-dependent* network
$G = (V,\ E,\ W,\ P,\ L)$ with $V = \{1, 2, ..., n\}$ being a set of nodes, $E \subseteq V \times V$ being a set of links, and W, P, L being a set of *link weights*, *node parking weight densities*, and *link delays* respectively.

$W = \{w_{i,j}(t) \mid (i,j) \in E\}$, $P = \{p_i(t) \mid i \in V\}$ are sets of time-dependent functions. Let us examine each of them in turn.

Link weight function $w_{i,j}(t)$, is a non-negative function defined over $t \in \mathbf{I}^+$, $w_{i,j} \ : \ \mathbf{I}^+ \rightarrow \mathbf{R}^+$. It represents the cost of traversing link (i,j) at departure time t.

Parking weight density function, $p_i(t) \ : \ \mathbf{I}^+ \rightarrow \mathbf{R}^+$, refers to the cost of spending one time unit at node $i \in V$ at time instance t. It may be beneficial to spend some time at a node before moving to another one. If a commodity arrives at node i at time t_1 and departs from the node at time t_2 where $0 \leq t_1 \leq t_2 < \infty$, then the *parking weight* function $P_i(t_1,t_2) \stackrel{\mathrm{def}}{=} \sum_{t=t_1}^{t_2} p(t)$.

The time it takes to traverse some link (i,j) is called *link delay* and denoted $d_{i,j}$. We assume constant, strictly positive, and finite delays, *i.e.*, $\forall t \in \mathbf{I}^+$ and $\forall (i,j) \in E$: $d_{i,j}(t) = d_{i,j} \in \mathbf{I}^+$, and $d_{i,j} < \infty$.

Definition 1: **Simple Topological Path**:
Let $R \subseteq G$ be a sub-network of G where $R \stackrel{\mathrm{def}}{=} (V_R \subseteq V,\ E_R \subseteq E,\ W_R \subseteq W,\ P_R \subseteq P,\ L_R \subseteq L)$. We say that R is a ***simple topological path*** if $V_R = \langle v_0, ..., v_{n-1} \rangle$ is an ordered sequence of nodes from V where every two nodes v_i, v_{i+1}, $i \in [0, n-1]$ are connected by a single bi-directional link $(v_i, v_{i+1}) \in E_R$. The node v_0 is termed the ***source***, and is denoted by ***s***. The node v_{n-1} is termed the ***destination***, and is denoted by ***d***.

For the sake of brevity, we denote the path either as $R = \langle s \equiv v_0, v_1, ..., v_{n-1} \equiv d \rangle$, specifying only the vertices comprising it, or simply as $R = (s,d)$, specifying only the source and destination. We also refer to R simply as *path* wherever this does not lead to any ambiguity. For convenience we assume for every node in the path fictitious self-link with delay 1, and weight being the parking weight density of this node. Traversing such a link correspond to parking at the node for one time unit.

Now we are ready to define the *Traveling Miser Problem (TMP)* informally as following. There is a miser that travels along a given simple topological path $R \subseteq G$. At time instance 0, the miser is at $s \in R$. The miser is required to reach destination $d \in R$ within the integer number $\mathcal{D}$ of time units, $0 \leq \mathcal{D} < \infty$, called *deadline*.

At any given time instance t, the miser is at some node $v_j \in R$, facing the following options for the next time unit:

- stay at the node v_j and pay $p_j(t)$ for parking;
- travel one link in the forward direction(*i.e.*, towards the destination), and pay $p_j(t) + w_{j,j+1}(t)$ for commuting;
- travel one link in the reverse direction(*i.e.*, towards the source)and pay $p_j(t) + w_{j,j-1}(t)$ for commuting;

The traveling miser problem is to devise a strategy that allows reaching the destination within $\mathcal{D}$ time units moving along a given path in either direction, and paying the minimal cumulative cost for both commuting and parking.

In order to formulate the problem and to reason about it more formally, we need a number of definitions. We prefer to keep them all together for the reader's convenience.

Definition 2: **Itinerary**:
For the given $s, d \in R$ an ***itinerary*** $\mathcal{I}(s,d) \stackrel{\mathrm{def}}{=} \langle s \equiv v(t_0), v(t_1), v(t_2), ..., v(t_a) \equiv d \rangle$ such that

- $\forall i \in [1,a] \;\; 0 \leq t_i \leq t_{i+1} < \infty$;
- $\forall i \in [1,a] \;\; v_{t_i} \in R$;
- $\forall v(t_i), v(t_{i-1}) \in \mathcal{I}(s,d) \;\; (v(t_{i-1}), v(t)) \in R$, and $t_i = t_{i-1} + d_{v(t_{i-1}),v(t_i)}$.

Thus, $\mathcal{I}(s,d)$ is, possibly a non-simple, path from source s to destination d consisting of the original path links plus the fictitious self-links as described above.

In practice, we are interested in itineraries for given source and destination that visit the destination only once. We call such itineraries *simple*.

Definition 3: **Arrival Time and Earliest Arrival Time**:
The value of t such that $v(t) \in \mathcal{I}(s,k)$, and $v(t) \equiv k \in R$ is termed ***arrival time*** to k and denoted $t_a(k)$. Minimal such value is called the ***earliest arrival time*** and denoted $t^*(k)$.

For the sake of brevity we refer to an itinerary whose arrival time to d is $t \in \mathbf{I}^+$, and $t < \infty$ as ***t-termination itinerary***.

Definition 4: **Length of an itinerary**: the number of links in an itinerary, denoted $L(\mathcal{I}(s,d))$.

Definition 5: **Parking Itinerary**:
For some node $j \in R$, an itinerary that consists only of $0 \leq t < \infty$ fictitious links (j,j) is referred to as ***parking itinerary (or simply parking)*** at j for time t, and denoted $\tilde{\mathcal{I}}(j,j)^t$.

Definition 6: **Itinerary Weight**:
For itinerary $\mathcal{I} = \langle s, v(t_1), ..., \ v_(t_a - 1), \ d \rangle$, ***itinerary weight*** is
$W(\mathcal{I}) \stackrel{\mathrm{def}}{=} \sum_{t=t_0}^{t_a(d)-1} w_{v(t),v(t+1)}(t)$.

Definition 7: **Itinerary Delay**:
For itinerary $\mathcal{I} = \langle s, v(t_1), ..., \ v_(t_a - 1), \ d \rangle$, ***itinerary delay*** is $D(\mathcal{I}) \stackrel{\mathrm{def}}{=} \sum_{t=t_0}^{t_a(d)-1} d_{v(t),v(t+1)}(t)$.

We are interested only in *finite* itineraries.

Definition 8: **An Itinerary is finite** if and only if:

- ***Itinerary length is finite***: $L(\mathcal{I}(s,d)) < \infty$;
- ***Itinerary weight is finite***: $W(\mathcal{I}(s,d)) < \infty$.

Definition 9: **Hop Distance**:
The number of links in the shortest finite itinerary $\hat{\mathcal{I}}(s,k)$ between the two nodes $s, k \in R$ is termed ***hop distance*** between s and k. Hop distance is denoted $HD(s,k)$.

Note that the hop distance in time-dependent network not necessarily equals the number of links between the two nodes in a given simple topological path, but is *at least* the latter. This is because the itinerary that equals the simple topological path, may be infinite if the link weight functions can assume infinite values.

Definition 10: **Viable Itinerary**:
A simple itinerary $\mathcal{I}(s,d)$, $s, d \in R$ is ***viable*** for a given deadline $0 \leq \mathcal{D} < \infty$, being an integer number of time units, if and only if:

- ***Itinerary is finite***;
- ***Itinerary is timely***: $D(\mathcal{I}(s,d)) \leq \mathcal{D}$.

Definition 11: **Reachable Node**:
Let T be an integer number of time units. Let R be some path in a time-dependent network G, and t be some time instance. A node $v_i \in R$ is ***reachable*** at time t within T time units from another node $v_j \in R, \ j \neq i$ if and only if there exists at least one simple viable itinerary $\mathcal{I}(R')$ for the deadline T and a subpath $R' = \langle v_j, ..., v_i \rangle$, where $R' \subseteq R$.

Definition 12: **Time Distance**:
The delay of a shortest finite itinerary $\hat{\mathcal{I}}(i,j)$:, $D(\hat{\mathcal{I}})$ is called

time distance between the nodes i and j, and denoted $TD(i,j)$.

A. Properties of Itineraries

This subsection describes the properties of itineraries that are useful for some proofs. Let & denote the concatenation operation.

Property 1: $\forall \mathcal{I}_1, \mathcal{I}_2 \;\; W(\mathcal{I}_1 \& \mathcal{I}_2) = W(\mathcal{I}_1) + W(\mathcal{I}_2)$.

Property 2: $\forall \mathcal{I}_1, \mathcal{I}_2 \;\; D(\mathcal{I}_1 \& \mathcal{I}_2) = D(\mathcal{I}_1) + D(\mathcal{I}_2)$.

Property 3: $\forall \mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3$ if $W(\mathcal{I}_1) \leq W(\mathcal{I}_2)$ and $W(\mathcal{I}_2) \leq W(\mathcal{I}_3)$, then $W(\mathcal{I}_1) \leq W(\mathcal{I}_3)$.

Property 4: $\forall i,j \in R(s,d)$: $HD(s,i) \leq HD(s,j)$, $\forall \mathcal{I}(i,d)$, $\forall \mathcal{I}(j,d) \;\; W(\mathcal{I}(j,d)) \leq W(\mathcal{I}(i,d))$.

Property 5: $\forall i,j \in R(s,d)$: $HD(s,i) \leq HD(s,j)$, $\forall \mathcal{I}(i,d) \, \forall \mathcal{I}(j,d) \;\; D(\mathcal{I}(j,d)) \leq D(\mathcal{I}(i,d))$.

Property 6: A node is always reachable from itself within 0 time units by the ***empty*** itinerary and $TD(j,j) = 0 \; \forall j \in R$.

Property 7: If node i is reachable from a node j, $i \not\equiv j$, $i,j \in R$, within $0 \leq T < \infty$ time units and $\forall i,j \in R: \; d_{i,j} = d_{j,i}$ then j is also reachable from i within T time units.

Property 8: Let $R = \langle 0, 2, ..., n-1 \rangle$ be a given path. $\forall \mathcal{I}(R)$

$$W(\mathcal{I}) \geq \sum_{i=0}^{n-1} \min_t \{w_{i,i+1}(t)\}.$$

Note that the lower bound for itinerary weight given by Property 8 may be unattainable.

III. Problem Formulation

Now we can formally formulate the Traveling Miser Problem as follows.

For a given path $R(s,d) \subseteq G$***, where*** G ***is a time-dependent network, and a finite deadline*** $\mathcal{D} \in \mathbf{I}^+$***, find a simple viable itinerary*** $\mathcal{I}(s,d) = \langle s \equiv v(t_0), v(t_1), ..., v(t_a) \equiv d \rangle$ ***for which*** $W(\mathcal{I})$ ***is minimal.***

A specific variant of TMP that corresponds to scenarios in which the network is stable most of the time, but some links may become congested for certain time periods after which they recover, is of special interest. This variation is called *k-block recoverable TMP*. In this variation of the model all parking weight densities are assumed to be constant in time $\forall t \leq \mathcal{D}$.

1) K-block Recoverable TMP: First, we define the notion of a *finite block*.

Definition 13: **Finite Block:**
Let $R(s,d)$ be a given time-dependent path. If $\exists t_1, t_2 \in \mathbf{I}^+ :$ $0 \leq t_1 \leq t_2 < \infty$ and $\exists i,j \in R(s,d)$ such that $\forall t_1 \leq t \leq t_2$ $w_{i,j}(t) \equiv \infty$, we say that a ***finite block*** of duration $t_2 - t_1$ occurs at link (i,j) at time instance t_1 (***block start time***). The link ***recovers*** at time t_2 (***block termination time***).

Definition 14: **Blocking Time**:
Let $R(s,d)$ be a time-dependent path. Let $\langle b_1, b_2, ..., b_k \rangle$ be an ordered sequence of pairs, where each pair represents a single finite block by its start and termination times: $\forall i \in [1,k] \; b_i \stackrel{\text{def}}{=} \langle t_i^{start}, t_i^{end} \rangle$ where $0 \leq t_{i-1}^{end} \leq t_i^{start} \leq t_i^{end} < \infty$. Given a block sequence, the blocking time, $B \stackrel{\text{def}}{=} \bigcup_{i=1}^{k} [t_i^{start}, t_i^{end}]$.

Now we define the k-block recoverable traveling miser problem as following.

Given a deadline $\mathcal{D} \in \mathbf{I}^+$***:*** $0 \leq \mathcal{D} \leq \infty$***, transformed time-dependent path*** $R(s,d)$***, block sequence*** $\langle b_1, b_2, ..., b_k \rangle$***, with the blocking time*** B, $\forall (i,j) \in R$ ***let link weight function***

$$w_{i,j}(t) \stackrel{\text{def}}{=} \begin{cases} \textbf{\textit{constant}}\; 0 < c_{i,j} < \infty & \textbf{\textit{if}}\; t \notin B \;\textbf{\textit{or}}\; i \equiv j \\ \infty & \textbf{\textit{otherwise}} \end{cases}$$

The k-block recoverable traveling miser problem is to find a simple viable itinerary $\mathcal{I}^*(s,d)$ ***such that*** $\forall \mathcal{I} \neq \mathcal{I}^*$***:*** $W(\mathcal{I}^*) \leq W(\mathcal{I})$***.***

The relation of this problem to the actual network scenarios is straightforward. The infinite weight of a link corresponds to the link congestion. The level of link utilization that should be interpreted as a congestion is controllable through the administrative means.

As one can easily verify, the k-block recoverable TMP is just a special case of the general one.

A. On-Line and Off-Line Problems

TMP can be formulated either as an *off-line* problem, or as an *on-line* one. In the off-line problem we assume that the input includes the full knowledge about the values of link weight functions along the path for any given instance of time.

The off-line variant is of interest for advanced planning of the optimal transmission schedules for the applications described in Section I based on the long observed, relatively static load patterns. This is useful, e.g., for the traffic that is generated in regular hours of the day.

The on-line variant is interesting for optimizing the less regular transmissions of the management data because it allows to take the momentary network conditions into account.

In the strictly on-line variant of the problem the values of time-dependent link weight functions are not known in advance, the miser has only a local knowledge about the path, and the current value of a link weight observed by the miser for a certain link does not tell anything about its future value.

An efficient algorithm for solving the strictly on-line general TMP seems to be unlikely.

Fortunately, the general variant of TMP is of less practical importance since it corresponds to the cases when the network is highly unstable. In most practical applications, these instability periods may be assumed to be rare and transient, meaning that most of the time the link weight functions assume finite constant values, except for some instability periods. Therefore we focus on the k-block recoverable TMP that corresponds to this practical case.

In what follows we will focus our attention on the off-line and on-line variants of k-block recoverable TMP, referring to it simply as TMP, wherever ambiguity does not arise.

IV. Algorithms

This section presents the algorithms for solving k-block recoverable TMP.

A. Off-Line Problem

The algorithm for finding minimal weight path in a general time-dependent network presented in [2] solves the general off-line traveling miser problem (as a special case) in polynomial time for any constant deadline $0 \leq T < \infty$. Even though [2] solves the problem for the case of continues (or piece-wise continues) time-dependent weight and delay functions, the same algorithm may be used for our discrete case.

Since the off-line k-block recoverable TMP is a special case of the general off-line TMP, the algorithm presented in [2] also solves the off-line k-block recoverable TMP in polynomial time for any finite deadline.

It is useful to highlight some key ideas from the solution presented in [2].

The algorithm is based on the following *concatenation property* of the optimal (with respect to weight) time-dependent paths.

Lemma IV.1: **Time-Dependent Path Concatenation Property**

Let $R(s,d)$ be a time-dependent path. Let $T \in \mathbf{I}^+$: $0 \leq T < \infty$ be some time instance. Let $\mathcal{I}^*(s,d) = \langle s \equiv v(t_0), v(t_1), ..., v(t_{k-1}), ..., v(t_{N-1}) \equiv d\rangle$ be a minimal weight finite T-termination itinerary of length at most N among all T-termination itineraries of length at most N. Then $\forall k,\ 0 \leq k \leq L(\mathcal{I}^*)$, itinerary $\mathcal{I}_k \stackrel{\text{def}}{=} \langle s \equiv v(t_0), v(t_1), ..., v(t_{k-1})\rangle$ is a minimal weight finite T-termination itinerary of length at most k for destination $v(t_{k-1}) \in R$.

Proof: See [2]. ∎

An important observation is that according to our model the length of any itinerary $\mathcal{I}(s,d)$ viable for some finite deadline T being an integer number of time units, is upper-bounded by $\hat{l} \stackrel{\text{def}}{=} \lceil \frac{T}{\min_D\{d_{i,j}\}} \rceil$.

In practice we are interested in viable optimal itinerary only with respect to some finite deadline T being an integer number of time units. Thus we can find optimal T-termination, $(T-1)$-termination, $(T-2)$-termination *etc.*, itineraries with length at most $\hat{l}$ using the concatenation property stated by Lemma IV.1, and choose a minimal-weight viable simple itinerary among them.

Indeed, the optimal time-dependent path concatenation property allows recursive constructing of the optimal t-termination itineraries for every $t \in \mathbf{I}^+ : 0 \leq t \leq T$ and for every length $l \in \mathbf{I}^+ :\ HD(s,d) \leq l \leq \hat{l}$.

Suppose, for a given path $R(s,k)$, $k \in R(s,d)$, $\mathcal{I}(s,k)$ is an optimal t-termination itinerary of length at most $\hat{l}$ where $0 \leq t \leq T$. Let us denote its weight by $W_k(t)$. Then we have the following recurrent relations:

$$W_k(t) = \min_{i \in \{k-1,k,k+1 \in R\}} (W_i(t-d_{i,k}) + w_{i,k}(t-d_{i,k})) \quad (1)$$

$$W_s(0) = 0 \quad (2)$$

$$\forall t:\ 0 < t \leq T, \forall i \in R : W_i(t) = \infty. \quad (3)$$

From these recurrent relations we can compute the weights for optimal t-termination itineraries for every node in R and also the minimal viable itinerary for reaching the destination before deadline T as shown in [2].

B. On-Line Problem (Miser Algorithm)

In this section we present an algorithm that copes with the on-line k-recoverable TMP in a restricted case of parking weight densities being constant throughout allowed delay specified for a packet. Then we show that for the special case of $k = 1$ the presented algorithm is $3 + 2\log_2 B$ competitive where B is the maximum block time.

For $k > 1$ the algorithm serves as a heuristic whose performance is demonstrated in Section V by means of simulation. B is assumed to be large enough to admit a solution.

The idea of the on-line algorithm for the k-block recoverable problem is as follows. The miser is advancing towards the destination using the non-fictitious links only (*i.e.*, using only the links belonging to the shortest itinerary) as long as he does not reach a block. At a block the miser has two options: either to stay at the node where the block has occurred (the stay is simulated by traversing the corresponding self-link), or move backwards. The adversary, attempting to increase the cost of the trip, is locking the miser at the most expensive node on the way towards the destination. Assuming that the miser knows an upper bound U for the block duration, a competitive strategy for the miser would be to pick up the cheapest node (in terms of the total cost of getting there and back plus spending a certain amount of time at the node as explained below) situated within roughly $U/2$ distance from its current location on the way back to the source, to spend the block time there. In other words, the miser has to pick up an itinerary with the minimal weight for the block time duration. The intuition behind the algorithm is that if the miser has to spend some time en route on the way to the destination due to a block, he better do this using the cheapest itinerary. This way the overall cost of the trip is minimized.

However, the miser does not know the exact duration of any block, only an upper bound, that is not necessarily tight. Thus, it is beneficial to have a strategy to find a tighter bound.

Definition 15: **Simple Round Trip Itinerary:**
Itinerary $\mathcal{I}_{rt}(j,k) \stackrel{\text{def}}{=} \langle j, ..., k, \underbrace{k, k, ..., k}_{0 \leq l}, ..., j\rangle$ such that $\forall i \neq k$ link $(i,i) \notin \mathcal{I}_{rt}(j,k)$ is termed ***simple round trip itinerary*** between two nodes $j, k \in R$.

In other words, a simple round trip itinerary between the two nodes $j, k \in R$ is an itinerary that is a *concatenation* of a shortest, possibly empty, itinerary that goes from node j to node k, possibly empty finite itinerary that uses only the (k,k) fictitious links, and the shortest, possibly empty, itinerary that goes from node k to node j.

Suppose a block occurs at some node $v_b \in R$ on the link leading towards the destination. The algorithm works in stages. At stage $j \in [1, \log_2 U]$ the miser chooses node $v_j^* \in R$ reachable from v_b within $T \leq 2^{j-1}$ time units, and from which v_b is reachable within $T' \leq 2^{j-1}$ time units, such that

$$W(\mathcal{I}_{rt_j}(v_b, v_j^*)) =$$

$$\inf_{v_j^*: HD(s,v_j^*) \leq HD(s,v_b)} W(\langle v_b, ..., \underbrace{v_j^*, ..., v_j^*}_{2^j - TD(v_j^*, v_b)}, v_j^*, ..., v_b \rangle)$$

.

Then the miser follows this minimal weight simple round trip itinerary for stage j.

In order to gain a logarithmic factor, the miser spends exponentially increasing periods of time away from the blocked node. After each such period the miser goes back to the blocked link and checks whether the block is lifted. If the block is lifted then the miser goes through, otherwise he again chooses the cheapest node, moves there, and spends there twice the time he spent in the previous step.

1) Analysis: Obviously, in the above algorithm, we call it *miser algorithm*, the miser will not go back and forth more than $\log_2 B$ times and cannot spend more than $2 * B$ time units at the cheapest node (the cheapest node may vary from one round trip to another) under any circumstances, where B being the actual duration of the block. Thus, the algorithm terminates after at most $log_2 B + 1$ simple round trips and the itinerary it implicitly builds, is finite.

It is easy to see that in case of multiple blocks, *i.e.*, when $k > 1$, the strategy is not competitive. If two blocks occur simultaneously on the two real links leading from the node (one in the forward direction for time B_1, and one in the opposite direction for time B_2), the weight of the resulting itinerary will increase by adding $w_{v_b,v_b} * B$ where $B = \min\{B_1, B_2\}$. In contrast, the optimal off-line algorithm would build its itinerary in such a way that it will spend all the blocking time in the cheapest place. Therefore an adversary can always force the ratio between the on-line and the off-line algorithms to be the ratio between the minimal and the maximal link weights of the path. Since blocks do not always follow the above worst case scenario in practice, the presented on-line strategy can be used as a heuristic. Its performance is studied using simulations in Section V under various assumptions on the blocks distribution.

However, in case of $k = 1$, the above on-line algorithm is logarithmically *competitive*. The following theorem states this formally.

Theorem 1: Let $R(s, d)$ be some time-dependent path. Let $\mathcal{D}$ be a finite deadline, and let U be the upper bound on the blocking time. Let $\mathcal{D}$ and U be such that exists at least one simple itinerary $\mathcal{I}(s, d)$ viable for $\mathcal{D}$. Let $d_{i,j} = d_{j,i}\ \forall i, j \in R$, and $TD(i, j) \leq \lceil U/2 \rceil$. If there can be only one block of duration $B \in \mathbf{I}^+ :\ 0 \leq B \leq U < \infty$, then the competitive ratio of algorithm RT is $2 \log_2 B + 3$.

Proof: As can be easily verified, the itinerary built by algorithm OPT in case of one block that happens at node $v_b \in R$ for time $B = t^{end} - t^{start}$, $0 \leq t^{start} \leq t^{end} \leq \mathcal{D} < \infty$, and t^{start} such that $TD(s, v_{b+1}) > t^{start}$ is

$$\mathcal{I}_{opt} \stackrel{\text{def}}{=} \langle s, v_1 ..., v^*, v^*, ..., v^*, ..., v_b, v_{b+1}, ..., s \rangle =$$

$$\hat{\mathcal{I}}(s, v^*)\ \&\ \tilde{\mathcal{I}}(v^*, v^*)^{B - TD(v^*, v_b)}\ \&\ \hat{\mathcal{I}}(v^*, v_b)\ \&\ \mathcal{I}(v_b, d).$$

In other words, according to algorithm OPT, the miser advances to the node with the minimal parking weight density reachable before the block begins, v^*:

$$w_{v^*,v^*}(t) = \inf_{\forall v \neq v_b:\ TD(s,v) \leq t^{start};\ 0 \leq t \leq t^{start}} w_{v,v}(t),$$

using the shortest itinerary, spends $B - TD(v^*, v_b)$ at this node by traversing only fictitious links (v^*, v^*) and then advances to the destination via node v_b that is guaranteed to be unblocked by the earliest possible arrival time to v_b. Thus for the optimal algorithm we have:

$$W(\mathcal{I}_{opt}) \stackrel{\text{def}}{=} (B - TD(v^*, v_b)) * w_{v^*,v^*} + W(\hat{\mathcal{I}}(s, d)) \quad (4)$$

Similarly, for the on-line algorithm we have:

$$W(\mathcal{I}_{rt}) \stackrel{\text{def}}{=} \sum_{j=1}^{\log_2 B + 1} [(2^j - 2TD(v_j^*, v_b)) * w_{v_j^*,v_j^*} + \quad (5)$$

$$2W(\hat{\mathcal{I}}(v_b, v_j^*))] + W(\hat{\mathcal{I}}(s, d))$$

From 4 we have that

$$W(\hat{\mathcal{I}}(s, d)) \leq W(\mathcal{I}_{opt}). \quad (6)$$

Hence,

$$W(\mathcal{I}_{rt}) \leq \sum_{j=1}^{\log_2 B + 1} [(2^j - 2TD(v_j^*, v_b)) * w_{v_j^*,v_j^*} +$$

$$2W(\hat{\mathcal{I}}(v_b, v_j^*))] + W(\mathcal{I}_{opt}).$$

Since

$$\sum_{j=1}^{\log_2 B + 1} [(2^j - 2TD(v_j^*, v_b)) * w_{v_j^*,v_j^*} + 2W(\hat{\mathcal{I}}(v_b, v_j^*))] \leq$$

$$\sum_{j=1}^{\log_2 B + 1} [(2B - 2TD(v_j^*, v_b)) * w_{v_j^*,v_j^*} + 2W(\hat{\mathcal{I}}(v_b, v_j^*))],$$

we have that

$$W(\mathcal{I}_{rt}) \leq \sum_{j=1}^{\log_2 B + 1} [(2B - 2TD(v_j^*, v_b)) * w_{v_j^*,v_j^*} +$$

$$2 * W(\hat{\mathcal{I}}(v_b, v_j^*))] + W(\mathcal{I}_{opt}).$$

Since under assumption of the theorem $\forall i, j \in R\ \ TD(i, j) \leq \lceil U/2 \rceil$, and link weights are strictly positive, the following inequality holds for any node v_j^* chosen by RT algorithm at stage $j \in [1,\ \log_2 B + 1]$, and for node v^* chosen by algorithm OPT:

$$2B * w_{v_j^*,v_j^*} - 2TD(v_j^*, v_b) * w_{v_j^*,v_j^*} + 2W(\hat{\mathcal{I}}(v_b, v_j^*)) \leq$$

$$2B * w_{v^*,v^*} - 2TD(v^*, v_b) * w_{v^*,v^*} + 2W(\hat{\mathcal{I}}(v_b, v^*)) \leq$$

$$2W(\mathcal{I}_{opt}).$$

But then we observe that

$$W(\mathcal{I}_{rt}) \leq \sum_{j=1}^{\log_2 B\ +1} [(2B - 2TD(v_j^*, v_b)) * w_{v_j^*, v_j^*} +$$

$$2{*}W(\hat{\mathcal{I}}(v_b, v_j^*))] + W(\mathcal{I}_{opt}) \leq 2\log_2 B{*}W(\mathcal{I}_{opt}) + 3W(\mathcal{I}_{opt}).$$

Therefore:

$$W(\mathcal{I}_{rt}) \leq (2\log_2 B + 3) * W(\mathcal{I}_{opt})$$

.

■

V. SIMULATIONS

In this section, we present a simulation study of the on-line miser algorithm proposed in IV-B. We refer to it simply as *Miser* for brevity. Since competitive analysis provides indication only about the worst case, it is beneficial to study the actual performance of the algorithm through simulation comparing it to other alternatives. We compare it against the simple active algorithm explained in Section I (see Figure 1), referring to the latter as *Simple*.

The overhead imposed on active nodes is quantified by the itinerary weight, *i.e.*, by the cost of the respective itineraries that these two algorithms build (see Definition 6). In addition, we are interested in the propagation time of a packet through the network.

We start our simulation study by comparing the itinerary weights for Simple and Miser algorithms when only one block may occur along the path during a specified deadline (*i.e.*, for 1-block recoverable TMP).

Figure 3 represents the simulation results as the itinerary weight (cost) being a function of the path length for two different cost models *pyramid*, and *uniform* whose meaning is as follows.

- ***Pyramid***: this cost model represents a deterministic distribution of parking densities to the nodes in the path simulating "cheap" edges and "expensive core". The parking weight density increases exponentially with the distance of a node from the edge, *e.g.*, $(1, 2, 4, 8, 16, 8, 4, 2, 1)$.
- ***Uniform:*** this cost model represents a random distribution of the parking weight densities to the nodes. Each cost is computed as 2^i, where i being an integer value $i \sim u(0, 6)$.

In both cases the costs are exponential. This is done in order to obtain a visible contrast between the itinerary weights of the studied algorithms.

For every cost model the graphs are obtained by averaging 500 runs of every algorithm for every path length ranging from 5 nodes to 20 nodes. In all the experiments the deadline was chosen to be $\mathcal{D} = 10000$ time units.

Every run for a given path length and a cost model is performed as following. At the beginning of each run we regenerate a path by allocating the parking weight densities to its nodes. Then, we randomly choose a link that will be congested. After this we draw a random congestion duration b from $U(10, \mathcal{D}/2)$ distribution, and initiate a block at time 0 that will be removed after b time units. When this scenario is formed we execute both algorithm on the same scenario recording its performance: itinerary weight (cost), and total transit time.

Figure 3 shows that Miser is always superior over Simple for both cost models. The difference is larger for the *pyramid* cost model. The explanation is that since the difference between the core and and the edges is very sharp, therefore when the Miser is caught by a long block, after a few iterations it will migrate far towards the source, dramatically decreasing its costs while the simple algorithm remain parked in a more expensive location. In case of the *uniform* model this contrast gap in the parking costs is alleviated.

As one would expect, the variability of the itinerary weights obtained in different runs is high, because in each run the weight depends on the random block location and duration. Therefore, although the average provides a convenient indication, it would be far more informative to characterize the results by means of *empirical cumulative distribution function (ECDF)*. Figure 4 shows the ECDFs for itinerary weights when the parking cost model is *uniform* and *pyramid*, respectively. As one can clearly see, in both cases Simple exhibits much higher probability of a very large itinerary weight. This means that it would impose much higher overhead on the active network if accidentally caught by the block at an expensive (*i.e.*, loaded) active node. Miser, in contrast, would smooth these outlying cases. Also, as one would indeed expect, the cost of Miser is logarithmically increasing with path length in case of the *pyramid* cost model since the core becomes more expensive exponentially in this case.

Similar information is presented in Figure 5 for the total transit time of the packets when the cost model was *pyramid*. As one can see, in general, Simple spends less time in the network, while Miser pays with time for the cost optimization it achieves. This happens because of over-estimation of a block upper bound that would occur in some cases.

Finally, it is beneficial to measure the dependency of itinerary weight on the block duration. Figure 6 depicts the cost as function of deterministically chosen block durations and *pyramid* cost model. As expected, the cost of Miser increases logarithmically, while the cost of Simple increases linearly. As one can clearly see, for the longer congestion periods, Miser is definitely superior. Another interesting behavior is the exponential height and width of the steps in the performance of Miser. This is a clear depiction of the 'nature' of the algorithm operation, that doubles its waiting time if the block is still present.

We want to attract the reader's attention to the fact that the encouraging results have been obtained for relatively short paths which corresponds to real topologies. The deadline assumed was also realistic given that the round trip time for such topologies is order of hundreds of milliseconds while the low-priority traffic deadlines is at minimum order of minutes. Finally, it is reasonable to assume that during a few minutes, indeed only one congestion would occur in the network, and the load on active nodes would not change dramatically, so that fixed parking weights are also reasonable. All this provides a strong motivation for the actual implementation of the proposed algorithm, expecting a real benefit from it.

The next set of simulations compares Miser and Simple in a more general case when the number of blocks was unlimited.

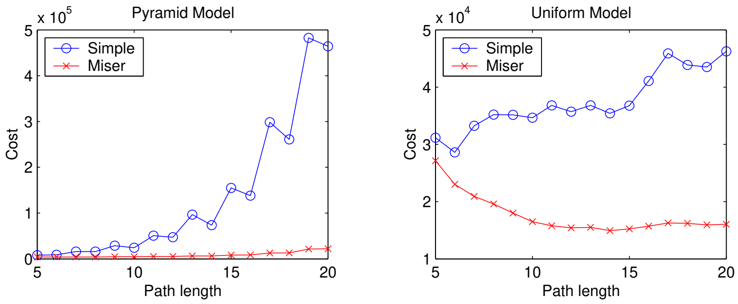

Fig. 3. Average Itinerary Weight

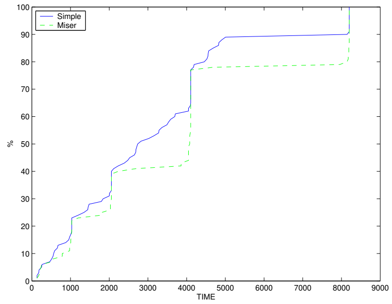

Fig. 5. ECDF of Itinerary Total Time (Pyramid Cost Model)

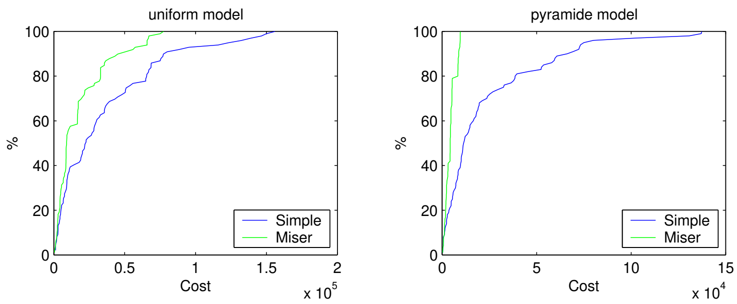

Fig. 4. ECDF of Itinerary Weight (Pyramid and Uniform Cost Models)

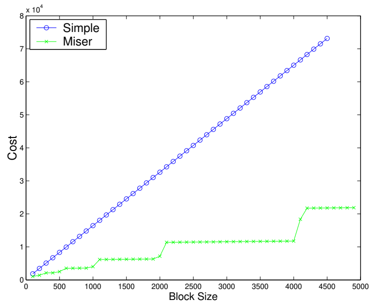

Fig. 6. Itinerary Weight vs Block Duration (Pyramid Cost Model)

As before we used the two cost models. In order to create multiple transient congestions we produced a random number of *communicating endpoints* generating a *cross traffic* through the nodes. For every pair, we randomly chosen an entry point and an exit point, and associated an identical fixed data transmission rate. At any given moment of time, two communicating endpoints can be either in an *ON* state where they transmit packets at a fixed rate, or in an *OFF* state where they are silent. The *ON* and *OFF* periods are drawn from the exponential distribution with λ_1 and λ_2 means respectively. The load on a link is simply given by summing the transmission rates of the cross traffic through it. Finally, we configure a threshold value for the links, so that when the load crosses over this value the link becomes blocked.

We have made the following calculation of the threshold value for every given path length. Assuming that the cross connections are distributed evenly along the path, the expected number of cross connection at the center of the path is $1/4$ of the existing connections, and therefore if we have n cross connection, with load one each, the expected load on the middle link is $\frac{\lambda_1 \cdot n}{4(\lambda_1+\lambda_2)}$. In order to get a reasonable number of link blocks due to congestion we thus need to set up the blocking threshold to be a fraction of this number.

Figure 7 shows an ECDF of 500 runs for both algorithms with the *uniform* cost model for different threshold values. The threshold values have been calculated as discussed above assuming that $\lambda_1 = \lambda_2 = 0.5$, and the number of cross connections is $n = l^2$, where l is the path length. As one can see, Miser is slightly superior in case of multiple blocks. The main problem we had with this set of simulations was a difficulty to generate real transient blocks synthetically. Therefore the marginal cases are prevalent meaning that the concession is either too short, or too long.

VI. Related Work

The problems studied in this paper have much in common with the well know problem of finding a minimal weight path. Finding a minimum weight path in a static network has long been the subject for extensive research [3]. The same problem for networks with time-dependent edge lengths has been studied by [4], [5], [6], [7], and most extensively by [8]. Polynomial time algorithms for computing the minimal delay path in such networks when delay functions are known, have been demonstrated.

As was correctly pointed out by [2], if real (even if strictly positive) time-dependent link weight functions are associated with traversing the links and parking in the nodes, the problem of finding a minimal weight path is, generally speaking, different from that of finding a path with minimal delay. The resulting paths are not necessarily simple and not necessarily minimal delay paths since in general delays are independent from the weights.

The well-known optimality principle [9], stating that a subpath of any optimal path is itself an optimal path widely used

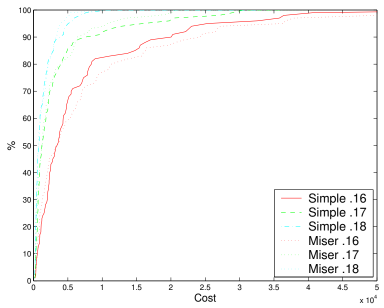

Fig. 7. ECDF of Itinerary Weight for Multiple Blocks (Uniform Cost Model)

for finding a minimal weight path in a static network, is not valid for time-dependent networks. This apparently prohibits an efficient solution. Different models have been presented to circumvent this problem: [10], [11].

In [2] a most general model has been presented, and similar, but more restricted optimality principle has been established. We have discussed this principle in Section IV-A along with the algorithm that is based on it. In [2] an algorithm for finding a minimal weight path in time-dependent network when the weight functions are known in advance, has been presented. Thus, this work, essentially solves our *off-line* problem.

In practice, however, the weight functions may be not known in advance. This is the *on-line* variant of the problem in which the values of link weight and delay functions become known only upon an arrival to a vertices or a link, and their future values are unknown.

Very few studies of the on-line variant of the problem exist. One special case is known as the *Canadian Traveler Problem* [12], [13]. In this problem a certain finite number of link failures (modeled by infinite link weights) may occur with failed links either remaining faulty forever, or being recovered after a finite period of time, recovery time. A strictly positive constant delay is associated with every link when it is not failed. In [12] a polynomial time strategy minimizing the total travel time from one vertices to another in such time-dependent network was presented. The crucial assumption made in [12] was letting the link down time periods be smaller than the internodal delays. This allowed an elegant recursive solution. Unfortunately, this assumption renders the solution impractical for most networking environments where the situation is just the opposite: an internodal delay of a non-faulty link is smaller than the typical down time period.

In our study we, in fact, revisit the Canadian Traveler Problem for the special case of the network being a single path, and we do not make any assumptions on the ratio between the internodal delay and the down time.

In [7] a distributed protocol for minimal delay routing in time-dependent networks oriented on the traditional routing model in which routers exchange control information about delays of the locally known links, is presented. Deployment of such protocol requires add-ons to the existing routing software, and, possibly, hardware.

Recently a new paradigm known as *active networks* has emerged [14]. This paradigm allows seamless deployment of routing protocols at the level of individual *active packets*. Active networks turn out especially useful for wide range of network management applications [1], [15], [16]. In this work we maintain that this paradigm may be successfully deployed for transferring large volumes of management traffic while minimizing its impact on the user applications that consume the same network resources.

VII. Conclusion and Future Work

We presented a novel framework for handling low priority traffic with non-stringent timing constraints resulting from various management applications. We demonstrated that the active networks paradigm may help in significantly reducing the competition for network resources between this low priority traffic and the user application traffic at times of congestion thanks to the intermediate parking capability provided by active networks. Intermediate parking enables to increase the network's goodput, and, as a side effect, makes the management traffic more robust in presence of network congestion.

We proposed an efficient scheduling algorithm that minimizes the overhead imposed on the active network infrastructure itself by the intermediate parking. We shown that the desired effect may be achieved without the routing path modifications by constructing efficient parking schedules along a fixed routing path. In restricted, but practical, case the proposed algorithm is logarithmically competitive. In less restricted cases it serves as a useful heuristic, as demonstrated by simulations. Interestingly enough, the encouraging results have been obtained for relatively short paths which corresponds to the real networks' dimensions.

Based on this study, we believe that the active network paradigm may be of practical usefulness in reducing the impact of large amount of management traffic on the user applications that consume the same network resources. We are planning to implement the presented algorithm using our active network testbed [1].

References

[1] Danny Raz and Yuval Shavitt, "Active networks for efficient distributed network management," *IEEE Communications Magazine*, vol. 38, no. 3, Mar. 2000.

[2] Ariel Orda and Rafael Rom, "Minimum weight paths in time-dependent networks," *Networks*, vol. 21, pp. 295 – 319, May 1991.

[3] R. Bellman, "On a routing problem," *Quart. Appl. Math.*, vol. 16, pp. 87 – 90, 1958.

[4] Emil Klafszky, "Determination of shortest path in a network with time-dependent edge-lengths," *Math. Operationsforsch. u. Statist.*, vol. 3, no. 4, pp. 255 – 257, 1972.

[5] Kenneth L. Cooke and Eric Halsey, "The shortest route through a network with time-dependent internodal transit times," *Journal of Mathematical Analysis and Applications*, vol. 12, pp. 493 – 498, 1966.

[6] J. Halpern, "Shortest route with time dependent length of edges and limited delay possibilities in nodes," *Zeitschrift fuer operations research*, vol. 21, pp. 117 – 124, 1977.

[7] Ariel Orda and Rafael Rom, "Distributed shortest path and minimum-delay protocols in networks with time-dependent edge-length," *Distributed Computing*, vol. 10, pp. 49 – 62, 1996.

[8] Ariel Orda and Rafael Rom, "Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length," *Journal of the ACM*, vol. 37, pp. 607 – 625, July 1990.

[9] R. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, New Jersey, 1957.

[10] Kiseok Sung, Michael G.H. Bell, Myeongki Seong, and Soondal Park, "Shortest paths in a network with time-dependent flow speeds," *European Journal of Operational Research*, vol. 121, pp. 32 – 39, 2000.

[11] Irina Ioachim, Sylvie Gelinas, Francois Soumis, and Jacques Desrosiers, "A dynamic prograamming algorithm for the shortest path problem with time windows and linear node costs," *Networks*, vol. 31, pp. 193 – 204, 1998, John Wiley & Sons, Inc.

[12] Amotz Bar-Noy and Baruch Schieber, "The canadian traveller problem," in *Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, San-Francisco, California, 1991, pp. 261 – 270.

[13] C.H. Papadimitriou and M. Yannakakis, "Shortest paths without a map," in *16th ICALP*, July 1989.

[14] Konstantinos Psounis, "Active networks: Applications, security, safety, and architectures," *IEEE Communications Surveys*, vol. 2, no. 1, First Quarter 1999, `http://www.comsoc.org/pubs/surveys/1q99issue/psounis.html`.

[15] Ryutaro Kawamura and Rolf Stadler, "Active distributed management for ip networks," *IEEE Communications Magazine*, vol. 38, no. 4, pp. 114–120, Apr. 2000.

[16] B. Schwartz, A. Jackson, T. Strayer, W. Zhou, R. Rockwell, and C. Partridge, "Smart packets for active networks," in *IEEE OPENARCH'99*, New York, NY, USA, Mar. 1999, pp. 90–97.