Topology Dynamics and Routing for Predictable Mobile Networks

Daniel Fischer
University of Luxembourg
6, R. Coudenhove-Kalergi
Luxembourg, Luxembourg
Email: daniel.fischer@uni.lu

David Basin
Information Security Group
ETH Zurich
Zurich, Switzerland
Email: basin@ethz.ch

Thomas Engel
University of Luxembourg
6, R. Coudenhove-Kalergi
Luxembourg, Luxembourg
Email: thomas.engel@uni.lu

***Abstract*—Predictable mobile networks are dynamic in terms of the mobility and connectivity of the network nodes. However, in contrast to mobile ad-hoc networks, their dynamics are mostly predictable, as the name suggests. Currently, no adequate topology model exists for such networks. Moreover, routing protocols based on either static or mobile ad-hoc topology models do not exploit this predictability and thus are too inefficient for use in some application areas.**

We present a model that formalizes predictable dynamic topologies as sequences of static snapshots. We use this model to design and prove the correctness of a protocol based on link-state routing, whose performance is superior to its static and ad-hoc counterparts. Our routing protocol accounts for occurrences of additional, unpredictable changes, as well as their interaction with predictable changes. As an application area, we focus on routing in spacecraft networks and explain the suitability of our protocol for this application domain.

I. Introduction

Problem Context and Motivation: Spacecraft networks and inter-spacecraft communication are topics of growing importance within space agencies and industry. Emerging application areas require an efficient, integrated, and interconnected communication environment in space. The current spacecraft infrastructures, with their isolated, specialized operational networks, fail to meet these requirements. Next-generation spacecrafts are expected to establish inter-spacecraft links and to create multi-purpose space networks [1].

Future spacecraft networks will have characteristics different than existing terrestrial networks. Spacecraft networks are mobile networks with independent, heterogeneous nodes but without spontaneous movement. In regular operation, each spacecraft has a predictable trajectory and its position and communication links are therefore predictable as well. Hence, spacecraft networks constitute a subclass of mobile ad-hoc networks that is characterized by its predictability. As in other networks, unpredictable changes may still occur and influence the evolution of the topology.

Predictable movements are absent in both mobile ad-hoc networks [2] and static network topologies. Therefore, models developed for either of these scenarios are inadequate to describe predictable mobile network topologies. A topology model that accounts for predictable movement and links is required as a basis for developing advanced networking concepts. The situation is similar, when one considers routing protocols that have been specifically developed for spacecraft networks. Although many solutions have been proposed for specialized constellations [3], [4], [5], [6], [7], no general routing protocol exists for predictable mobile networks. Such a protocol, which copes with both scheduled and unscheduled topology changes, is required to support a heterogeneous interconnected and integrated spacecraft-network environment.

Approach and Contribution: We present a formal topology representation of predictable mobile networks described by a sequence of static network topology snapshots. The starting point for our work is [3], which proposes to abstract mobile networks as a series of snapshots. We take this idea, base a formal model on it, and use this model to develop a specialized routing protocol. Snapshots describe the network connectivity at particular time intervals and snapshot sequences formalize how the network connectivity changes over time. In this way, we incorporate predictability while abstracting away from details such as trajectory and flight dynamics for individual orbits and flight paths. Following the formalization of our topology model, we use it to develop a general purpose Predictable Link-State Routing (PLSR) protocol that accounts for both predictable and unpredictable changes and their interaction. We show the correctness of our protocol and measure its performance using simulation.

Organization: The remainder of this paper is organized as follows. In Section 2, we provide background information on spacecraft networks. In Section 3, we introduce and formalize the snapshot topology model. In Section 4, we address the management and distribution of the topology information and, in Section 5, we present our PLSR protocol and prove its correctness. In Section 6, we present performance evaluation results. Finally, we review related work in Section 7 and we draw conclusions in Section 8.

II. Background on Spacecraft Networks

Satellite communication is traditionally of two kinds: either (1) direct point-to-point links from control centers to spacecrafts or (2) bent-pipe communication applications, where spacecrafts relay a data stream to a large user community. Both approaches do not make use of general networking technologies. Earlier attempts to use more sophisticated spacecraft communication models in the form of Low Earth Orbit (LEO) constellations with inter-spacecraft links [8], [9] had limited commercial success due to their inability to integrate the spacecraft network with other terrestrial networks [1].

The new application areas require integrated multi-purpose spacecraft networks and constellations consisting of spacecrafts of different sizes, types, and flight dynamics. These spacecrafts may also be in contact with different terrestrial end-user terminals or networks. Hence, a topology model for space networks must be able to describe a large variety of differently behaving nodes and links with the common property of predictability.

Spacecraft networks are controlled and managed by a control center that is connected to the network via one or more ground stations. We will restrict our attention to the case of a single ground station in this paper. This is without loss of generality as multiple ground stations can always synchronize their actions over a high-speed terrestrial network. Radio frequency modulation and optical communication are the most common technology used for space-link communication. Note that the physical communication medium, which is used for spacecraft communication, can have a direct impact on the efficiency of the routing protocol.

III. Topology Model

In this section, we present a formal topology model for predictable mobile networks that also handles unpredictable topology changes. While our work is motivated by spacecraft networks, our model is general and covers all kinds of predictable mobile networks.

A. Topology Properties and Approach

Designing and reasoning about algorithms for spacecraft networks requires a topology model that describes networks with the following properties.

- *Mobility and routing:* Each node in the network is mobile and provides routing functionalities.
- *Predictability:* The network topology changes in a predictable way at given points in time by adding or removing nodes or links.
- *Unpredictability:* Unpredictable changes may occur at any time in the network. They are caused by unforeseen events such as node failures, but not by node movement.

We first formalize a topology model that handles predictable changes based on a notion of snapshots. A snapshot describes the (static) state of the network topology during a time interval and a snapshot sequence describes the topology's expected evolution over time. In actual applications, however, predictable mobile networks are also subject to unpredictable changes. In our approach, predictable changes are formalized by a snapshot transition function, whereas unpredictable changes must be discovered and corrected for, essentially by superimposing them upon the snapshot sequences.

B. Predictable Topology Changes

We specify the behavior of the network topology in the presence of predictable changes and introduce a model that characterizes predictability in mobile networks. The concept of snapshots was informally introduced in [3], where the authors use snapshots to describe the mobility of a Low Earth Orbit (LEO) spacecraft network. They state that a snapshot represents the topology of the spacecraft network at a particular instance of time. In the following, we formalize the snapshot concept, associating it with a time interval rather than a time instance.

Definition 1: Time Intervals and Transitions.

- A *time set* is a triple $(T, \leq, t_0)$. The elements of the set T are called *time points*, $\leq$ is a total order on T, and t_0 is the smallest element of T under $\leq$. We will often leave $\leq$ and t_0 implicit and simply speak about the time set T. For $t', t'' \in T$ we will use the notation $[t', t''[$, for $t' < t''$, to represent the *time interval* $\{t \in T \mid t' \leq t < t''\}$.
- Let I be a countably infinite subset of T. We call I a *set of transition points* if for all $t \in T$ there are $t', t'' \in I$ with $t \in [t', t''[$. From this definition, it follows that $t_0 \in I$. We will enumerate the elements of I as $t_0, t_1, \ldots$, where for all $i \in \mathbb{N}$, $t_i < t_{i+1}$.

We can now describe the topology during a given time interval $[t_i, t_{i+1}[$.

Definition 2: Connectivity Graph and Snapshots. Let $t_i, t_{i+1} \in I$.

- The predicted network topology during the interval $[t_i, t_{i+1}[$ is described by a directed *connectivity graph* $G_i = (V_i, E_i)$, where V_i is the set of network nodes and E_i is the set of active communication links.
- For $t_i \in I$, a *snapshot* $S_i = \langle t_i, G_i \rangle$ represents the predicted network connectivity during the time interval $[t_i, t_{i+1}[$.
- The predicted evolution of the topology over time is described as a *sequence of snapshots* $S = \langle S_0, \ldots \rangle$. A transition point t_i, for $i > 0$, represents the time point when the network connectivity changes from S_{i-1} to S_i.

A sequence of snapshots describing predictable changes can be defined by an initial snapshot S_0 and a transition function δ that maps snapshots to snapshots. The transition function describes the way that the network connectivity is expected to change over time. In particular, the snapshot sequence defined is $\langle S_0, S_1, S_2, \ldots \rangle$, where $\delta(S_i) = S_{i+1}$, for $i \in \mathbb{N}$.

Note that for spacecraft networks, defining δ requires sophisticated calculations using the orbital mechanics and flight dynamics of the involved spacecrafts. We abstract away from these considerations here and focus on the nature of such snapshot transitions.

C. Predictable and Unpredictable Changes

Although predictable changes form the vast majority of the events that determine the topology dynamics in a spacecraft network, unpredictable changes can also occur at any time due to incidents that cannot be anticipated, such as node failures.

Unpredictable changes are not covered by the transition function and must be discovered by the nodes using link-sensing. This means that nodes must check their neighborhood

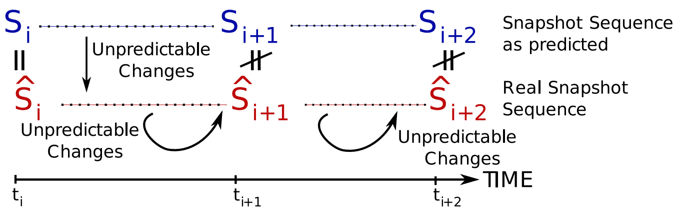

Fig. 1. Deviation between reality and the predicted snapshot sequence caused by unpredictable changes

for unpredictable changes either proactively, at regular time intervals, or reactively. Unpredictable changes transform the reality that is represented by a given snapshot. Let S_i be the current snapshot and let u be an unpredictable change that occurs within S_i and describes the addition or removal of an edge in the topology graph G_i. Following the occurrence of u, the snapshot S_i no longer correctly represents reality. The reality corresponds instead to a modified snapshot $\hat{S}_i$, whose topology graph accounts for u. Said another way, the reality can be abstracted to a snapshot $\hat{S}_i$ that differs from S_i only in the presence of the unpredicted change. Figure 1 illustrates this and the fact that the deviation accumulates with each additional unpredictable change.

This raises two problems that need to be resolved when designing protocols for predictable mobile networks. The first problem is related to the interaction of predictable and unpredictable changes at transition points. As the predictable snapshot sequence does not include unpredictable changes, such changes may be overridden by the snapshot topology images. The second problem is the need for regular realignment of the original predicted snapshot sequence $\langle S_0, S_1, S_2, ...\rangle$ with the real sequence $\langle \hat{S}_0, \hat{S}_1, \hat{S}_2, ...\rangle$ to incorporate modifications due to past unpredictable changes. In this paper, we provide solutions to both problems. Note that these problems (and opportunities) do not arise in a mobile ad-hoc setting with no a-priori topology knowledge of node movement and communication opportunities since all changes are unpredictable.

IV. Topology Information Management

In predictable mobile networks, routing protocols can only benefit from the presence of predictability if the nodes that perform routing have access to the topology information for upcoming snapshots. In this section, we elaborate on the distribution and management of this topology information. Distribution is required because it is generally infeasible to store all topology management information in the nodes at the time of network initialization (snapshot S_0).

We present a topology information management algorithm that supports the link-state based routing scheme that we introduce in Section V-A.

A. Distribution of Snapshots

Spacecrafts have adequate memory for storing snapshot sequences of reasonable length [10]. Each node stores a sequence $\langle S_j$,..,$S_{j+k}\rangle$ of $k+1$ snapshots, with $k \geq 0$, and requires an update with a new sequence $\langle S_{j+k+1}, ...\ \rangle$ before time t_{j+k+1}. This update process must also modify snapshot updates to account for unpredictable changes that have affected previous snapshots.

A special property of spacecraft networks is the presence of a ground station node that has extensive computational resources and can pre-compute sets of snapshots. Furthermore, by participating in the network, the ground station also receives information on unpredictable changes and can incorporate them in future snapshots. To facilitate this, all unpredictable changes are time-stamped. This allows the nodes to distinguish between those unpredictable changes that have been used as an input for the snapshot computation and those that occur between snapshot computation and the time point at which the computed snapshot becomes valid (i.e. time point t_i for a snapshot S_i). Information on unpredictable changes that reach the ground station is stored there in a data structure UC, which stores the changes together with their time stamps.

Providing an initial pre-computed snapshot sequence to all network nodes eliminates the need for an initialization phase in the routing protocol. Nodes that are added to the network by a predictable change can also be equipped with an initial sequence of snapshots.

We will use the following definitions in our formalization of snapshot distribution and later for our routing algorithm.

Definition 3: Restricted Graphs and Neighborhood. Let $G = (V, E)$ be a graph and $V' \subseteq V$ a subset of nodes.

- Let $E\,|_{V'} = E \cap (V' \times V')$ denote the *restriction of* E *to* V' and let $G\,|_{V'} = (V', E\,|_{V'})$ denote the *restriction of* G *to* V'.
- We define the *edges neighboring* v *in the graph* $G = (V, E)$ as $\{(x, y) \in E \mid x = v \vee y = v\}$. Similarly, the *neighbors of* v *in the graph* $G = (V, E)$ are $\{w \in V \mid (v, w) \in E \vee (w, v) \in E\}$.

Definition 4: Transition-affected subgraph. Let $t_i \in I$ be a transition point and S_i the corresponding snapshot. We define:

- $VT_i = \{v \in V_i \mid \exists (x, y) \in E_i \setminus E_{i+1} : (x = v) \vee (v = y)\} \cup \{v \in V_i \mid \exists (x, y) \in E_{i+1} \setminus E_i : (x = v) \vee (y = v)\}$. This is the set of nodes v in V_i for which the application of a snapshot transition at time t_{i+1} causes v to update its information concerning neighboring nodes or edges.
- $GT_i = G_i\,|_{VT_i}$ as the *restriction of* G *to* VT_i. This defines the *transition-affected subgraph* associated with a transition from snapshot S_i to S_{i+1}.

Distribution Scheme: Let $\langle S_i, ..., S_{i+k}\rangle$ be a sequence of pre-computed snapshots that is to be distributed. The following sets define the predictable changes for a transition point t_i:

$$\begin{aligned}\texttt{add\_edges}(i) &= E_{i+1} \setminus E_i\\ \texttt{delete\_edges}(i) &= E_i \setminus E_{i+1}\\ \texttt{add\_nodes}(i) &= V_{i+1} \setminus V_i\\ \texttt{delete\_nodes}(i) &= V_i \setminus V_{i+1}\end{aligned}$$

Algorithm 1 describes, in high-level pseudo-code, the steps that are taken by the ground station for each sequence of k

snapshots. A new snapshot S_j is calculated from S_{j-1} using δ (line 3). The computed snapshot is then updated with the information stored on past unpredictable changes (line 4). Afterwards, a time stamp is added, indicating the time point at which the most recent unpredictable change has reached the ground station (line 6). Then, all entries are removed from the data structure `UC` (line 7), which holds all unpredictable changes received since the last computation. Finally, the new snapshot sequence is distributed to the other nodes in the network (line 8–10).

Algorithm 1 Snapshot Sequence Calculation and Distribution Function `snap_calc`(i, k) for a ground station

Takes i (first snapshot index of the new sequence) and k (number of snapshots in the sequence) as input.

```
1:  var $S_{i-1}$,...,$S_{i+k}$ /* Snapshot variables, where $S_{i-1}$ contains the last computed snapshot change */
2:  for all j=0 to k do
3:      $S_{i+j} \leftarrow \delta(S_{i+j-1})$
4:      $S_{i+j} \leftarrow$ ApplyStoredUnpredictableChanges($S_{i+j}$)
5:  end for
6:  UpdateTimeStamp($\langle S_i, ..., S_{i+k} \rangle$)
7:  DeleteAllChanges(UC)
8:  for all Neighbors $x$ of $GS$ /* Start Flooding*/ do
9:      Send ($\langle S_i, ..., S_{i+k} \rangle$, $x$)
10: end for
```

Distribution Scheme Efficiency: The change in the topology $\delta(S_i)$ is limited to the subgraph GT_i of G_i. Adding or deleting an edge from E_i involves two nodes x and y and one edge (x, y) in GT_i and adds only a constant amount of space to the size of the update for S_{i+1}. For each node that is added or deleted from V_i, the number of edges for that change is limited to the number of neighboring edges, which is $2 \mid V_i \mid$ edges for a node change, in the worst case. Given this, we can reduce the data overhead by propagating just the differences between the snapshots S_i and S_{i+1}. Note that instead of using flooding (as in Algorithm 1), one could alternatively use a more efficient multicast solution, which would also increase the scalability of the solution.

V. Routing Protocol

We now specify our PLSR protocol for predictable mobile networks. We build it on top of (basic) link-state routing, with extensions to handle predictable changes described by snapshot sequences. We also prove the protocol correct.

A. Routing Protocol Baseline Selection

Predictable mobile networks share many properties with mobile ad-hoc and static networks. We therefore adapt an existing approach rather than developing a completely new protocol. We start by discussing the rationale for the link-state baseline protocol that we have selected.

Appropriateness of Link-State Routing: Ad-hoc routing protocols use specialized techniques to provide routing in environments consisting of randomly-moving, resource-constrained nodes. Our environment provides substantially more information as well as special nodes without resource restrictions. Unmodified ad-hoc routing protocols are easily seen to be inadequate for predictable mobile networks since they cannot manage the a-priori knowledge about predictable changes and must therefore consider all changes as unpredictable. Moreover, in space applications, rapid changes often occur, i.e., over short time intervals. As a result, ad-hoc routing protocols would produce substantial routing overhead and would be unable to maintain stable routing tables, especially if long propagation delays are involved.

In contrast to the above, static networks do not suffer from changes caused by network mobility. Routing protocols can therefore build extensive and accurate information on the network topology, which is also a desirable property for predictable mobile network routing. We assess the two static network protocol families: link-state routing, as represented by the OSPF-2 protocol [11], and distance-vector routing, as represented by RIP [12]. We compare these protocols with respect to the computational and memory resources of an onboard spacecraft computer [10].

In [13], a performance analysis of the OSPF-2 protocol is undertaken. The OSPF protocol has higher memory requirements than distance-vector protocols. Still, the required memory for a large number of nodes is far less than is available on spacecrafts. [13] also states that link-state routing has a better computational performance than distance-vector routing because a complete run of the Dijkstra algorithm is not required after each update of the local topology image; instead, an incremental update suffices. This is particularly beneficial in predictable mobile networks since relatively few nodes belong to each transition-affected subgraph (recall Definition 4).

For these reasons, we choose the link-state approach based on a combination of static and ad-hoc routing. As a baseline, we use the core link-state protocol that is part of both, OLSR and OSPF and contains the most essential link-state functionalities. In this protocol, each node maintains its own view of the complete network topology in the form of a link-state database (LSDB), which constitutes a graph. The local shortest path first (SPF) routing tree, which acts as a routing table for the routing protocol, is calculated from the LSDB using Dijkstra's algorithm [14]. A node that detects an (unpredictable) change in an adjacent node or edge updates its LSDB and SPF tree and then propagates the change through the network using a flooding algorithm.

B. Link-State Algorithm

Each unpredictable change u describes the addition or deletion of a single edge in the topology graph. When a node changes its LSDB to incorporate such a modification, we say that *the node has updated its LSDB with the change dictated by* u. Since the unpredictable change u affects only one edge in the network, only the adjacent nodes can discover the occurrence of u using link-sensing. The adjacent nodes will, after detecting the change, initiate a Link-State Advertisement (LSA), describing the unpredictable change u. We model the message sent describing u with the tuple (x, y, t, b) of type $V_i \times V_i \times T \times \{0, 1\}$ where, for a snapshot S_i, $(x, y) \in E_i$ is the affected edge, $t \in T$, with $t_i \leq t < t_{i+1}$ is the time point when u occurs and b is 0, if the link goes down and 1 otherwise. Note that more than one unpredictable change may occur at the same time point. For example, this occurs when a node with more than one adjacent edge fails.

If two consecutive unpredictable changes u and u' affect the same edge, then u' always reverses the modifications dictated by u. Furthermore, we say an unpredictable change u is *superseded* by another unpredictable change u', if u' occurs after u and modifies the same edge as u.

Algorithm 2 Basic link-state main loop for a node v

```
1:  var $x$ /* Variable $x$ describes a change */
2:  var $lsa$ /* Variable $lsa$ contains an LSA as described in Section V-B*/
3:  while true do
4:     if $x$ = DetectChange() then
5:        UpdateLSDB($x$) /* Updates LSDB with $x$ */
6:        $lsa \leftarrow$ CreateLSA($x$) /* New LSA from $x$ */
7:        ModifySPFTree()
8:        /* LSA Flooding */
9:        for all neighbors $y$ of $v$ /* $v$ is the current node */ do
10:          Send ($lsa, y$) /* Send $lsa$ to $v$ */
11:       end for
12:    end if
13:    if ($lsa$, $w$) $\leftarrow$ Receive() /* Receive lsa from $w$ */ then
14:       if IsFresh($lsa$) then
15:          UpdateLSDB($lsa$)
16:          ModifySPFTree()
17:          for all neighbors $s$ of $v$ except $w$ /* LSA Flooding */ do
18:             Send($lsa, y$)
19:          end for
20:       else
21:          Drop($lsa$) /* LSA not recent */
22:       end if
23:    end if
24: end while
```

Algorithm Description: Algorithm 2 shows the routing main loop of the basic link-state protocol for a node v in the network. In the algorithm, if v detects a change on the link to another node x (line 4), it adjusts its own LSDB and then creates an LSA message for the change (lines 5 and 6). The shortest path first (SPF) tree is updated for the new LSDB (line 7). This LSA is then flooded to all other nodes in the topology (line 9–11). If v receives an LSA from a node w (line 13), it checks its freshness and, depending on the result, the LSA is either processed (lines 15–19) or dropped (line 21). Processing an LSA includes updating the local LSDB with the changes, continuing the flooding, and storing the LSA for future freshness checks. This flooding mechanism allows for rapid propagation of network changes in the network at the cost of increased traffic overhead.

C. Requirements and Assumptions

We adapt the above protocol for use in predictable mobile environments by incorporating information from snapshot sequences. Before describing our adaptations, we state our requirements for the protocol. In particular, we require two notions of consistency.

Definition 5: Change Consistency. Let S_i be the current snapshot.

- *Predictable change consistency* concerning a predictable change at time t_{i+1} holds if there exists a $t \in T$ with $t_{i+1} < t < t_{i+2}$ and all nodes $v \in V_{i+1}$ have a consistent view of the predictable change at t_{i+1} by time point t.
- *Unpredictable change consistency* concerning an unpredictable change u that occurs at time point t, with $t_i \leq t < t_{i+1}$, holds if at some time point $\hat{t}$ associated with some snapshot S_k, with $k > i$ and $t_k \leq \hat{t} < t_{k+1}$ one of the following has occurred: (1) all nodes $v \in V_k$ have updated their LSDB with the change dictated by u (i.e. add or delete an edge as described by `unpred_change(`x, y, t`)`), (2) u has been superseded by another unpredictable change u' that occurs at a time point t' with $t \leq t' \leq \hat{t}$, or (3) all nodes that have updated their LSDB with the changes dictated by u are deleted by $\hat{t}$ before they can further propagate u.

Predictable change consistency is achieved when the nodes have updated their LSDB as dictated by the last predictable change. Therefore, the time interval of inconsistency is very small in practice and no communication is required to achieve predictable change consistency. In contrast, for an unpredictable change u, it cannot be guaranteed that the propagation process is completed within in the same snapshot in which u occurs. The conditions state:

- either all nodes have learned of the change u or
- the change is no longer relevant as it has (1) been superseded or (2) all information on u has been deleted from the topology

Note that unpredictable change consistency is also a requirement for the basic link-state scheme (Algorithm 2).

The link-state databases of all network nodes reflect the reality at time t (in snapshot $\hat{S}_t$) only if predictable change consistency holds for all previous snapshot transitions and unpredictable change consistency holds for all unpredictable changes that have occurred prior to t. As unpredictable changes are not expected to occur often in spacecraft networks, the nodes' LSDBs will often reflect reality.

Network Separation through Unpredictable Changes: Before we state our requirements, we consider a pathological case that may occur as a consequence of an unpredictable change. Consider an unpredictable change u that separates the topology into the main graph (containing the ground station) and an isolated subgraph. The subgraph can neither propagate any further unpredictable changes that affect one of its edges, nor can it receive snapshot sequence update messages from the ground station. Observe that after the time point when the nodes in the separated subgraph run out of snapshots, all predictable changes must be treated as unpredictable changes.

We will not consider this pathological case in our paper since it is very unlikely to occur in spacecraft networks. For example, spacecraft networks use redundant nodes to avoid network separation. We only sketch a solution here. Namely, we can equip all nodes with the capability of switching into a "full ad-hoc" mode that becomes active as soon as the nodes of a separated subgraph exhaust their snapshot information. When the connection to the ground station is restored and fresh snapshot sequences can be acquired, the nodes switch back to predictable mobile routing mode and notify the main topology about unpredictable changes that may have occurred during the time of separation.

Protocol Requirements: Our requirements are twofold: (1) following a predictable change in the network topology, predictable change consistency holds and (2) following an unpredictable change in the network topology, unpredictable change consistency holds.

Meeting the first requirement increases the quality of the packet forwarding process since all routing decisions of PLSR are correct whenever no unpredictable changes are currently propagating through the network. As predictable change consistency holds following each snapshot transition S_i to S_{i+1}, the time interval $[t_{i+1}, t]$, for $t \in T$ with $t_{i+1} < t < t_{i+2}$, where the routing tables are inconsistent with the network topology is negligible. If packets are forwarded during $[t_{i+1}, t]$, then errors may arise as either the previous snapshot is used or the LSDB of the forwarding node is in an inconsistent state. However, if the forwarding process is synchronized with Algorithm 3 so that forwarding is delayed between lines 6 and 13 (where the routing tables are being rebuilt), then all packets are correctly forwarded with respect to the current network topology. Note that this effect on the quality of packet forwarding is a special property of topologies where predictable change consistency holds. It does not hold for unpredictable changes and it therefore also fails for other non-predictable networks.

Assumptions: For the definition of the algorithm and its correctness proof, we make the following assumptions:

1) Clock synchronization[1] is required in order to synchronously apply the snapshot transition at all nodes.
2) We assume that at a transition point t_i, each node has already received the information required to calculate the next snapshot S_{i+1} as described in Section IV-A.
3) The time period required for local node computations, such as LSDB updates and SPF tree calculations, is negligible.
4) We assume that a flooding algorithm, initiated by a set of nodes $V \subseteq V_i$ during a snapshot S_i, reaches all other nodes $w \in S_j$, for some $j \in T$ and $j \geq i$.
5) The topology graph is always connected (also after both predictable and unpredictable changes).

Although clock synchronization is a common problem in distributed systems, spacecrafts employ high-precision clocks as many of their commands are time critical. Therefore, only small clock deviations are expected and this assumption is reasonable for spacecraft networks. Although Algorithm 3 includes the reception of new snapshot sequences, we do not consider this in our proof since distribution can be decoupled from the routing algorithm itself. Assumption 3 is also reasonable given the capacities of modern spacecraft on-board computers. Restricting the network dynamics in Assumption 4 expresses the need for the stability of the network topology and its evolution in terms of predictable changes. In particular, it excludes anomalies such as a topology that grows faster than the time required for messages to spread through the network. Assumption 5 is motivated by our previous discussion on network separation.

D. Routing Protocol Definition

Algorithm 3 presents our link-state routing algorithm for predictable mobile networks. This algorithm handles both predictable and unpredictable changes, as well as the interaction between the two. Because unpredictable changes cause the snapshot abstraction to deviate from reality, their interaction with the predictable changes must be specifically addressed.

Predictable Changes: For each transition point t_{i+1}, GT_i is the transition-affected subgraph. We modify the core routing protocol to handle predictable changes and update the topology image at each transition point t_{i+1} (line 6). Following our assumption, information on snapshot changes (`add_edges`(i), `delete_edges`(i), `add_nodes`(i), and `delete_nodes`(i)) for the next snapshot has already been received and hence the LSDBs can be updated locally at the transition point t_{i+1} (line 8). Since only nodes in GT_i are affected, a complete re-run using the Dijkstra algorithm is not required (only an incremental update) to compute the local SPF trees (line 13). However, if a node is added by a predictable change, this node must learn of any recent unpredictable changes from one of its neighbors. Therefore all nodes neighboring the newly added node create an LSA for each unpredictable change that they have stored in their `LSDB_UC` and send them to the new node (line 10).

Unpredictable Changes: During the time interval $]t_i, t_{i+1}[$ within a snapshot S_i, our core link-state algorithm operates as described in Algorithm 2. Moreover, an unpredictable change that occurs during $]t_i, t_{i+1}[$, or is received via an LSA is stored in an additional data structure `LSDB_UC` (lines 22 and 33). Namely, the tuple (x, y, t, b) is stored together with a time stamp indicating the reception or detection time of the most recent unpredictable change. Should an unpredictable change result in the addition of a new node (line 16), this node must be provided with the current LSDB (line 18).

[1]We will assume perfect clock synchronization in our proof for the sake of simplification. Our routing protocol would also work with small clock deviations between the nodes.

Interaction: Interaction between predictable and unpredictable changes occurs at transition points when an unpredictable change directly affects an edge with a neighboring node in GT_i. In this situation, the order in which the changes are applied is important. We first apply the predictable changes to the LSDB (line 8) and then the unpredictable changes, which are recorded in the data structure `LSDB_UC` (line 12). In this way, unpredictable changes are preserved across snapshot transitions.

Reception of Snapshot Sequences: If a new snapshot sequence is received from another node (line 40), its time stamp is retrieved and the new sequence is stored for later use (line 41). Afterwards, obsolete entries are deleted from `LSDB_UC` (line 42) and the new sequence is further distributed (lines 43–45).

E. Protocol Correctness

We now prove that our routing protocol is correct.

Theorem 1 (Protocol Correctness): The PLSR Protocol has the two requirements defined in Definition 5.

Proof: Our proof proceeds in two parts.

- Part 1: We show that predictable changes satisfy our routing protocol requirements, that is, that predictable change consistency holds following each transition.
- Part 2: We show that unpredictable changes also satisfy our requirements, that is, that unpredictable change consistency holds for each unpredictable change u.

Part 1: Consider a snapshot transition from S_i to S_{i+1}. At the transition point t_{i+1}, each node $v \in V_i$ updates its LSDB locally (line 8) using previously distributed snapshot information (Assumption 2). As clocks are synchronized, the LSDB update process happens simultaneously on all nodes at time t_{i+1} (line 6). Let $t \in T$, with $t_{i+1} < t$, be the time when all nodes have completed their LSDB update for this snapshot transition. By Assumption 3, $[t_{i+1}, t]$ is negligible and therefore $t < t_{i+2}$. Hence, all nodes in V_{i+1} reach a consistent view of the network concerning this predictable change at time t within S_{i+1} and thereby achieve predictable change consistency.

Part 2: Let u be an unpredictable change. We show that neither the occurrence of further snapshot transitions nor subsequent unpredictable changes prevent establishing unpredictable change consistency following u.

For our proof, we require notation for describing the set of nodes in the topology that have updated their LSDB with the changes dictated by u at some time point t. Let u be an unpredictable change that is discovered at a time point $t_{u_0} \in T$ and let t_{u_n} be the first time point when either all nodes in the topology have updated their LSDB with the changes dictated by u, u has been superseded, or all information on u has been deleted from the topology. We define V_u^t as a family of sets indexed by t, with $t \in \{t_{u_0},...,t_{u_n}\}$, that describes how information on u propagates over time. In particular $V_u^t = \{v \in V_i \mid t_i \leq t < t_{i+1}$ for a snapshot S_i and v has updated its LSDB by time t with the changes dictated by $u\}$. Therefore

Algorithm 3 PLSR main loop for a node v.

```
1:  var $x$ /* describes a change */
2:  var $lsa$ /* LSA as described in Section V-D */
3:  var $S_i$,...,$S_{i+k}$ /* Snapshot variables, contain a snapshot
    sequence consisting of $k$ snapshots updates */
4:  var $n, s, v, w$ /* Variables describe nodes */
5:  while true do
6:      if CurrentTime() == NextTransitionPoint() then
7:          $t \leftarrow$ CurrentTime()
8:          $newNodes \leftarrow$ UpdateLSDB(LoadSnapshotChanges($t$))
9:          for all neighbors $n$ in $newNodes$ do
10:             SendAllChanges(LSDB_UC, $n$)
11:         end for
12:         LoadUnpredictableChanges()
13:         ModifySPFTree()
14:     end if
15:     if $x \leftarrow$ DetectChange() then
16:         $n \leftarrow$ GetNeighborNode($x$)
17:         if $n \notin$LSDB then
18:             Send(LSDB, $n$) // Send LSDB to new node $n$
19:         end if
20:         UpdateLSDB($x$) /* Updates LSDB */
21:         ModifySPFTree()
22:         UpdateLSDB_UC($x$)
23:         $lsa \leftarrow$ CreateLSA($x$) /* New LSA from $x$ */
24:         for all neighbors $w$ of $v$ do
25:             Send ($lsa, w$) /* Send $lsa$ to $w$ */
26:         end for
27:     end if
28:     /* Receive an LSA $lsa$ from a node $s$ */
29:     if ($lsa$, $s$) $\leftarrow$ Receive() then
30:         if IsFresh($lsa$) then
31:             UpdateLSDB($lsa$)
32:             ModifySPFTree()
33:             UpdateLSDB_UC($lsa$)
34:             for all neighbors $w$ of $v$ except $s$ /* LSA Flooding
                */ do
35:                 Send($lsa, w$)
36:             end for
37:         end if
38:     end if
39:     /* Receive a new snapshot sequence*/
40:     if $\langle S_i, ..., S_{i+k} \rangle \leftarrow$ Receive() then
41:         timestamp $\leftarrow$ StoreSnapshotSequence($\langle S_i, ..., S_{i+k} \rangle$)
42:         DeleteOldEntriesFromLSDB_UC(timestamp)
43:         for all neighbors $w$ of $v$ do
44:             Send ($\langle S_i, ..., S_{i+k} \rangle$, $w$)
45:         end for
46:     end if
47:     if $newLSDB \leftarrow$ Receive() then
48:         SetLSDB($newLSDB$)
49:     end if
50: end while
```

${V_u}^{t_{u_0}}$ contains the node that has discovered u through link-sensing and is neighboring the edge that is affected by u. This node then starts the propagation of the change dictated by u. In the following, we use the term *add nodes to* V_u^t to describe that, for $t = t_{u_i}$, nodes $v_0, ..., v_k$ have been added to ${V_u}^{t_{u_i}}$ to get ${V_u}^{t_{u_{i+1}}}$, that is, ${V_u}^{t_{u_{i+1}}} = {V_u}^{t_{u_i}} \cup \{v_0, ...v_k\}$. We use the term *delete nodes from* V_u^t analogously.

We state the following lemma, whose proof follows our proof of the main theorem.

Lemma 1: Properties of V_u^t

Let S_i be a snapshot and $t \in T$, such that $t \in [t_i, t_{i+1}[$.

1) The snapshot transition at t_{i+1} may cause the addition or deletion of nodes from V_u^t by predictable changes.
2) If $t \in [t_{u_0}, t_{t_{u_n}}]$ and nodes of V_u^t are deleted from the topology by a series of (predictable or unpredictable) changes, either the changes dictated by u have (2a) reached all nodes in the topology, (2b) continue propagating, or (2c) other unpredictable changes occur that originate from the deletion of all nodes in V_u^t.
3) An LSA broadcast originating from a node in V_u^t, with $t \in [t_{u_0}, t_{t_{u_n}}[$, will eventually reach all other nodes in the network by some time point $\hat{t}$ associated with some snapshot S_k, for $t_k \geq t_i$, $t_k \in I$, and $t_k \leq \hat{t} < t_{k+1}$, unless u is superseded by a more recent unpredictable change or all nodes in V_u^t are deleted.

Returning to the proof of the theorem, consider an unpredictable change u that occurs at t_0, with $t_i \leq t_0 < t_{i+1}$. Following this, u triggers the creation of $V_u^{t_0}$, which contains the node that discovered u using link-sensing. This node starts propagating the changes dictated by u. By part (3) of the lemma, either the propagation will reach all nodes in the network at some time point $\hat{t}$ in some snapshot S_k (then $V_u^{\hat{t}} = V_k$), u is superseded by more recent unpredictable change u' ($V_{u'}^{\hat{t}} = V_k$, if u' has been fully propagated by $\hat{t}$), or all nodes in V_u^t are deleted during the propagation process ($V_u^{\hat{t}} = \emptyset$). In the last case, a number of new unpredictable changes u'_0,...,u'_n may originate from the deletion.

■

We now prove Lemma 1.

Proof of Lemma: We prove the three parts individually.

Part (1). Suppose the snapshot transition from S_i to S_{i+1} affects V_u^t. The transition may delete or add nodes. First, consider the deletion of nodes. Due to predictable change consistency and the storage before the transitions (lines 22 and 33) and re-application of topology modifications dictated by unpredictable changes after the transition (line 12), it follows that either $V_u^{t_{i+1}} = V_u^t$ (no nodes are deleted) or $V_u^{t_{i+1}} = V_u^t \backslash \{x_0,...,x_n\}$ if the nodes $\{x_0,...,x_n\} \subset V_i$ have been deleted by the predictable change (line 8). Alternatively, consider the addition of nodes. Let $\{v_0,...,v_n\}$ be the set of nodes that are added to the topology by the snapshot transition (line 8) and have the property that each $v \in \{v_0,...,v_n\}$ is neighboring at least one node in V_u^t. Then, $V_u^{t_{i+1}} = V_u^t \cup \{v_0,...,v_n\}$ (line 10).

Part (2). Because of (1), predictable changes may delete (one or more) nodes from V_u^t . For t_i, this happens if $V_u^t \cap \texttt{delete\_nodes}(i-1) \neq \emptyset$. Furthermore, the deletion of nodes from V_u^t can be caused by unpredictable changes at any time point t' with $t' \in [t_{u_0}, t_{u_n}[$. Because of Assumption 5, however, node deletion will never separate the topology graph. In the following, we consider the two cases that may arise when nodes are deleted from V_u^t.

A) The nodes in $\{x_0, ..., x_n\} \subsetneq V_u^t$ are deleted at time point t. This deletion of nodes in $\{x_0, ..., x_n\}$ does not affect the propagation of u because either the nodes in $V_u^t \setminus \{x_0, ..., x_n\}$ continue the flooding process (lines 34–36) (case 2b) or the propagation was already completed (case 2a).

B) Should all nodes from V_u^t be eventually deleted by a series of (predictable or unpredictable) changes before some of them can further propagate u, then the flooding is terminated. If only predictable changes contribute to the deletion process, no action is required since the affected edge is no longer part of the local LSDBs and therefore u has no further impact on the topology. Alternatively, when unpredictable changes contribute to the deletion process of the nodes from V_u^t, then the flooding of u is also terminated but further action is required. In particular, for a snapshot S_i, nodes in $V_i \setminus V_u^t$ that have been neighbors of the deleted nodes in V_u^t will discover the deletion through link-sensing (line 15) and a series of new unpredictable changes u'_0,...,u'_n is generated (one unpredictable change for each link-sensing event that discovered a change). These changes are then subsequently propagated (lines 24–26). Following this, the deletion of all nodes from V_u^t causes the removal of the changes dictated by u from the topology (case 2c in the lemma).

Part (3). The general case, in which the propagation of u to all nodes in the topology happens without the predictable or unpredictable addition or deletion of nodes from V_u^t, is covered by Assumption 4. Part (2) shows that the deletion of nodes either has no effect on the propagation or all nodes that have integrated the changes dictated by u are deleted. Predictable changes may cause the addition of nodes to V_u^t (Part 1). Since this increases the number of nodes in V_u^t, propagation of u is either continued or completed. Hence, it remains to be shown that (3) also holds if new nodes are added by unpredictable changes (line 15). Let u be an unpredictable change that adds a node w to the topology at time t, and let v be a neighboring node of w. If $v \in V_u^t$, w is provided with information on u upon detection by v (line 18). Otherwise, if $v \notin V_u^t$, then w will eventually receive information on u through the standard propagation mechanism (lines 34–36), as is stipulated by Assumption 4. ■

VI. Routing Protocol Performance

We now assess the performance of the PLSR protocol, both analytically and using simulation.

A. PLSR Performance Analysis

Recall that each δ function update is limited to the subgraph GT_i for snapshot S_i and that a complete run of the underlying

Dijkstra algorithm is not required to update the local SPF trees. Therefore, the runtime complexity is on the order of the complexity of adding or deleting a set of nodes in the basic link-state algorithm (Algorithm 2). Also, for PLSR, snapshot transitions are inexpensive in terms of network communication resources since predictable changes do not create routing messages.

We must also take into account the PLSR routing overhead that is caused by snapshot image messages that result from the snapshot distribution process described in Section IV-A. If, for each snapshot S_i, only the changes with respect to the previous snapshot are transmitted, the snapshot information update only needs to transmit the tuple (x, y, b) of type $V_i \times V_i \times \{0, 1\}$ for each predictable change, where b is 0 when the edge (x, y) goes down and 1 otherwise. Assuming that a multicast message from the ground station requires n 1-hop transmissions to reach all nodes in the network, then the total overhead in terms of data for the distribution of a single change is $2n| V_i |^2$.

For unpredictable changes, their occurrence is handled like in the basic link-state protocol (Algorithm 2), but the information is also preserved across snapshots using the data structure `LSDB_UC`. As unpredictable changes are expected to occur only infrequently, `LSDB_UC` does not require much memory. The above observations suggest that the PLSR protocol is efficient enough in terms of memory and computational resources to be used in spacecraft on-board computers [10].

B. Simulation

We compare PLSR's performance in a predictable mobile network with the performance of the Optimized Link State Routing (OLSR) protocol using the ns-2 simulator [15] with the standard wireless communication model. We first describe the set-up of our simulations and afterwards the results. Our simulations focus on routing message overhead and traffic throughput since these are the most relevant metrics for spacecraft networks. We do not measure the convergence time for unpredictable changes since the majority of the changes are predictable and therefore PLSR routing is still possible during convergence, even if suboptimal.

We have implemented our PLSR protocol, described in Algorithm 3, by adapting an implementation of the OLSR protocol. We have performed our analysis in a general setting, independent of any specific spacecraft-network scenario. In particular, our topology generation tool simulates the random movement of nodes. This underlines the generic properties of the PLSR protocol but also leads to a large confidence interval (i.e. topologies with extremely low and high measurement results may exist). We therefore performed many simulation runs for each configuration and averaged the results. The vast majority of changes are predictable, but an unpredictable change occurs with a probability of 0.1 in an entire simulation run (100 seconds of simulated time). Both protocols, OLSR and PLSR, run on the same topology. We performed simulation runs for random, connected topologies with 5, 10, 15, and 20 nodes. Note that these are realistic sizes for typical spacecraft networks.

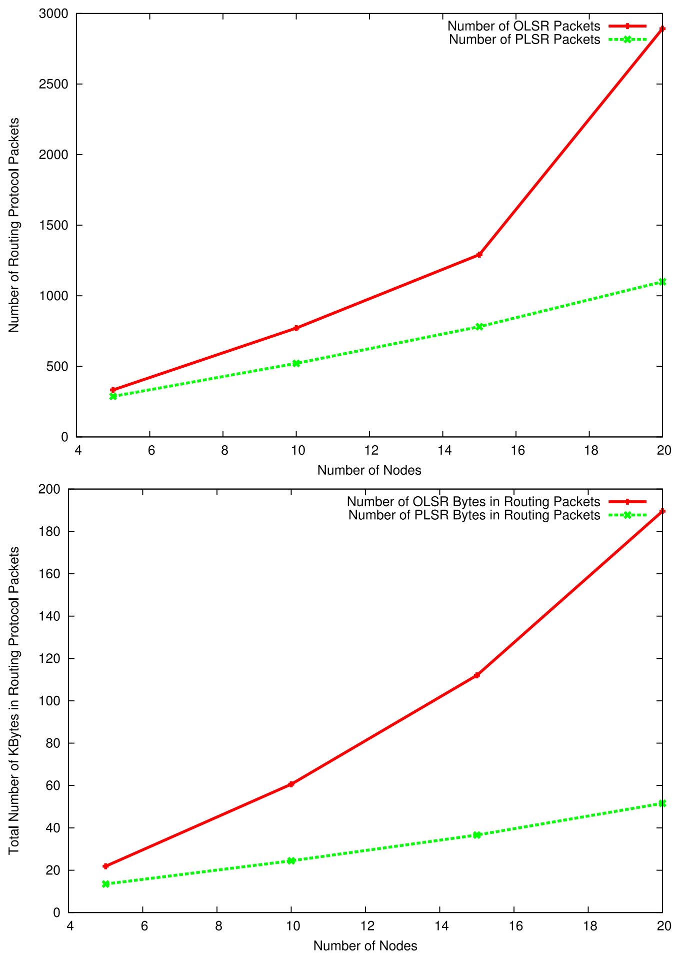

Fig. 2. Simulation Results for Routing Overhead

We measured the number of routing protocol packets and the associated overhead in Kilobytes. From the results, illustrated in Figure 2, we see that PLSR uses substantially less routing messages and has less traffic overhead than OLSR. The difference to the original OLSR protocol becomes larger as the number of nodes grows. The size and number of the required flooding messages is responsible for this. The PLSR protocol produces a higher number of routing messages and traffic in the simulation than one might expect from the analytical performance analysis. One reason for this is that periodic link-sensing for unpredictable changes is still necessary. The other reason is that in a topology with totally random node movement, an unpredictable change is most likely to be discovered (and lost) multiple times by the same nodes. This is because the unpredictable change may iteratively enter and leave the nodes' link-sensing range. Therefore, one unpredictable change may initiate multiple flooding processes. In a typical application scenario, such as a LEO spacecraft network, this kind of interaction with repeated flooding is much less likely to occur.

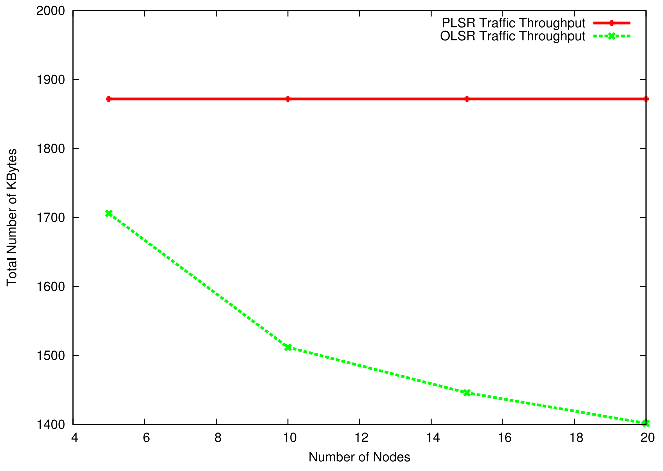

Fig. 3. Simulation Results for Traffic Throughput

We further measured the traffic throughput between two random nodes for PLSR and OLSR using a traffic production of 200 packets per second. In Figure 3 we can see that PLSR provides almost constant maximum throughput (the small deviations are due to unpredictable changes) since it has access to topology information. The OLSR throughput is always less since OLSR lacks this ability and all changes are therefore unpredictable. Furthermore, the OLSR measurement results show a degradation of the traffic throughput. This degradation comes from the fact that the number of changes increases with the number of nodes. Moreover, should the propagation delay between the nodes grow larger, the OLSR performance degrades further.

To summarize, our simulation results support the thesis that the PLSR protocol is better suited for predictable mobile networks than OLSR. These are general simulation results and we expect that, by accounting for parameters specific to spacecrafts, the performance gain will be even more substantial. Such an investigation is the subject of ongoing work.

VII. Related Work

For LEO constellations, a topology model can be constructed based on Manhattan networks. In [16], Maxemchuk investigates such networks and proposes routing solutions. Ferreira *et. al* [17] present a combinatorial model for MANETs along with approaches for solving different routing-over-time problems. Merungu *et. al* [18] propose a space-time routing framework based on time-evolving graphs rather than snapshots. However, they focus on the problems of handling temporal disconnection and the minimization of associated packet buffering times. This makes their approach similar to Delay Tolerant Network (DTN) routing. They do not consider unpredictable changes.

Other routing schemes have been proposed for spacecraft networks. In [19], Donner *et. al* describe a networking concept based on Multiprotocol Label-Switching (MPLS) for spacecraft constellations. They avoid however the key issue of proper route adaptation by assuming permanent and periodic topologies. Also, the usage of spacecraft-based LSDBs is rejected because of the extensive link-state communication. We solve this problem by only using link-state communication in response to unpredictable changes. Zang *et. al* [20] provide an overview on routing protocols in DTNs and other intermittently connected MANETs. They describe deterministic protocols where they assume that all future movements and connections are known to one or more protocol entities beforehand. Surprisingly, Zhang *et. al.* do not mention spacecraft networks in this context.

During the time that LEO constellations appeared to be a promising technology, several routing protocols were proposed, including simple datagram routing [3], [21], topological network design [4], routing based on geographic locations [5] and routing in multi-layered constellations [22]. All of these were non-generic, that is, specific to the LEO architecture. Several proposals have advocated the use of Asynchronous Transfer Mode (ATM) based routing in LEO space networks [23], [6]. However, the large success of the IP protocol makes integration with these networks difficult. Hashimoto et al. [7] presents a hybrid approach using IP and ATM.

In deep-space communications, long propagation delays and intermittent connectivity links may be present in the network. The DTN approach [24] addresses this problem by introducing an overlay layer to handle these obstacles. DTN solutions are compatible with the connectionless packet-switched network that we assume. In [25], Jain *et. al* discuss routing problems in DTNs. While their focus on routing with finite buffers, they also address DTN routing and flow control with complete knowledge. They do not, however, address unpredictable failures and their impact on the topology snapshots.

VIII. Conclusions and Future Work

We have presented a model of predictable mobile topologies and used it to design a routing algorithm, Predictable Link-State Routing, that exploits the predictability of network connections. We have shown that our algorithm is correct and performs well compared to other mobile link-state routing protocols.

In our ongoing work, we use and validate the topology model and the proposed PLSR protocol in real-world application scenarios of the European Space Agency. These include a LEO/GEO spacecraft hybrid network and communication between a terrestrial ground station and a Mars Rover using relay spacecrafts. We derive snapshot sequences from flight dynamics data and perform simulations for both scenarios including spacecraft and ground-station failures. Preliminary results suggest that in these real-world settings, PLSR better supports high-bandwidth data streaming and produces substantially less routing overhead than other link-state and distance-vector based routing protocols. Although the two application scenarios are quite different, our model is generic enough to serve them both.

The topology model and PLSR protocol that we have proposed in this paper are general purpose. Nevertheless, we have focused on spacecraft networks as a target application area and explained why our approach is well suited for this domain. Spacecraft networks do, however, have other features that we have not considered in this paper. In particular, a spacecraft network should be able to integrate with external nodes and networks, including other spacecraft networks. While our model and algorithm should be extensible in this respect, such integration is the subject of future work. We will also investigate the scalability of PLSR and snapshot update delivery in networks with substantially more than 20 nodes and the behavior of PLSR when errors occur in the time synchronization of the nodes.

Finally, as most of the foreseen applications of space networks will support critical infrastructures, network integration also necessitates securing the PLSR protocol. In our future work, we will also investigate security requirements and possible solutions.

References

[1] B. Evans, M. Werner, E. Lutz, M. Bousquet, G. E. Corazza, G. Maral, R. Rumeau, and E. Ferro, "Integration of satellite and terrestrial systems in future multimedia communications," *IEEE Wireless Communications*, vol. 12, no. 5, pp. 72–80, October 2005.

[2] C. E. Perkins, *Ad Hoc Networking*, 2nd ed. Reading, MA: Addison-Wesley, 2004.

[3] V. V. Gounder, R. Prakash, and H. Abu-Amara, "Routing in LEO-based satellite networks," in *Wireless Communications and Systems, 1999 Emerging Technologies Symposium*. IEEE, April 1999, pp. 22.1–22.6.

[4] H. Chang, B. W. Kim, C. G. Lee, Y. Choi, S. L. Min, H. S. Yang, and C. S. Kim, "Topological design and routing for low-earth orbit satellite networks," in *Proceedings of the Global Telecommunications Conference, 1995*. IEEE, November 1995, pp. 529–535.

[5] O. Ercetin, S. Krishnamurthy, S. Dao, and L. Tassiulas, "A predictive QoS routing scheme for broadband low earth orbit satellite networks," in *The 11th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC 2000*. IEEE, September 2000, pp. 1064–1074.

[6] M. Werner, C. Delucchi, H. J. Vogel, G. Maral, and J. J. de Ridder, "ATM-based routing in LEO/MEO satellite networks with intersatellite links," *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 1, pp. 69 – 82, Januar 1997.

[7] Y. Hashimoto and B. Sarikaya, "Design of IP-based routing in a LEO satellite network," in *3rd International Workshop on Satellite-Based Information Services*. Mobicomm, October 1998, pp. 81–88.

[8] R. J. Leopold and A. Miller, "The IRIDIUM communications system," *IEEE Potentials*, vol. 12, no. 2, pp. 6–9, April 1993.

[9] D. P. Patterson, "Teledesic: a global broadband network," in *Proceedings of the IEEE Aerospace Conference, 1998*. IEEE, March 1998, pp. 547 – 552.

[10] AtmelCooperation, "TSC695F SPARC 32-bit space processor - user manual," 2003.

[11] *Open Shortest Path First V2*, RFC, Internet Engineering Task Force (IETF) Request for Comments 2328, April 1998.

[12] *Routing Information Protocol V2*, RFC, Internet Engineering Task Force (IETF) Request for Comments 2453, November 1998.

[13] *OSPF protocol analysis*, RFC, Internet Engineering Task Force (IETF) Request for Comments 1245, July 1991.

[14] E. W. Dijkstra, "A note on two problems in connection with graphs," *Numer. Math*, vol. 1, no. 1, pp. 269–271, 1959.

[15] S. Bajaj, L. Breslau, D. Estrin, K. Fall, S. Floyd, P. Haldar, M. Handley, A. Helmy, J. Heidemann, P. Huang, S. Kumar, S. McCanne, R. Rejaie, P. Sharma, K. Varadhan, Y. Xu, H. Yu, and D. Zappala, "Improving simulation for network research," Technical Report, University of Southern California, 1999.

[16] N. Maxemchuk, "Routing in the manhattan street network," *IEEE Transactions on Communications*, vol. 35, no. 5, pp. 503 – 512, May 1987.

[17] A. Ferreira, "Building a reference combinatorial model for MANETs," *IEEE Network*, vol. 18, no. 5, pp. 24 – 29, October 2004.

[18] S. Merugu, M. Ammar, and E. Zegura, "Routing in space and time in networks with predictable mobility," Georgia Institute of Technology, GA, Tech. Rep. GIT-CC-04-07, 2004.

[19] A. Donner, M. Berioli, and M. Werner, "MPLS-based satellite constellation networks," *IEEE Selected Areas in Communications*, vol. 22, no. 3, pp. 438–448, April 2004.

[20] Z. Zhang, "Routing in intermittently connected mobile ad-hoc networks and delay tolerant networks: Overview and challenges," *IEEE Communications Surveys & Tutorials*, vol. 8, no. 1, pp. 24–37, April 2006.

[21] E. Ekici, I. Akyildiz, and M. Bender, "Datagram routing algorithm for LEO satellite networks," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*. IEEE, August 2002, pp. 500 – 508.

[22] I. F. Akyildiz, E. Ekici, S. Member, M. D. Bender, and S. Member, "MLSR: A novel routing algorithm for multilayered satellite IP networks," IEEE/ACM *Transactions on networking*, vol. 10, p. 3, 2002.

[23] A. Hung, M. Montpetit, and G. Kesidis, "ATM via satellite: A framework and implementation," *Wireless Networks*, vol. 4, pp. 141–153, 1998.

[24] K. Fall, "A delay-tolerant network architecture for challenged internets," in *SIGCOMM '03: Proceedings of the 2003 conference on applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2003, pp. 27–34.

[25] S. Jain, K. Fall, and R. Patra, "Routing in a delay tolerant network," in *SIGCOMM '04: Proceedings of the 2004 conference on applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2004, pp. 145–158.