A Delay-Tolerant Network Routing Algorithm Based on Column Generation

Guilherme Amantea, Hervé Rivano, Alfredo Goldman

A Delay-Tolerant Network Routing Algorithm Based on Column Generation

Guilherme Amantea
Instituto de Pesquisas Tecnológicas
São Paulo, São Paulo, Brazil
Email: gcamantea@ieee.org

Hervé Rivano
Inria Urbanet
Université de Lyon
INSA-Lyon, CITI-Inria, France
http://perso.citi.insa-lyon.fr/hrivano

Alfredo Goldman
Department of Computer Science
Instituto de Matemática e Estatística
Universidade de São Paulo
São Paulo, São Paulo, Brazil
Email: gold@ime.usp.br

***Abstract*—Delay-Tolerant Networks (DTN) model systems that are characterized by intermittent connectivity and frequent partitioning. Routing in DTNs has drawn much research effort recently. Since very different kinds of networks fall in the DTN category, many routing approaches have been proposed. In particular, the routing layer in some DTNs have information about the schedules of contacts between nodes and about data traffic demand. Such systems can benefit from a previously proposed routing algorithm based on linear programming that minimizes the average message delay. This algorithm, however, is known to have performance issues that limit its applicability to very simple scenarios. In this work, we propose an alternative linear programming approach for routing in Delay-Tolerant Networks. We show that our formulation is equivalent to that presented in a seminal work in this area, but it contains fewer LP constraints and has a structure suitable to the application of Column Generation (CG). Simulation shows that our CG implementation arrives at an optimal solution up to three orders of magnitude faster than the original linear program in the considered DTN examples.**

I. Introduction

Underwater, rural, vehicular, interplanetary and sensor networks are examples of systems that frequently have high latency, intermittent connectivity, low bandwidth, high packet drop rates and inexistence of end-to-end paths at any given point in time. Delay-Tolerant Network [1], or DTN, is a model that takes into account these characteristics. Consequently, DTN routing algorithms can present better performance in many of these networks than traditional protocols, such as TCP, that rely on a more stable environment.

Delay-Tolerant Networks can have known connectivity schedules, such as Low Earth Orbit satellites and ground stations, or unpredictable and opportunistic, such as sensor devices carried by wild animals. Recently, an increasing number of network traces from real installations have inspired the creation of routing algorithms [2], [3]. Connectivity patterns between nodes extracted from those traces help to improve the performance of protocols in networks without a fixed node contact schedule.

Routing algorithms applied in a typical DTN, except in the simplest cases, are fundamentally sub-optimal because they lack access to information normally unavailable during their execution. Examples of such information include buffer usage at every node in the network at any given time and future data traffic demand. These could be used to route messages in such a way to avoid collisions and packet drop due to the unavailability of buffer space at some node. Using network traces, a routing algorithm could have access to that information and produce optimal routing, which can be used as benchmark for comparison with other routing protocols.

A recent effort in this direction was done by Ferreira et al. [4], where evolving graphs were used to provide reference routing algorithms and protocols. Using the evolving graphs model, they can provide optimal lower bounds for delivering packets. However, when there are flows, the choice of which packet should be routed *first* is not trivial, and the problem becomes NP-complete again. Even with this limitation, it was possible to provide interesting tools which give approximative bounds [5] extremely fast. However, there is still the need for algorithms that provide optimal solutions.

An exact routing algorithm that minimizes the average message delay has been proposed by Alonso et al. [6] and Jain et al. [7]. This linear programming-based approach computes an optimal result, therefore requiring information about exact contact times between nodes, buffer occupancy at each node at each time, and sizes and injection times of all messages present in the system during the observation period. This optimal routing can be of great value since it can serve as reference against other protocols. According to Jain et al., however, even a simple scenario, which contains only a remote village exchanging messages with a city through three communication options, produced an LP formulation with close to 500,000 constraints and 550,000 variables and took 16,000 CPLEX iterations. Instances of realistic size are hence intractable.

Previous work [8] suggests that more investigation of solutions is needed. The largest fraction of research works citing the seminal paper of Jain et al. [7] reference it only to illustrate the existence of articles in the same topic as their own, typically in their related work section. There are some exceptions to this rule, but none of them, to the best of our knowledge, investigated alternative formulations to the same problem as that tackled by the optimal algorithm proposed by Jain et al.

In this work, we present an alternative LP formulation that has a smaller number of constraints and whose structure is suitable to the application of the well-known LP technique called Column Generation (CG). Using CG, the LP variables are not all explicitly listed. Instead, they are generated "as needed" during successive executions of much smaller subproblems of

the original problem.

This paper is organized as follows. The Related Work Section II discusses a classification of DTN routing protocols into *flooding* and *forwarding*. Then it mentions algorithms from the *forwarding* category and draws an alternative way to model one of these algorithms from the literature. Then it discusses another work which has investigated this model and hints at the key ideas of a well-known technique that can be applied to improve the performance of the discussed forwarding algorithm. The Problem Model Section III describes the linear programming formulation by the seminal work of Jain et al. [7], then defines a time-expanded graph model that can be used to create an alternative formulation to the same problem, which is also described in this Section. Then the two problem sizes are compared. The application of column generation on the new formulation is also described. Section IV, Experiments, presents three examples of simple networks used as input to the implementation of the original linear program and o the column generation implementation and their performance differences are discussed. Section V gives a summary of the observations described in this paper.

II. Related Work

DTNs can be classified in two categories: *flooding* and *forwarding*. Routing algorithms can operate in environments that are unreliable and unpredictable. The lack of information about the network removes from routing algorithms the possibility to know where to send a data packet with a high probability that it will eventually reach its destination. One way to increase that probability is to send multiple copies of each message to neighbor nodes. That way, a single packet drop is not sufficient for the message to be lost. Creating a larger number of copies, in ideal storage conditions, increases the chance that at least one copy reaches its destination. Also, on average, it decreases message delay, since there is a higher probability that some copies of the message will traverse the good, low-latency paths between source and destination. On the other hand, in real network installations, a large number of message copies can quickly fill node buffer storage. As a consequence, the network can become congested and a large packet drop rate may be observed.

DTN routing protocols classified as *forwarding* operate in more predictable environments. Some networks have stable topology or, at least, one that follows a known pattern. This allows algorithms to devise good forwarding paths between a message source and its destination. In some cases, the source node uses topology information to determine a good path and encodes this path into the message so that intermediate nodes know where to send it. In other cases, *per-hop* routing may be applied, in which a new trajectory for the message is calculated at each node in the message's path. A mixed approach is sometimes employed so that intermediate nodes can choose whether to follow the previously-calculated route or forward the message according to its own routing decisions, which may be based on more updated information about the network. Typical forwarding algorithms use only a single copy for each message. The obvious shortcoming of this approach is its limited applicability to networks in which the required information is available. If this cannot be provided *a priori*, the protocol needs some way to retrieve it during execution and disseminate it to other nodes.

Jain et al. [7] propose six forwarding algorithms with increasing levels of knowledge requirements. They use an abstraction called *Knowledge Oracle* to encapsulate a specific kind of information used by some of these algorithms. The simplest of them, called *First Contact*, uses no information about the network to forward messages except the list of neighbors each node has. Nodes employing this algorithm randomly choose one of their neighbors and forwards the message to it. Other proposed algorithms use information such as average time a node must wait until a certain edge becomes available for use or exact knowledge about at which point in time each edge will be available. As expected, they show in simulations that algorithms with access to more information have better performance.

One of the algorithms proposed by Jain et al. [7] is based on linear programming and uses all the available knowledge oracles, i.e. exact information about when contacts happen between any pair of nodes, contact durations, instantaneous buffer occupancy at any node and meta-data about all possible messages that are in the system during the execution of the algorithm. This algorithm is able to produce optimal routings with minimum average message delay, even considering buffer usage at all network nodes and avoiding congestion. However, this algorithm relies on a time-discretization step and the linear programming formulation depends on the number of nodes, edges and time intervals, which makes it very large. To illustrate, the authors observed that even a very simple instance with a city communicating with a remote village through three link options produced 500,000 constraints, 550,000 variables and 16,000 iterations. Creating alternative ways to tackle this same problem more efficiently is a main goal of this paper.

The linear program presented in the referred work [7] is not a standard multi-commodity minimum flow problem because the network is dynamic, which means edges may be available during some time intervals and unavailable in others. The classic network flows text by Ford and Fulkerson [9], however, suggests a way to expand the original network creating copies of each node for each time point. This way, copies of the original edges can be transported to the time-expanded graph between those nodes associated with time points at which the edges are available. Using this technique to model the dynamic properties of the network, a multi-commodity minimum cost flow solution can be used as if the network were static. This allows the application of well-known network flows techniques to increase performance.

Hay and Giaccone [10] describe a similar time-expanded graph to routing in DTNs. Although this graph represents the dynamic properties of a DTN, it can itself be used in a static context, which is exploited by the authors. They focus on a single source, single destination problem model (yet, as they mention, their model can be used to compute multi-commodity flows by applying linear programming). Algorithms are presented which compute a minimum delay path (with or without the minimum number of hops), maximum bandwidth path (with or without minimum delay), among others. Such procedures are based on graph theory such as Dijkstra's shortest path algorithm, breadth-first search and maximum flow algorithms.

Malandrino et al. [11] also apply a time-expanded graph in a DTN context. They model a system comprising communications-enabled vehicles and base stations connected to the Internet as a time-expanded graph and use this graph in an algorithm to find optimal routing of data that is downloaded from Internet-based servers to the vehicles. Such data includes news reporting, navigation maps or multimedia files. The authors then apply this linear programming-based algorithm to evaluate the performance of content downloading by analyzing the impact of different system settings, such as the penetration rate of vehicular communication technology.

In this work, we use linear programming and focus on the multi-commodity flow problem applied to the time-expanded graph. We apply the column generation technique as described by Ahuja et al. [12]. Since the LP arc flow formulation can become very large due to the amount of time-expanded nodes and edges, we use an equivalent path flow formulation, which has a smaller number of constraints but a much larger number of variables, since there is one variable for each possible path in the network between the pair of source and destination nodes of each message. Using column generation, however, we are able to solve the problem without having to explicit list all the variables. Instead, an iterative solving of subproblems takes place, each of which contains a solution that is closer to the original problem's optimal solution than the that of the subproblem from the previous iteration. At each iteration, an optimality check is performed which generates a new variable to consider in the following subproblem in order to improve the solution if the current one is non optimal. The Dijkstra's shortest path algorithm is the relevant optimality check.

III. Problem Model

In this Section, we give the problem model as described by Jain et al. [7]. Then we discuss some of its characteristics and the motivation and insights for our alternative formulation, which is also presented in this Section.

A. Original Linear Programming Formulation

Alonso, Jain, Fall and Patra [6], [7] proposed a linear programming formulation that assigns messages to edges and times in a DTN multigraph $G(V, E)$ in such a way to minimize the average message delivery delay. For ease of reference, since their model is the result of a combination of two papers, and to establish the nomenclature used in this work, we provide the formulation as described by Jain et al.

Multiple edges may exist between vertices because multiple options may exist in the network to transfer data between nodes. The variables in this formulation are defined in terms of the set of messages, the set of edges and the set of disjoint time intervals that partition the entire observed time period. The considered set of time intervals $\mathcal{I}$ is determined as described by Alonso et al. [6]. For an explanation about the objective function and linear constraints, we refer the reader to the original articles.

Input data: Values of the following sets and functions are retrieved from each network definition and are considered input to the algorithm.

- V, the set of network nodes.
- E, the set of network edges.
- $\mathcal{I}$, the set of time intervals, obtained as described by Alonso et al. [6]. Each time interval is defined in terms of its borders (i.e. $I_q \in \mathcal{I} \implies I_q = [t_{q-1}, t_q)$).
- $c : E \times \mathbb{R}^+ \to \mathbb{R}^+$, where $c_{e,t}$ is the capacity of edge $e \in E$ and time $t \in \mathbb{R}$.
- $d : E \times \mathbb{R}^+ \to \mathbb{R}^+$, where $d_{e,t}$ is the delay of edge $e \in E$ and time $t \in \mathbb{R}$.
- b_v is the storage capacity of the node v.
- I^v is the set $\{(w, v) \in E\}$ of edges whose destination node is v.
- O^v is the set $\{(v, w) \in E\}$ of edges whose source node is v.
- K is the set of messages.
- $\omega(k)$, $s(k)$, $d(k)$, $m(k)$, for $k \in K$, are, respectively, the injection time, source node, destination node and size of message k.

Variables: The following are the variables used in the linear programming formulation.

- $N^k_{v,t}$ is the size of the portion of message k occupying buffer at node v at time $t \in \mathcal{I}$.
- $X^k_{e,I}$ is the size of the portion of message k transmitted (at the source of the edge) over edge e during the interval $I \in \mathcal{I}$.
- $R^k_{e,I}$ is the size of the portion of message k received (at the destination of the edge) over edge e during the interval $I \in \mathcal{I}$.

Objective function: The goal is to minimize the average delivery delay, which is equivalent to minimizing the sum of the delays for all messages:

$$\text{Min} \sum_{v \in V} \sum_{k \in K} \sum_{I_q \in \mathcal{I}} (t_{q-1} - \omega(k))\Big(\sum_{e \in I^v} R^k_{e,I_q} - \sum_{e \in O^v} X^k_{e,I_q}\Big) \quad (1)$$

LP constraints:

$$\sum_{e \in I^v} R^k_{e,I_q} - \sum_{e \in O^v} X^k_{e,I_q} = \begin{cases} N^k_{v,t_q} - N^k_{v,t_{q-1}} + m(k) & \text{if } s(k) = v,\ \omega(k) = t_q \\ N^k_{v,t_q} - N^k_{v,t_{q-1}} & \text{otherwise} \end{cases} \quad k, v, I_q \quad (2)$$

$$R^k_{e,I_q \oplus d_{e,t_{q-1}}} = X^k_{e,I_q} \quad k, e, I_q \quad (3)$$

$$\sum_{k \in K} N^k_{v,t_{q-1}} \leq b_v \quad v, I_q \quad (4)$$

$$\sum_{k \in K} X^k_{e,I_q} \leq c_{e,t_{q-1}} \cdot |I_q| \quad e, I_q \quad (5)$$

$$N^k_{v,t_0} = \begin{cases} m(k) & \text{if } v = s(k),\ t_0 = \omega(k) \\ 0 & \text{otherwise} \end{cases} \quad k, v \quad (6)$$

$$N^k_{v,t_h} = \begin{cases} m(k) & \text{if } v = d(k) \\ 0 & \text{otherwise} \end{cases} \quad k, v \quad (7)$$

In these constraints, $\oplus$ represents the *shifting* operation. I.e. $I_q \oplus s = I_p \iff [t_{p-1}, t_p) = [t_{q-1} + s, t_q + s)$. The set of constraints (3) indicates that every part of message k transmitted at the beginning of edge e during time interval I_q should be received at the end of the same edge after the delay imposed by e at time t_{q-1}. For certain feasible problems, however, they may yield an infeasible formulation. Since the delay of an edge may change over time, it is possible that there are intervals $I_p, I_q \in \mathcal{I}$ such that $I_p \oplus d_{e,t_{p-1}} = I_q \oplus d_{e,t_{q-1}}$. In this case, if $t_q > t_p$ and $d_{e,t_{q-1}} = 0$ and $d_{e,t_{p-1}} > 0$, the formulation becomes infeasible since $R^k_{e,I_q \oplus d_{e,t_{q-1}}}$ is constrained to have two potentially different values. Because of this, in our implementation, we substitute (3) by the following without hurting the original intent:

$$R^k_{e,I_p} = \sum_{I_q \in S_{e,p}} X^k_{e,I_q}$$

$$\text{where} \quad S_{e,p} = \{I_q \in \mathcal{I} : I_q \oplus d_{e,t_{q-1}} = I_p\} \quad k, e, I_p \qquad (8)$$

We make another modification inspired by the formulation presented in the previous work by Alonso et al. [6] (equations numbered (6) in that paper, page 5). We replace equations (2) by the following.

$$\sum_{e \in I^v} R^k_{e,I_q} - \sum_{e \in O^v} X^k_{e,I_q} =$$

$$\begin{cases} N^k_{v,t_q} - N^k_{v,t_{q-1}} - m(k) & \text{if } s(k) = v,\ \omega(k) = t_{q-1} \\ N^k_{v,t_q} - N^k_{v,t_{q-1}} + m(k) & \text{if } d(k) = v,\ t_q = t_h \\ N^k_{v,t_q} - N^k_{v,t_{q-1}} & \text{otherwise} \end{cases} \qquad (9)$$

These equations represent the fact that, if v is the source of message k, which is generated at the moment t_{q-1}, then the net flow of node v (i.e. $\sum_{e \in O^v} X^k_{e,I_q} + N^k_{v,t_q} - (\sum_{e \in I^v} R^k_{e,I_q} + N^k_{v,t_{q-1}})$) is the size of message k. In other words, all the flow for k that is transmitted away ($\sum_{e \in O^v} X^k_{e,I_q}$) by v plus the flow that remains stored (N^k_{v,t_q}) in v minus what is received ($\sum_{e \in I^v} R^k_{e,I_q}$) by v minus what was previously stored ($N^k_{v,t_{q-1}}$) in v is the size of message k, which means that $m(k)$ units of flow are introduced in the system. If v is the destination of k, then the net flow of v for message k should be the negative of the size of k at t_h (the end of the simulation), which means the message is consumed by v (removed from the system). If v is not destination nor source for k, then its net flow for k should be zero.

By creating an extra time interval in $\mathcal{I}$ such that $\forall k \in K$ $\omega(k) > t_0$, we can drop equations (6). Equations (7) can also be dropped in light of the new set of equations (9).

B. Time Expanded Graph

The main difference between the formulation from Section III-A and that of the standard min-cost multicommodity flow is the dependency on time intervals. The classic network flows book by Ford and Fulkerson [9] describes a way to model the time factor without creating different types of variables other than those that account for the amount of flow transmitted at each edge of the network graph. Instead, if $T = |\mathcal{I}|$, the original network graph $G = G(V, E)$ is expanded into $G(T) = G(V(T), E(T))$ so that each vertex and edge has a copy for each time point. This way, our dynamic problem is reduced to a regular (static) min-cost multicommodity flow on the time-expanded graph $G(T)$.

Consider the example from Figure 1. Each edge in this example is annotated with its capacity c and delay d in the format (c, d). Edge $(0, 1)$, for instance, has capacity 2 and delay 1. Suppose that this network is observed during a time period composed of four time intervals, namely $\{[0, 1), [1, 2), [2, 3), [3, 4)\}$. We want to represent the fact that the flow transmitted at the source of an edge in this network arrives at its destination after the edge's delay. Also, we need a way to allow some flow to be stored in nodes for some time period until a good opportunity for transmission arises.

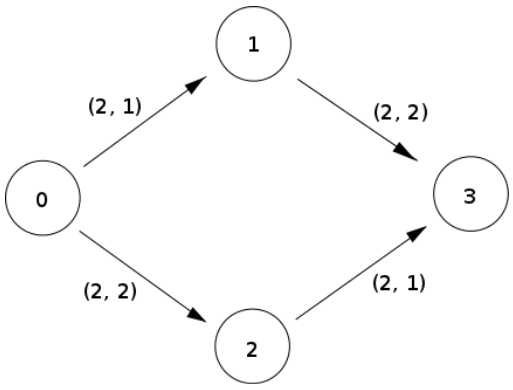

Fig. 1. Network example

The time-expanded network for the example in Figure 1 is displayed in Figure 2. Each of the four nodes has four copies, one for each time point. Each edge also has copies for time points. The edge $(0, 2)$, for example, has two copies: one that connects node 0 at time 0 with node 2 at time 2 (since its delay is 2) and another that connects node 0 at time 1 with node 2 at time 3. An edge from the original network has a copy for each time point except those that would mean that some flow would arrive at its destination after the last time period. For example, units of flow can only be transmitted from node 0 to node 2 at time 0 or 1 through edge $(0, 2)$. If it was transmitted at time 2, for example, it would arrive at its destination after a delay of 2, which means time 4, which is outside our set of observed time intervals.

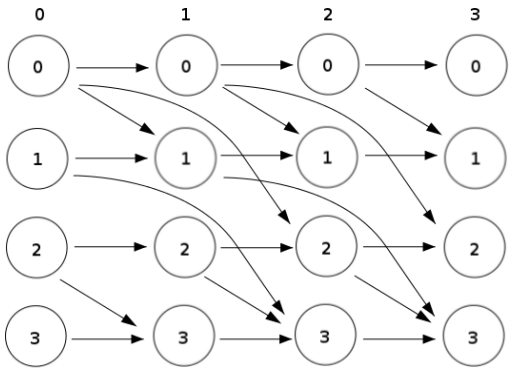

Fig. 2. Time-expanded network example

Another important characteristic of this time-expanded graph are the edges between pairs of nodes with the same label.

Each copy of node 0 is connected by an edge with another of its copies associated with the next time point. These *hold-over* edges represent buffer storage during one time interval. If some flow units pass through the edge that connects node 0 at time 0 and node 0 at time 1, that means that these units were stored in node 0 during the interval $[0, 1)$.

If we solve a standard (static) multi-commodity flow problem in the time-expanded graph, we can interpret its result in terms of the dynamic properties of the original network.

$G(T)$ is constructed as follows. Assume that $I_{\mathcal{I}}$ is the set of time-points formed by the extreme points of the time intervals in $\mathcal{I}$. Also, for $p = 0, 1, ..., T$, t_p is the value of the $(p+1)$-th time-point in $I_{\mathcal{I}}$, $i(t) \in \{0, 1, ..., T\}$ is the index of time-point $t \in I_{\mathcal{I}}$, $d_{e,p} := d_{e,t_p}$ and $c_{e,p} := c_{e,t_p}$.

Each vertex $v \in V$ has a copy $v(p) \in V(T)$ for $p \in \{0, 1, ..., T\}$. Each edge $e = (v, w) \in E$ has a copy $e(p) = (v(p), w(i(t_p + d_{e,p}))) \in E(T)$ for $p \in \{0, 1, ...i(t_T - d_{e,p})\}$. Also, for each vertex $v \in V$, we create *hold-over* edges $(v(p), v(p + 1)) \in E(T)$ for $p \in \{0, 1, ..., T - 1\}$. When some flow passes through one of these hold-over edges, it means that data is stored in v's buffer for one time-step. The capacity of edge $e(p) = (v(p), w(i(t_p + d_{e,p}))) \in E(T)$ is defined as $c(e(p)) = c_{e,p}$. Similarly, the delay of that edge is $d(e(p)) = d_{e,p}$. The capacity of the hold-over edge $(v(p), v(p + 1)) \in E(T)$ is b_v, which is the buffer storage capacity of node $v \in V$. The delay of the same edge is 0.

C. Alternative Formulation

Using a time-expanded graph as in Section III-B, we can see that the following formulation is equivalent to the one described in Section III-A.

$$\text{Min} \sum_{k \in K} dx^k$$

$$\sum_{k \in K} x_e^k \leq c(e) \qquad \forall e \in E(T) \qquad (10)$$

$$\mathcal{N} x^k = a^k \qquad \forall k \in K \qquad (11)$$

$$0 \leq x_e^k \leq c(e) \qquad \forall e \in E(T), \forall k \in K$$

Here, x^k is an array with dimension $|E(T)|$ such that x_e^k is the size of the portion of message $k \in K$ transmitted through edge $e \in E(T)$. d is also an array with dimension $|E(T)|$ such that d_e is the delay of edge $e \in E(T)$. $\mathcal{N}$ is the incidence matrix of $G(T)$, such that its dimension is $|V(T)| \times |E(T)|$ and:

$$\mathcal{N}_{v,e} = \begin{cases} +1 & \text{if } e = (v, w) \text{ for some } w \in V(T) \\ -1 & \text{if } e = (w, v) \text{ for some } w \in V(T) \\ 0 & \text{otherwise} \end{cases}$$

The array a^k has dimension $|V(T)|$ and is defined as follows.

$$a^k_{v(t_q)} = \begin{cases} m(k) & \text{if } v = s(k) \text{ and } t_q = \omega(k) \\ -m(k) & \text{if } v = d(k) \text{ and } t_q = t_h \\ 0 & \text{otherwise} \end{cases} \qquad (12)$$

For simplicity, define, for each $t \in I_{\mathcal{I}}$ and each $e = (v, w) \in E$ of the original graph G, $e(t) := e(i(t)) \in E(T)$ and $v(t) := v(i(t)) \in V(T)$. This way, $e(t) = (v(t), w(t + d_{e,t}))$. To see that this formulation is equivalent to the one described in Section III-A, we can interpret the values of x^k as following equalities.

$$X^k_{e,I_q} = x^k_{e(t_{q-1})} \qquad \forall I_q \in \mathcal{I} \qquad (13)$$

$$R^k_{e,I_q} = \sum_{t_p \in S_{e,q-1}} x^k_{e(t_p)} \qquad \forall I_q \in \mathcal{I} \qquad (14)$$

$$\text{where } S_{e,q} = \{t_p \in I_{\mathcal{I}} : t_p + d_{e,t_p} = t_q\}$$

$$N^k_{v,t_q} = x^k_{(v(t_{q-1}),v(t_q))} \qquad \forall q \in \{1, ..., T\} \qquad (15)$$

The first equality follows directly from the definition of the variables X^k_{e,I_q} and x^k. The second equality follows from the first equality and the constraint set (8). The last equality follows from the definition of the hold-over edges $(v(p), v(p+1))$ in $G(T)$. We also have, for $i = v(t_{q-1})$:

$$\sum_{\{j:(j,i) \in E(T)\}} x^k_{(j,i)} = \sum_{e \in I^v} R^k_{e,I_q} + N^k_{v,t_{q-1}} \qquad (16)$$

$$\sum_{\{j:(i,j) \in E(T)\}} x^k_{(i,j)} = \sum_{e \in O^v} X^k_{e,I_q} + N^k_{v,t_q} \qquad (17)$$

The right side of equation (16) accounts for the sum of flows of all edges that arrive at node $v \in V$ of the original graph at time t_{q-1} plus the flow that stays stored in v during $[t_{q-2}, t_{q-1})$. This is equal to the sum of flows that arrive at $v(t_{q-1})$ in the expanded graph. Analogously, the right side of equation (17) accounts for the sum of flows of all edges that leave node v at time t_{q-1} plus the flow that stays stored in v during $[t_{q-1}, t_q)$, which is equal to the sum of flows in the expanded graph on edges that leave node $v(t_{q-1})$. From this, we have, for $i = v(t_{q-1})$:

$$\begin{aligned} -a^k_i &\overset{(11)}{=} \sum_{\{j:(j,i) \in E(T)\}} x^k_{(j,i)} - \sum_{\{j:(i,j) \in E(T)\}} x^k_{(i,j)} \\ &\overset{(16),\,(17)}{=} \sum_{e \in I^v} R^k_{e,I_q} - \sum_{e \in O^v} X^k_{e,I_q} + N^k_{v,t_{q-1}} - N^k_{v,t_q} \\ &\overset{(12)}{=} \begin{cases} -m(k) & \text{if } v = s(k) \text{ and } t_{q-1} = \omega(k) \\ m(k) & \text{if } v = d(k) \text{ and } t_q = t_h \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

This shows that equations (11) are equivalent to equations (9). The effect of equations (8) are captured by how edges are constructed in the time-expanded graph ($e(p) = (v(p), w(t_p + d_{e,p})) \in E(T)$ for $p \in \{0, ..., i(t_T - d_{e,p})\}$ and $e = (v, w) \in E$). This implies that the flow transmitted at the source of an edge arrives at the edge's destination after $d_{e,p}$. Equations (10) and the construction of the capacities of hold-over edges capture equations (4) and (5) with the remark that, by construction, all intervals $I_q \in \mathcal{I}$ have the same length, which means that capacities can be redefined to drop the $|I_q|$ factor from (5).

In the original formulation, each R^k_{e,I_q} variable has cost coefficient $t_{q-1} - \omega(k)$. Each X^k_{e,I_q} has cost coefficient $-(t_{q-1} - \omega(k))$. To determine the cost coefficients of the new formulation, we observe, from equations (13) and (14) that each variable $x^k_{e(t_{q-1})}$ associated with a non-hold-over

edge $e \in E(T)$ in the new formulation should appear in the objective function with two coefficients:

$$(t_{q-1} - \omega(k))X^k_{e,I_q} \overset{(13)}{=} (t_{q-1} - \omega(k))x^k_{e(t_{q-1})}$$
$$(t_{p-1} - \omega(k))R^k_{e,I_p} \overset{(14)}{=} \sum_{t_{q-1} \in S_{e,p-1}} (t_{p-1} - \omega(k))x^k_{e(t_{q-1})}$$
$$\text{where } S_{e,p} = \{t_q \in I_{\mathcal{I}} : t_q + d_{e,t_q} = t_p\}$$

Since $t_{p-1} = t_{q-1} + d_{e,t_{q-1}}$ in the above, the cost coefficient of $x^k_{e(t_{q-1})}$ for $e \in E$ should be the difference

$$\begin{aligned}&(t_{p-1} - \omega(k)) - (t_{q-1} - \omega(k)) = \\ &(t_{q-1} + d_{e,t_{q-1}} - \omega(k)) - (t_{q-1} - \omega(k)) = \\ &t_{q-1} - t_{q-1} + d_{e,t_{q-1}} - \omega(k) + \omega(k) = \\ &d_{e,t_{q-1}}\end{aligned}$$

Since $d_{e,t_{q-1}} = d(e(t_{q-1}))$ from the definition of edge delays in the time-expanded graph, and the fact that both the N^k_{v,t_q} variables in the original formulation and the delays of the hold-over edges in the time-expanded graph are equal to 0, the edge delays in the time-expanded graph are appropriate cost coefficients in the new formulation.

D. Problem Sizes

We can compare the sizes of the two problems in terms of the number of variables and the number of linear constraints. Since the formulation from Section III-B has one variable for each message and for each edge in the time-expanded graph, we need to determine how this number compares to the number of variables in the original formulation.

There are two kinds of edges $e \in E(T)$. Those created from edges in the original graph (not time-expanded) and hold-over edges. From the construction of the former type of edges, there are at most $|E| \cdot (|\mathcal{I}| + 1)$ of them, namely, $(v(t_p), w(t_p + d_{e,p}))$, $\forall p \in \{0, ..., T\}$, $\forall e = (v, w) \in E$. From the construction of hold-over edges, there are $|V| \cdot |\mathcal{I}|$ of them, which are $(v(p), v(p+1))$, $\forall p \in \{0, ..., T-1\}$, $\forall v \in V$. In total, $|E(T)| \leq |E| + |\mathcal{I}| \cdot (|E| + |V|)$. Consequently, the alternative formulation has $|K| \cdot |E(T)| \leq |K| \cdot |E| + |K| \cdot |\mathcal{I}| \cdot (|E| + |V|)$ variables. The original formulation has three kinds of variables: $|K| \cdot |E| \cdot |\mathcal{I}|$ variables R^k_{e,I_q}, $|K| \cdot |E| \cdot |\mathcal{I}|$ variables of type X^k_{e,I_q} and $|K| \cdot |V| \cdot (|\mathcal{I}|+1)$ variables of type N^k_{v,t_q}. This sums up to $2 \cdot |K| \cdot |E| \cdot |\mathcal{I}| + |K| \cdot |V| \cdot (|\mathcal{I}|+1)$. The difference between the number of variables in the alternative and the original formulations is

$$\begin{aligned}&|K||E| + |K||\mathcal{I}||E| + |K||\mathcal{I}||V| \\ &- (|K||V| + 2|K||\mathcal{I}||E| + |K||\mathcal{I}||V|) \\ &= |K||E| - |K||V| - |K||\mathcal{I}||E| \leq 0\end{aligned}$$

The number of constraints in the original formulation is:

$$\begin{aligned}&|K| \cdot |V| \cdot |\mathcal{I}| && \text{for constraints (2)} \\ &+ |K| \cdot |E| \cdot |\mathcal{I}| && \text{for constraints (3)} \\ &+ |V| \cdot |\mathcal{I}| && \text{for constraints (4)} \\ &+ |E| \cdot |\mathcal{I}| && \text{for constraints (5)} \\ &+ |K| \cdot |V| && \text{for constraints (6)} \\ &+ |K| \cdot |V| && \text{for constraints (7)} \\ &= |\mathcal{I}| \cdot (|K| + 1) \cdot (|V| + |E|) + 2 \cdot |V| \cdot |K|\end{aligned}$$

The difference between $|K| \cdot |V(T)| + |E(T)| = |K| \cdot |V| \cdot (|\mathcal{I}| + 1) + |E| + |\mathcal{I}| \cdot (|E| + |V|)$, which is the number of constraints in the alternative formulation, and the number of constraints in the original formulation is

$$\begin{aligned}&|K||V|(|\mathcal{I}| + 1) + |E| + |\mathcal{I}|(|E| + |V|) \\ &- |\mathcal{I}|(|K| + 1)(|V| + |E|) + 2|V||K| \\ &= |E| - |K||\mathcal{I}||E| - |K||V| \leq 0\end{aligned}$$

The conclusion is that the alternative formulation is no larger than the original formulation. Moreover, since $|\mathcal{I}|$ may be exponentially large, the alternative formulation may be significantly smaller than the original, although both problems have sizes of the same order of magnitude.

E. Column Generation

The arc flow formulation described in Section III-C can be represented as the path flow formulation (a well known fact from the literature).

$$\text{Min} \sum_{k \in K} \sum_{P \in \mathbb{P}^k} d(P)f(P)$$

$$\sum_{k \in K} \sum_{P \in \mathbb{P}^k} \delta_e(P)f(P) \leq c(e) \qquad \forall e \in E(T) \qquad (18)$$
$$\sum_{P \in \mathbb{P}^k} f(P) = m(k) \qquad \forall k \in K \qquad (19)$$
$$f(P) \geq 0 \qquad \forall k \in K, P \in \mathbb{P}^k$$

In this formulation, $\mathbb{P}^k$ is the set of all paths between the source and destination of message $k \in K$. If $v \in V$ is message k's source in the original graph G, $w \in V$ is its destination and $t \in I_{\mathcal{I}}$ is its injection time, we say that $v(t) \in V(T)$ is its source in the time-expanded graph and $w(t_h) \in V(T)$ is its destination. $f(P)$ is the variable in this formulation and represents the amount of flow passing through path $P \in \mathbb{P}^k$ for all $k \in K$. $d(P)$ represents the delay of path P and is equal to $\sum_{e \in P} d(e)$. $\delta_e(P)$ is an indicator whose value is 1 if $e \in P$ and 0 otherwise.

This path flow formulation has fewer constraints than the arc flow formulation. Specifically, the arc flow formulation has $|E(T)| + |V(T)| \cdot |K|$ constraints while the path flow formulation has only $|E(T)| + |K|$. On the other hand, the number of variables in much larger in the path flow formulation, since there is one for each possible path between the pair of nodes associated with each message. However, only a small number of these variables carries flow in the optimal solution [12]. This can be exploited by the *column generation* technique, which allows us to never explicitly list all the variables in the formulation.

Column generation is an algorithmic technique for solving linear programs with an exponentially large set of variables which takes its roots in the duality theory [13]. Each linear program, denoted *primal* in this context, has an associated and unique *dual* program. For each constraint of the primal, there is a dual variable that is defined. These variables are related to the slack variables which arise in the simplex solving method, even though this is more general. Similarly, for each variable

of the primal, there is a constraint in the dual, which binds the dual variables related to the primal constraints in which the concerned primal variable appears. This is done in a way that the duality association is reflexive (the dual of the dual of a LP is the original LP).

In our case, the dual of the linear program (18)-(19) is

$$\text{Max} \sum_{e \in E(T)} c(e)w(e) - \sum_{k \in K} m(k)\sigma_k$$

$$\sum_{e \in P} w_e + \sigma_k \leq d(P) \qquad \forall P \in \mathbb{P}^k, \forall k \in K \qquad (20)$$

$$\sigma_k, w_e \geq 0 \qquad \forall k \in K, \forall e \in E(T)$$

Each instantiation of the primal variables is similarly associated to an instantiation of the dual variables such that the primal values represent a sub-optimal feasible solution if and only if the dual values is a non feasible solution, i.e. at least one constraint of the dual is violated. Both set of primal and dual values represent a feasible solution if and only if there are both optimal (with the property that the primal and dual optimal objectives values are the same).

Exploiting this property, the column generation principle is to first solve the primal on a restricted set of variables (also called columns, hence the column generation), considering that the variables outside this set are zero. In our case, it corresponds to considering a restricted set of paths P_0. Solving of the primal on this restricted set of variable is thus fast and, if there exists a feasible solution f_0, it is related to a set of dual values $\mathcal{D}_0 = (w_0, \sigma_0)$. If f_0 is suboptimal, the aforementioned property of the duality claims that $\mathcal{D}_0$ is a non feasible solution of the dual. There is then at least one constraint that is violated and which is in bijection with a variable of the primal, hence a path p_0. The separation theorem claims that solving again the primal on the set of paths $P_1 = P_0 \cup \{p_0\}$ will improve the solution [13]. The process loops until no such path exists. When reaching this state, it means that the dual variables represent a feasible solution. Since the primal does too, duality theory claims that both the primal and the dual are optimal.

Going back to our problem, deciding if the solution is optimal consists in finding a $k \in K$ and a path $P \in \mathbb{P}^k$ such that (20) is violated, i.e. $\sum_{e \in P} w_e + \sigma_k \geq d(P)$ which can be rewritten as

$$\sum_{e \in P} (d(e) - w_e) \geq \sigma_k \qquad (21)$$

We interpret $\sum_{e \in P}(d(e) - w_e)$ as the cost of path P in terms of the modified cost $d(e) - w_e$ of each of its edges $e \in P$. Then we check whether condition (21) is satisfied for all paths not in the basis by identifying, among all paths $P \in \mathbb{P}^k$, one with the minimum modified cost and verifying whether its modified cost is greater than or equal to σ_k. If it is, the algorithm can stop since it has found an optimal solution. If not, this path can enter the primal set of variables while the value of the objective function decreases. In this case, a new iteration can take place with this new set and so on. To identify such a path we use Dijkstra's minimum cost path algorithm on the time-expanded graph once for each message $k \in K$ considering $d(e) - w_e$ as edge costs.

The execution time of a column generation approach depends on the number of generated columns, which is different for each input. Since not all columns are listed by this approach, however, the reduction in the number of constraints in comparison with the original formulation has dramatic impact on the solving time. While the arc flow formulation has $|K| \cdot |V(T)| + |E(T)|$ constraints, the path flow formulation has only $|E(T)| + |K|$ constraints.

IV. Experiments

An implementation of the original linear program presented by Jain et al. [7] with the modifications from Section III-A was created and used to solve three instances of simple routing problems. The same three instances were also used as input to an implementation of the column generation approach with the objective to compare execution times and problem sizes in terms of number of variables and number of constraints. Both implementations are based on IBM ILOG Optimization Suite's CPLEX callable library.

The first network instance is inspired by the Remote Village example described by Jain et al. [7]. In this example, a city with constant Internet access communicates with a remote rural village through three different channels. A store-and-forward Low-Earth Orbit satellite becomes in range of the city and the village for a number of time periods per day offering connection between them. A motorcycle courier visits the remote village with a storage device such as a flash drive and collects and deploys network data to and from the village during his visit. Then he travels to the city and forwards the data and collects responses. These trips are also made a number of times per day. The other routing option is a dial-up connection which is available during one time period per day. The observation period is 48 hours.

Since some of the delays in these connection options are in the order of seconds, the number of time intervals necessary for a 48 hours period is large. In this instance, the time discretization step described by Alonso et al. [6] resulted in 187,802 time intervals. Considering only two messages being routed, the LP size and solving time on a computer with 24 Intel®Xeon®processors at 2.4 GHz and 128 GB of RAM can be found in the *Scenario 1* column of Table I.

The same network instance was used as input to the column generation implementation. As expected, it created the same number of time interval, since both implementations share the same time-discretization step. The path flow problem size with two messages and its solving time in the same computer can be found in the *Scenario 1* column of Table II.

Another simple network example used as input for the experiments is an adapted version of the instance borrowed from Ahuja et al. [12]. It is a simple network with 6 nodes, 7 edges and 2 messages. Such a simple example took almost no time to execute in linear programming implementations when a small observation period of 40 time units was used. When we used a larger period of 200,000 time units to represent the duration of contacts between nodes, we could see significant differences between the implementations. Columns *Scenario 2* of Tables I and II show problem sizes and execution times of the application of this network as input for the original linear program and the column generation, respectively.

TABLE I. PROBLEM SIZES AND SOLVING TIME OF THE ORIGINAL FORMULATION INSTANCES REPRESENTING EACH OF THE THREE DESCRIBED NETWORKS.

	Scenario 1	Scenario 2	Scenario 3
Constraints	6 M	7.8 M	11.5 M
Variables	5.63 M	8 M	11 M
Solving Time	24m4s	30s	110m

TABLE II. PROBLEM SIZES OF PATH FLOW FORMULATIONS REPRESENTING EACH OF THE THREE DESCRIBED NETWORKS AND THEIR COLUMN GENERATION SOLVING TIMES.

	Scenario 1	Scenario 2	Scenario 3
Constraints	2.2 M	1.4 M	5.7 M
Generated Paths	2	5	1
Solving Time	4.4s	3.6s	32s

An artificial network example was created with 70 nodes. Between each pair of nodes, a directed edge was created with 50% probability, which resulted in 1035 edges. Each edge was assumed to be active during 1 hour and 30 minutes starting at time 0 with uniformly random delays between 1 and 5 seconds. The original linear program took 1.5 million iterations to produce an optimal routing of only one message. Its problem size and execution time can be found in column *Scenario 3* of Table I. The column with the same name in Table II shows this network's path flow problem size and column generation execution time.

As can be seen in Tables I and II, the column generation problem sizes have much smaller number of linear constraints than the original formulation. Furthermore, execution time of the column generation implementation can be three orders of magnitude faster in some cases.

V. CONCLUSION

We proposed a linear programming arc flow formulation to routing in Delay Tolerant Networks that minimizes average message delay, which is based on the linear program presented by Jain et al [7]. Our usage of a time-expanded graph allows us to apply standard multi-commodity minimum cost static flow theory and interpret the results in terms of the dynamic properties of the network used to build the graph. We compared the size of the problem in the original formulation with that of the arc flow formulation and concluded that the latter is no larger than the former in terms of number of constraints and number of variables.

We applied the column generation technique on the path flow representation of our arc flow formulation. The advantage of a path flow representation is a much smaller number of LP constraints. On the other hand, it has an exponencial number of variables, each of which represents one of the possible paths in the network between a message source and its destination. However, column generation allows us to solve the problem on a restricted set of such variables. Since all the variables are never explicitly listed by the algorithm, an optimal solution is typically found much faster than solving the original arc flow formulation.

Simulation on example network instances show that the difference between execution times of the column generation implementation and the implementation of the original formulation presented by Jain et al. [7] can reach up to three orders of magnitude using the IBM ILOG CPLEX callable library. We also observed that the path flow formulation used by column generation produced a much smaller number of constraints than the original formulation. Exact solutions that cannot feasibly be identified in some network instances using previous formulations are now possible.

For future work, we plan to compare the optimal flows identified by our linear programming approach with the output from the very fast routing method based on evolving graphs [4], which may not give optimal routings in the presence of flows. We would also like to expand our experiments to validate our method using real mobility traces.

REFERENCES

[1] K. Fall, "A delay-tolerant network architecture for challenged internets," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM '03. New York, NY, USA: ACM, 2003, pp. 27–34. [Online]. Available: http://doi.acm.org/10.1145/863955.863960

[2] N. Ristanovic, G. Theodorakopoulos, and J.-Y. L. Boudec, "Traps and pitfalls of using contact traces in performance studies of opportunistic networks," in *Proceedings of the 2012 IEEE INFOCOM*, 2012, pp. 1377–1385.

[3] M. Karaliopoulos and C. Rohner, "Trace-based performance analysis of opportunistic forwarding under imperfect node cooperation," in *Proceedings of the 2012 IEEE INFOCOM*, march 2012, pp. 2651 – 2655.

[4] A. Ferreira, A. Goldman, and J. Monteiro, "Performance evaluation of routing protocols for manets with known connectivity patterns using evolving graphs," *Wireless Networks*, vol. 16, no. 3, pp. 627–640, Apr. 2010. [Online]. Available: http://dx.doi.org/10.1007/s11276-008-0158-6

[5] A. Goldman, P. Floriano, and A. Ferreira, "A tool for obtaining information on DTN traces," in *4th Extreme Conference on Communication (ExtremeCom 2012)*, Zurich, Suisse, 2012, p. 6. [Online]. Available: http://hal.inria.fr/hal-00742993

[6] J. Alonso and K. Fall, "A linear programming formulation of flows over time with piecewise constant capacity and transit times," *IRB-TR-03*, vol. 7, 2003.

[7] S. Jain, K. Fall, and R. Patra, "Routing in a delay tolerant network," *SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 145–158, Aug. 2004. [Online]. Available: http://doi.acm.org/10.1145/1030194.1015484

[8] G. Amantea and A. Goldman, "Review and analysis of citations," in *Proceedings of 8th Experimental Software Engineering Latin American Workshop*, 2011, p. 9.

[9] D. R. Ford and D. R. Fulkerson, *Flows in Networks*. Princeton, NJ, USA: Princeton University Press, 2010. [Online]. Available: http://portal.acm.org/citation.cfm?id=1942094

[10] D. Hay and P. Giaccone, "Optimal routing and scheduling for deterministic delay tolerant networks," in *Sixth International Conference on Wireless On-Demand Network Systems and Services, 2009. WONS 2009*, 2009, pp. 27–34.

[11] F. Malandrino, C. Casetti, C.-F. Chiasserini, and M. Fiore, "Optimal content downloading in vehicular networks," *IEEE Transactions on Mobile Computing*, vol. 12, no. 7, pp. 1377–1391, 2013.

[12] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: theory, algorithms, and applications*. Prentice hall, 1993.

[13] M. Grtschel, L. Lovsz, and A. Schrijver, "The ellipsoid method and its consequences in combinatorial optimization," *Combinatorica*, vol. 1, no. 2, pp. 169–197, 1981. [Online]. Available: http://dx.doi.org/10.1007/BF02579273