Survivability in Time-Varying Networks

Qingkai Liang and Eytan Modiano, *Fellow, IEEE*

Abstract—Time-varying graphs are a useful model for networks with dynamic connectivity such as vehicular networks, yet, despite their great modeling power, many important features of time-varying graphs are still poorly understood. In this paper, we study the survivability properties of time-varying networks against unpredictable interruptions. We first show that the traditional definition of survivability is not effective in time-varying networks, and propose a new survivability framework. To evaluate the survivability of time-varying networks under the new framework, we propose two metrics that are analogous to MaxFlow and MinCut in static networks. We show that some fundamental survivability-related results such as Menger's Theorem only conditionally hold in time-varying networks. Then, we analyze the complexity of computing the proposed metrics and develop approximation algorithms. Finally, we conduct trace-driven simulations to demonstrate the application of our survivability framework in the robust design of a real-world bus communication network.

Index Terms—Network robustness, mobile networks, time-varying networks

1 Introduction

Time-varying graphs have emerged as a useful model for networks with time-varying topology, especially in the context of communication networks. Examples include vehicular ad hoc networks [1], [2], space networks [3], [4], mobile sensor networks [5], [6], whitespace networks [7], [8] and millimeter-wave (mmWave) networks [9]. In Fig. 1, we illustrate a simple time-varying graph and its snapshots over 3 time slots.

In many applications of time-varying networks, transmission reliability is of a great concern. For example, it is critical to guarantee transmission reliability for vehicular networks that are often used to exchange traffic and emergency information. Unfortunately, time-varying networks are particularly vulnerable due to their constantly changing topology that results from different forms of interruptions. One type of interruptions are called *intrinsic* interruptions which originate from the inherent nature of the network, such as node mobility in vehicular networks. For certain types of networks, such intrinsic interruptions are often *predictable*. For example, it is easy to predict the temporal patterns of topology for a time-varying network formed by either public buses [1], [2] or satellites [3], [4] which have fixed tours and schedules; in low-duty-cycle sensor networks [10], [11], the sleep/wake-up pattern is periodic and can be predicted accurately; in whitespace networks [7], [8], the states of secondary links in the next few hours can be known a prior by using the whitespace database [12]; a recent study [13] also shows that human mobility has 93 percent potential predictability. In contrast, the other type of interruptions are *extrinsic* and *unpredictable*. For example, the predictions about the evolution of network topology are prone to errors and could be inaccurate due to various unforeseen factors such as unexpected obstacles (e.g., shadowing caused by humans, particularly in mmWave networks [9]), hardware malfunctions and natural disasters [23]. These unpredictable disruptions may greatly degrade network performance and are referred to as *failures*. The goal of this paper is to understand the robustness of time-varying networks against unpredictable interruptions (failures) while treating the predictable interruptions as an inherent feature of the network.

Due to the unpredictability of failures, it is desirable to evaluate the *worst-case survivability*. In static networks, this is usually defined to be the ability to survive a certain number of failures as measured by the mincut of the graph. However, this definition is not effective in time-varying networks. By its very nature, a time-varying network may have different topologies at different instants, so its connectivity or survivability must be measured over a long time interval. To be more specific, we would like to highlight two important temporal features that are neglected by the traditional notion of survivability.

First, failures have significantly different durations in a time-varying network. For example, an unexpected obstacle may only disable the link between two nodes for several seconds, after which the link reappears. In contrast, the traditional definition of survivability is intended for a static environment and fails to account for links reappearing. The duration of failures has a crucial impact on the performance of time-varying networks. For example, in the time-varying network shown in Fig. 1, a one-slot failure of any link cannot separate node A and node D while a two-slot failure (i.e., a failure that spans two consecutive slots) can disconnect D from A by disabling link $A \rightarrow B$ in the first two slots.

Second, failures may occur at different instants. This feature is obscured in static networks but has a great influence on time-varying networks due to their changing connectivity. For example, if a two-slot failure occurs to link $A \rightarrow B$ at the beginning of slot 2, node D is still reachable from node A within the three slots; however, if the two-slot failure happens at the beginning of slot 1, there is no way to travel from A to D within three slots.

• *The authors are with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139. E-mail: {qingkai, modiano}@mit.edu.*

Manuscript received 6 Apr. 2016; revised 10 Nov. 2016; accepted 19 Nov. 2016. Date of publication 6 Dec. 2016; date of current version 2 Aug. 2017. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TMC.2016.2636152

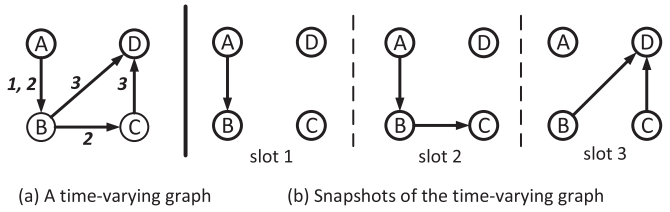


Fig. 1. (a) Original time-varying graph, where the numbers next to each edge indicate the slots when that edge is active. The traversal delay over each edge is one slot. (b) Snapshots of the time-varying graph.

To handle the above non-trivial temporal factors, we propose a new survivability framework for time-varying networks. Our framework captures both the *number* and the *duration* of failures. The contributions of this paper are as follows:

- *Model.* We propose a new survivability framework, i.e., (n, δ)-survivability, where the values of n and δ characterize the number and the duration of failures the network can tolerate. Moreover, by tuning the two parameters, our framework can generalize many existing survivability models. We further propose two metrics, namely MinCut_δ and $\mathsf{MaxFlow}_\delta$, in order to assess robustness in time-varying networks.
- *Theory.* We provide new graph-theoretic results that highlight the difference between static and time-varying graphs. For example, we show that some fundamental survivability-related results such as Menger's Theorem[1] only conditionally hold in time-varying graphs.
- *Computation.* Due to the difference between static and time-varying graphs, the evaluation of survivability becomes very challenging in time-varying networks. We analyze the complexity of computing the proposed survivability metrics and develop efficient approximation algorithms.
- *Application.* We conduct trace-driven simulations to demonstrate the application of our framework in a real-world communication network used in a public transportation system. It is shown that our survivability framework has strong modeling power and is more suitable for time-varying networks than existing approaches.

The remainder of this paper is organized as follows. In Section 2, we formalize the model of time-varying graphs. In Section 3, the new survivability framework and its associated metrics are introduced. In Section 4, we investigate computational issues in the proposed framework. In Section 5, trace-driven simulations are conducted to demonstrate the application of our framework in a bus communication network. Finally, related work and conclusions are given in Sections 6 and 7, respectively.

2 Model of Time-Varying Graphs

In this section, we formalize the model of time-varying graphs and introduce some important terminology and assumptions that will be frequently used throughout the paper. A useful tool for transforming time-varying graphs is also introduced.

2.1 Definitions and Assumptions

Time-varying graphs are a high-level abstraction for networks with time-varying connectivity. The formal definition, first proposed in [16], is as follows.

Definition 1 (Time-Varying Graph). *A time varying graph $\mathcal{G} = (G, \mathcal{T}, \rho, \zeta)$ has the following components:*

(i) *Underlying (static) digraph $G = (V, E)$;*
(ii) *Time span $\mathcal{T} \subseteq \mathbb{T}$, where $\mathbb{T}$ is the time domain;*
(iii) *Edge-presence function $\rho : E \times \mathcal{T} \mapsto \{0, 1\}$, indicating whether a given edge is active at a given instant;*
(iv) *Edge-delay function $\zeta : E \times \mathcal{T} \mapsto \mathbb{T}$, indicating the time spent on crossing a given edge at a given instant.*

This model can be naturally extended by adding a node-presence function and a node-delay function. However, it is trivial to transform node-related functions to edge-related functions by the technique called node splitting (see [19], Chapter 7.2); thus, it suffices to consider the above edge-version characterization.

In this paper, we consider a *discrete and finite* time span, i.e., $\mathcal{T} = \{1, 2, \ldots, T\}$, where T is a bounded integer indicating the time horizon of interests, measured in the number of slots. In practice, T may have different physical meanings. For instance, it may refer to the deadline of packets or delay tolerance in delay-tolerant networks; it may also correspond to the period of a network whose topology varies periodically (e.g., satellite networks with periodical orbits). The slot length of a time-varying graph is arbitrary as long as it can capture topology changes in sufficient granularity. Note that although we adopt a discrete-time model, it can also be used to represent most continuous-time systems by discretization. For example, if a link is active from time $t = 0$ to $t = 0.11s$ and then inactive from time $t = 0.11s$ to $t = 0.15s$, we can discretize the time with the slot length of $0.01s$, with the link being active for the first 11 slots and inactive for the remaining four slots.

Under the discrete-time model, the edge-delay function ζ can take values from $\mathbb{N} = \{0, 1, 2, \ldots\}$. Note that *zero* delay means that the time used for crossing an edge is negligible as compared to the slot length. Throughout the rest of this paper, we consider the case where edge delay is one slot, i.e., $\zeta(e, t) = 1$ for any $e \in E$ and $t \in \mathcal{T}$, however, it is trivial to extend the analysis to arbitrary traversal time. For example, we can replace a link with delay $M > 0$ slots by M links (each with one-slot delay) that are concatenated in series.

The edge-presence function ρ indicates the *predictable* topology changes in a time-varying network. Examples of such predictable topology changes include those in a space communication network with known orbits [3], [4], in a mobile social network consisting of students who share fixed class schedules [14], in a low-duty-cycle sensor network with periodic sleep/wake-up patterns [10], [11], in a whitespace network with planned channel reclamation [7], [8], in a mmWave network with scheduled beam steering [9], etc. In contrast, *unpredictable* topology changes (also referred to as *failures* in this paper) include those caused by unexpected shadowing, unscheduled channel reclamation,

1. In graph theory, Menger's Theorem is a special case of the max-flow-mincut theorem, which states that the maximum number of edge- or node-disjoint paths equals to the size of the minimum edge or node cut, respectively.

hardware malfunctions, etc. *Note that this model does not require perfect predictions of future topology changes since any prediction errors can be treated as failures.*

2.2 Terminology

Definition 2 (Contact). *There exists a contact from node u to node v in time slot t if $e = (u, v) \in E$ and $\rho(e, t) = 1$. This contact is denoted by (e, t) or (uv, t).*

Intuitively, a contact is a "temporal edge", indicating the activation of a certain edge in a certain time slot. In the example shown in Fig. 1, there exists a contact $(\mathsf{AB}, 1)$, showing that link $\mathsf{A} \rightarrow \mathsf{B}$ is active in slot 1.

Definition 3 (Journey [15]). *In a time-varying graph, a journey from node s to node d is a sequence of contacts: $(e_1, t_1) \rightarrow (e_2, t_2) \rightarrow \cdots \rightarrow (e_n, t_n)$ such that for any $i < n$*

(i) $\mathsf{start}(e_1) = s, \mathsf{end}(e_n) = d$;
(ii) $\mathsf{end}(e_i) = \mathsf{start}(e_{i+1})$;
(iii) $\rho(e_i, t_i) = 1$;
(iv) $t_{i+1} > t_i$ *and* $t_n \leq T$.

Intuitively, a journey is just a "time-respecting" path. Conditions (i)-(ii) mean that intermediate edges used by a journey are spatially connected. Condition (iii) requires that intermediate edges remain active when traversed. Condition (iv) indicates that the usage of intermediate edges must respect time and the journey should be completed before the time horizon T. For example, there exists a journey from A to D in Fig. 1: $(\mathsf{AB}, 1) \rightarrow (\mathsf{BC}, 2) \rightarrow (\mathsf{CD}, 3)$ when $T = 3$.

Definition 4 (Reachability). *Node d is reachable from node s if there is a journey from s to d.*

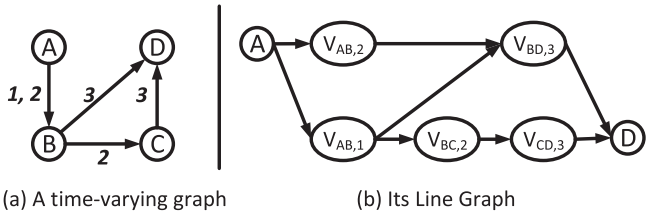
Intuitively, reachability can be regarded as "temporal connectivity" which indicates whether two nodes can communicate within T slots. For example, node D is reachable from node A in Fig. 1, meaning that a message from A can reach D within $T = 3$ slots.

2.3 A Useful Tool: Line Graph

A line graph is a useful tool which allows us to transform a time-varying graph into a static graph that preserves the original reachability information. Readers may temporarily skip the details and revisit this section when necessary.

The transformation uses a similar idea to the classical *Line Graph* [34] which illustrates the adjacency between edges. Here, we adapt the idea of Line Graphs to a time-varying setting by accounting for the temporal features of time-varying graphs. Given a time-varying graph $\mathcal{G}$ with source s and destination d, its Line Graph $L(\mathcal{G})$ is constructed as follows.

- For each contact (e, t) in the original time-varying graph $\mathcal{G}$, create a corresponding node in the Line Graph; the new node is denoted by $v_{e,t}$. In addition, create a node for the source s and a node for the destination d, respectively.
- Add a directed edge from node v_{e_1,t_1} to node v_{e_2,t_2} in the Line Graph if $(e_1, t_1) \rightarrow (e_2, t_2)$ is a feasible journey from $\mathsf{start}(e_1)$ to $\mathsf{end}(e_2)$. Also, add an edge from node s to node $v_{e,t}$ if $\mathsf{start}(e) = s$, and add an edge from node $v_{e,t}$ to node d if $\mathsf{end}(e) = d$.



(a) A time-varying graph (b) Its Line Graph

Fig. 2. Illustration of Line Graph (src: A, dst: D).

An example of the Line Graph is shown in Fig. 2. The Line Graph is useful in the sense that it preserves the information of every s-d journey in the original time-varying graph. In Fig. 2, we can observe the correspondence between journey $(\mathsf{AB}, 1) \rightarrow (\mathsf{BC}, 2) \rightarrow (\mathsf{CD}, 3)$ and path $\mathsf{A} \rightarrow \mathsf{V}_{\mathsf{AB},1} \rightarrow \mathsf{V}_{\mathsf{BC},2} \rightarrow \mathsf{V}_{\mathsf{CD},3} \rightarrow \mathsf{D}$. This is generalized in Observation 1 whose correctness is easy to verify.

Observation 1. *Every s-d journey in a time-varying graph has an one-to-one correspondence to some s-d path in its Line Graph.*

Finally, we estimate the computational complexity for constructing the Line Graph. Denote by $|C|$ the number of contacts in the time-varying graph. We first need to create a node in the Line Graph for each contact, which takes $O(|C|)$ computation. Next, we need to construct directed edges for the Line Graph. Since it only takes $O(1)$ time to determine whether there should be directed edges between a pair of nodes in the Line Graph, the edge construction procedure takes $O(|C|^2)$ computation. As a result, the overall computational complexity for constructing the Line Graph is $O(|C|^2)$.

3 Survivability Model and Metric

In this section, we begin to investigate the survivability properties of time-varying networks. Specifically, we are interested in their resilience against *unpredictable interruptions* (i.e., failures) such as unexpected shadowing, hardware malfunctions, etc.

We first develop a new survivability model for time-varying networks. Next, several metrics are introduced to evaluate survivability under the new model. Finally, we present some graph-theoretic results regarding these metrics, which highlights the key difference between time-varying and static networks. In particular, we will show that some fundamental survivability-related results in static networks, such as Menger's Theorem, only conditionally hold in time-varying networks. Such a difference makes it challenging to evaluate survivability in a time-varying network.

3.1 (n, δ)-Survivability

In static networks, the *worst-case* survivability is usually defined to be the ability to survive a certain number of failures wherever these failures occur. This definition is still feasible but very ineffective in time-varying networks because it fails to capture many temporal features of failures (e.g., duration and instant of occurrence). As discussed in the introduction, these temporal features have significant impacts on time-varying networks. Hence, we extend the survivability model in order to account for these temporal effects and propose the concept of (n, δ)-Survivability. We first define (n, δ)-survivability *for a given source-destination pair*, i.e., pairwise (n, δ)-survivability.

Definition 5 (Pairwise (n,δ) Survivability). *In a time-varying graph $\mathcal{G}$, a source-destination pair (s,d) is (n,δ)-survivable if d is still reachable from s after the occurrence of any n failures, with each failure lasting for at most δ consecutive slots.*

Note that the above definition assumes that each failure is *consecutive* in time and also has a finite duration (as opposed to permanent failures in the traditional model). Thus, there could be *multiple failures* occurring to the same link, where each failure happens at a distinct instant. For example, if a link fails in slots $t = 1, 2, 4, 5$ and $\delta = 2$ slots, then there are *two* 2-slot failures that happen in slot 1 and 4, respectively. We can further define global (n,δ)-survivability.

Definition 6 (Global (n,δ)-Survivability). *A time-varying network is (n,δ)-survivable if all pairs of nodes are (n,δ)-survivable.*

Since it only takes $O(|V|^2)$ to check all pairs of nodes, global (n,δ)-survivability can be easily derived from pairwise (n,δ)-survivability. Therefore, we will focus on pairwise (n,δ)-survivability for a given pair of nodes (s,d) throughout the rest of this paper.

Discussion: The above definitions do not impose any assumption about *when* and *where* the n failures occur and thus imply the *worst-case* survivability. In other words, (n,δ)-survivability means the network can survive n failures that last for δ slots *wherever* and *whenever* these failures occur. The parameter n reflects "spatial survivability", indicating *how many* failures the network can survive, and the parameter δ reflects "temporal survivability", indicating *how long* these failures can last.

Note also that (n,δ)-survivability is a generalized definition. For example, if $\delta = T$ (note that T is the time horizon), then (n,δ)-survivability reflects the number of *permanent failures* the network can tolerate, which becomes the conventional notion of survivability used in static networks.

Finally, it should be mentioned that failures can be either link failures or node failures. Since node failures can be converted to link failures by node splitting (see [19], Chapter 7.2), we will consider link failures unless otherwise stated.

3.2 Survivability Metrics

In static networks, two commonly-used survivability metrics are: MinCut, i.e., the minimum number of edges whose deletion can separate the source and the destination, and MaxFlow, i.e., the maximum number of edge-disjoint paths from the source to the destination. If MinCut (or MaxFlow) equals n, the destination is still connected to the source after any $n-1$ link failures. However, by its very nature, a time-varying network has different topologies at different instants, so its connectivity or survivability must be measured over a long time interval and these static metrics cannot be directly applied to time-varying networks. In this section, we introduce two new metrics for (n,δ)-survivability. The fundamental relationship between the two metrics will be further discussed in Section 3.3.

3.2.1 *Survivability Metric:* MinCut_δ

Before we proceed to the first survivability metric, it is necessary to introduce the notions of δ-removal and δ-cut.

Definition 7 (δ-removal). *A δ-removal is the deletion of a link for δ consecutive time slots.*

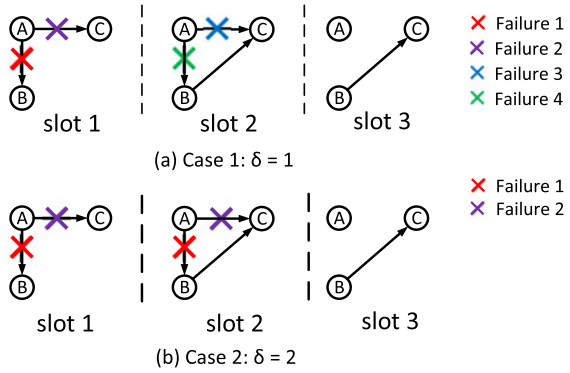


Fig. 3. Illustration of MinCut_δ and (n,δ)-survivability. The source-destination pair is (A, C). (a) When $\delta = 1$, each failure lasts for one slot and it requires at least four one-slot failures to disconnect A and C, i.e., $\mathsf{MinCut}_1 = 4$. (b) When $\delta = 2$, each failure lasts for two slots and it requires only two 2-slot failures to disconnect the network, i.e., $\mathsf{MinCut}_2 = 2$. As a result, the network is $(3,1)$-survivable and $(1,2)$-survivable, meaning that it can tolerate any three failures lasting for one slot or any single failure lasting for two slots.

Intuitively, a δ-removal just corresponds to a link failure that lasts for δ consecutive time slots.

Definition 8 (δ-cut). *A δ-cut is a set of δ-removals that can render the destination unreachable from the source.*

The above definition is similar to the traditional notion of graph cuts except that δ-cuts also account for the duration of removals.

Now we are ready to introduce the first metric for (n,δ)-survivability, namely MinCut_δ. This metric directly follows from the definition of (n,δ)-survivability and is analogous to MinCut in static networks.

Definition 9 (MinCut_δ). *MinCut_δ is the cardinality of the smallest δ-cut, i.e., the minimum number of δ-removals needed to render the destination unreachable from the source.*

Discussion. First, MinCut_δ gives the minimum number of δ-removals required to disconnect the time-varying network. In particular, when $\mathsf{MinCut}_\delta = n$, the source-destination pair can safely survive any $n-1$ failures that last for δ slots and is thus $(n-1,\delta)$-survivable. Second, MinCut_δ generalizes MinCut in static networks since we can simply set $\delta = T$ such that a δ-removal becomes a permanent link removal.

Fig. 3 gives an illustration of MinCut_δ and (n,δ)-survivability. By computing the value of MinCut_δ, we can conclude that the network is $(3,1)$-survivable and $(1,2)$-survivable, meaning that it can tolerate *any 3 failures lasting for one slot* or *any single failure lasting for two slots*. By comparison, the traditional notions of graph cut and survivability presume permanent failures and fail to account for the influence of failure duration. Thus, (n,δ)-survivability is a much finer-grained characterization of network survivability and is particularly suitable for time-varying networks.

Formulation. MinCut_δ corresponds to the following Integer Linear Programming (ILP) problem:

$$\begin{aligned} \min \quad & \sum_{(e,t)\in C} y_{e,t} \\ \text{s.t.} \quad & \sum_{(e,t)\in R(\delta,J)} y_{e,t} \geq 1, \ \forall J \in \mathcal{J}_{sd}, \\ & y_{e,t} \in \{0,1\}, \ \forall (e,t) \in C. \end{aligned}$$

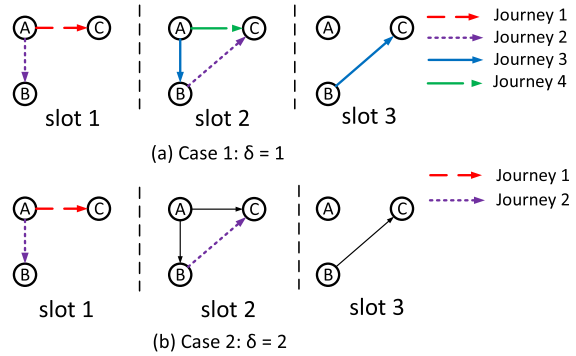


Fig. 4. Illustration of δ-disjoint journeys. The source-destination pair is (A, C). (a) When $\delta = 1$, any two different δ-disjoint journeys cannot use the same link within the same slot, and there are four δ-disjoint journeys. (b) When $\delta = 2$, only two δ-disjoint journeys exist since any link cannot be used by two δ-disjoint journeys within two slots. For example, link A $\rightarrow$ B has been used by Journey 2 in slot 1, so any other δ-disjoint journey cannot use this link in slot 1 or 2.

Here, $y_{e,t}$ is a binary variable indicating whether a δ-removal occurs to edge e in slot t, and C is the set of contacts in the time-varying graph. $\mathcal{J}_{sd}$ is the set of feasible journeys from s to d. For any $J \in \mathcal{J}_{sd}$, we define $R(\delta, J)$ as the set of contacts $\{(e,t)\}$ such that if $y_{e,t} = 1$ then journey J will be disrupted, i.e., $R(\delta, J) = \{(e,t) | \ \exists (e,t') \in C_J \text{ s.t. } 0 \leq t' - t < \delta\}$, where C_J is the set of contacts used by journey J. Thus, the first constraint in the above ILP forces every journey from s to d to be disrupted by at least one of the selected δ-removals, such that d is not reachable from s.

The above formulation is concise but has an exponential number of constraints because the number of possible journeys is exponential in the number of contacts. There also exists a compact ILP formulation which is less intuitive and omitted here for brevity. The complexity and the algorithm for solving the above ILP will be further discussed in Section 4.2.

3.2.2 *Survivability Metric:* MaxFlow$_\delta$

The second survivability metric, namely MaxFlow$_\delta$, is analogous to MaxFlow in static networks. Before the detailed definition of this metric, we first introduce the notion of δ-disjoint journeys.

Definition 10 (δ-disjoint Journey). *A set of journeys from the source to the destination are δ-disjoint if any two of these journeys do not use the same edge within δ time slots.*

Mathematically, suppose $\mathcal{J}$ is a set of δ-disjoint journeys. For any two journeys $J_1, J_2 \in \mathcal{J}$, if edge e is used by J_1 in slot t, then J_2 cannot use the same edge e from slot $t - \delta + 1$ to slot $t + \delta - 1$. In other words, sliding a window of δ slots over time, we can observe at most one active journey over each edge within the window. Fig. 4 gives an example of δ-disjoint journeys for the cases where $\delta = 1$ and $\delta = 2$.

It is easy to see that each one of the δ-disjoint journeys keeps a "temporal distance" of δ slots from others. Due to the temporal distance, any failure that lasts for δ slots can influence at most one of these δ-disjoint journeys. Consequently, the maximum number of δ-disjoint journeys in a time-varying network is a good indicator of its survivability. The more δ-disjoint journeys there exist, the more failures (lasting for δ slots) the network can survive. Now it is natural to introduce the second survivability metric MaxFlow$_\delta$.

Definition 11 (MaxFlow$_\delta$). MaxFlow$_\delta$ *is the maximum number of δ-disjoint journeys from the source to the destination.*

Discussion. First, we would like to compare MaxFlow (for static networks) and MaxFlow$_\delta$ (for time-varying networks). MaxFlow considers disjoint paths which require *spatial disjointness*, i.e., any two disjoint paths never use the same link. This requirement is too demanding for time-varying networks because such networks often have sparse spatial connectivity. In the example of bus communication networks (see Section 5), we will see that a time-varying network may not have any spatially-disjoint paths. Thus, MaxFlow is not an appropriate metric for time-varying networks. By comparison, MaxFlow$_\delta$ considers δ-disjoint journeys, which allows for *temporal disjointness*. Moreover, MaxFlow$_\delta$ generalizes MaxFlow since we can simply set $\delta = T$ so that δ-disjoint journeys become spatially disjoint.

Second, MaxFlow$_\delta$ not only gives us a measure of network survivability but also tells us how to achieve such survivability. The idea is similar to Disjoint-Path Protection in static networks [26], [27], where disjoint paths are used as backup routes. In time-varying networks, we can send packets along different δ-disjoint journeys to increase transmission reliability. If we use n δ-disjoint journeys (i.e., MaxFlow$_\delta \geq n$), the transmission can survive any $n-1$ failures that last for δ slots and is thus $(n-1, \delta)$-survivable.

Formulation. MaxFlow$_\delta$ corresponds to the following ILP:

$$\begin{aligned} \max \quad & \sum_{J \in \mathcal{J}_{sd}} x_J \\ \text{s.t.} \quad & \sum_{J:(e,t) \in R(\delta,J)} x_J \leq 1, \ \forall (e,t) \in C \\ & x_J \in \{0,1\}, \ \forall J \in \mathcal{J}_{sd}. \end{aligned}$$

Here, x_J is a binary variable indicating whether journey J should be added to the set of δ-disjoint journeys. All the other notations have the same meanings as in the formulation of MinCut$_\delta$. The first constraint checks every edge and forces this edge to be used by at most one of the δ-disjoint journeys in any time window of δ slots. The above formulation also has an exponential number of constraints. A compact formulation also exists but is omitted for brevity. The complexity and the algorithms for solving the above ILP will be further investigated in Section 4.1.

3.3 Analysis of Metrics

Recall that in static networks, the well-known Menger's Theorem shows that Mincut equals MaxFlow; due to this equivalence, we can compute MaxFlow and MinCut efficiently (e.g., the Ford-Fulkerson algorithm). Hence, it is necessary to study the fundamental relationship between MinCut$_\delta$ and MaxFlow$_\delta$, in order to gain insights into their computation. Let MinCut$_\delta^{\mathrm{R}}$ and MaxFlow$_\delta^{\mathrm{R}}$ be the LP relaxation for the ILP formulation of MinCut$_\delta$ and MaxFlow$_\delta$, respectively. It is easy to show that MinCut$_\delta^{\mathrm{R}}$ is the *dual problem* of MaxFlow$_\delta^{\mathrm{R}}$. By strong duality and the properties of LP relaxation, we make the following observation:

$$\text{MaxFlow}_\delta \leq \text{MaxFlow}_\delta^{\mathrm{R}} = \text{MinCut}_\delta^{\mathrm{R}} \leq \text{MinCut}_\delta.$$

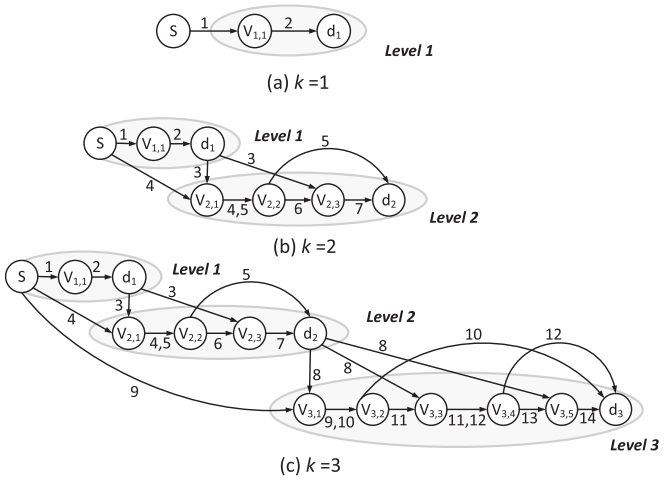


Fig. 5. Examples used in the proof of Property (II) in Theorem 1. The source-destination pair is (s, d_k) in graph $\mathcal{G}_k$ ($k = 1, 2, 3$). Edge traversal delay is one slot.

As a result, as long as Menger's Theorem holds in time-varying networks (i.e., $\mathsf{MaxFlow}_\delta = \mathsf{MinCut}_\delta$), all of the four quantities will be equivalent, and we can simply compute $\mathsf{MaxFlow}_\delta$ and MinCut_δ by solving their LP relaxations. Interestingly, the following theorem shows that Menger's Theorem only "conditionally" holds in time-varying networks.

Theorem 1. *Time-varying graphs have the following survivability properties:*

- *(I) If $\delta = 1$, then Menger's Theorem holds for any time-varying graph, i.e.,* $\mathsf{MaxFlow}_1 = \mathsf{MinCut}_1$.
- *(II) For any $\delta \geq 2$, there exist instances of time-varying graphs such that* $\mathsf{MaxFlow}_\delta < \mathsf{MinCut}_\delta$. *Moreover, the gap ratio $\frac{\mathsf{MinCut}_\delta}{\mathsf{MaxFlow}_\delta}$ can grow without bound.*

Proof. We prove the two properties separately.

- *Proof of Property (I).*

Consider a time-varying graph $\mathcal{G}$ with the source s and the destination d. Let $\mathsf{MaxFlow}$ be the maximum number of *node-disjoint* paths from s to d in the Line Graph of $\mathcal{G}$ and MinCut be the cardinality of the smallest *node cut* that separates s and d in the Line Graph. It is not hard to verify the following lemma. □

Lemma 1. $\mathsf{MaxFlow}_1 = \mathsf{MaxFlow}$ *and* $\mathsf{MinCut}_1 = \mathsf{MinCut}$.

Remark 1. Lemma 1 does not holds for $\delta \geq 2$. For example, if $\delta = 2$, there is only one δ-disjoint journey in Fig. 2a but there are two node-disjoint paths in its Line Graph.

Now we can apply the node-version Menger's Theorem to the Line Graph and obtain $\mathsf{MaxFlow} = \mathsf{MinCut}$. By Lemma 1, we can conclude that

$$\mathsf{MaxFlow}_1 = \mathsf{MaxFlow} = \mathsf{MinCut} = \mathsf{MinCut}_1.$$

- *Proof of Property (II).*

The non-trivial part is to show that the gap ratio can be arbitrarily large. We construct a family of time-varying graphs $\{\mathcal{G}_k\}_{k\geq 1}$ such that $\frac{\mathsf{MinCut}_\delta}{\mathsf{MaxFlow}_\delta} = k$ for any $\delta \geq 2$ in the kth graph. The constructions for $k = 1, 2, 3$ are shown in Fig. 5. We can observe that $\mathcal{G}_1$ is a single-level graph; $\mathcal{G}_2$ is built upon $\mathcal{G}_1$, where the first level is exactly $\mathcal{G}_1$; similarly, $\mathcal{G}_3$ is built upon $\mathcal{G}_2$, where the first two levels are exactly $\mathcal{G}_2$.

We use inductions to prove that $\mathsf{MaxFlow}_\delta = 1$ while $\mathsf{MinCut}_\delta = k$ for any $\delta \geq 2$ in the kth graph $\mathcal{G}_k$.

- In $\mathcal{G}_1$, the source-destination pair is (s, d_1). It is obvious that $\mathsf{MaxFlow}_\delta = \mathsf{MinCut}_\delta = 1$ for any $\delta \geq 2$.
- In $\mathcal{G}_2$, the source-destination pair is (s, d_2). We want to show that $\mathsf{MaxFlow}_\delta = 1$ but $\mathsf{MinCut}_\delta = 2$ for any $\delta \geq 2$. To see $\mathsf{MaxFlow}_\delta = 1$, we notice that there are two possible choices for traveling from s to d_2. One is via node d_1 and the other is to directly descend to level 2. The former choice yields only one δ-disjoint journey from s to d_2 since we know from $\mathcal{G}_1$ that there is only one δ-disjoint journey from s to d_1. For the latter choice, the only possibility is $s \to v_{2,1} \to v_{2,2} \to v_{2,3} \to d_2$ but this journey cannot be δ-disjoint of any journey in the first choice (i.e., via node d_1) for any $\delta \geq 2$. Hence, there is only one δ-disjoint journey from s to d_2, i.e., $\mathsf{MaxFlow}_\delta = 1$ for any $\delta \geq 2$. Now it remains to show $\mathsf{MinCut}_\delta = 2$ and we prove this by showing that any single δ-removal cannot disconnect d_2 from s. If the δ-removal takes place in level 1, there exists a feasible journey from s to d_2 via $s \to v_{2,1} \to v_{2,2} \to v_{2,3} \to d_2$ in slots 4, 5, 6, 7. If the δ-removal occurs to some contact outside level 1, the journey from s to d_1 is still available. Moreover, there exists at least one journey from d_1 to d_2 since there are two spatially disjoint journeys from d_1 to d_2 (one journey is via $d_1 \to v_{2,1} \to v_{2,2} \to d_2$ in slots 3, 4, 5, and the other journey is via $d_1 \to v_{2,3} \to d_2$ in slots 3, 7). As a result, d_2 is still reachable from s via $s \to d_1 \to d_2$. Now it is safe to conclude that any single δ-removal cannot disconnect d_2 from s, which implies $\mathsf{MinCut}_\delta \geq 2$. Note that d_2 can be easily made unreachable from s with 2 δ-removals (e.g., disable the two contacts from s). Therefore, $\mathsf{MinCut}_\delta = 2$

Note that the key part in $\mathcal{G}_2$ is the "shortcut edge" $v_{2,2} \to d_2$ which can only be used by journeys that travel through d_1.

Now we generalize the above idea to arbitrary k. In general, graph $\mathcal{G}_k$ has k levels. Different levels share the same source s and have their own "virtual destinations" $d_1, \ldots, d_k$. Note that the real source-destination pair in $\mathcal{G}_k$ is (s, d_k). Besides, the ith level has $2i - 1$ "inner nodes", where we denote $v_{i,j}$ the jth inner node at level i. The "inner contacts" in level i include the following.

- $s \to v_{i,1}$: active in slot i;
- $v_{i,j} \to v_{i,j+1}$: active in slot i for any even j and in slots $i - 1$, i for any odd j;
- $v_{i,2i-1} \to d_i$: active in slot i;
- $v_{i,j} \to d_i$: active in slot $i - 1$ for any even j.

There is also a "cross-level edge" from a lower level to a higher level, i.e., $d_i \to v_{i+1,j}$ which is active in slot i for any $1 \leq i \leq k - 1$ and odd j.

Then we prove by induction that in $\mathcal{G}_k$, $\mathsf{MaxFlow}_\delta = 1$ and $\mathsf{MinCut}_\delta = k$ for any $\delta \geq 2$. When $k = 1$, we have shown that $\mathsf{MinCut}_\delta = 1$ and $\mathsf{MaxFlow}_\delta = 1$ for any $\delta \geq 2$. Now suppose the claim holds in $\mathcal{G}_1, \ldots, \mathcal{G}_k$ for some $k \geq 1$. In $\mathcal{G}_{k+1}$, there are two possible choices for traveling from s to d_{k+1}: one is via node d_k (i.e., $s \to d_k \to d_{k+1}$) and the other is to directly descend to the $(k+1)$th level (i.e., $s \to v_{k+1,1} \to v_{k+1,2} \to \cdots \to d_{k+1}$). By our induction, there is only one

δ-disjoint journey from s to d_k, so the former choice can yield only one δ-disjoint journey from s to d_{k+1}. Moreover, whichever journey from s to d_{k+1} that travels via d_k is chosen, it cannot be δ-disjoint of the latter choice of journey (i.e., $s \rightarrow v_{k+1,1} \rightarrow v_{k+1,2} \rightarrow \cdots \rightarrow d_{k+1}$) for any $\delta \geq 2$. As a result, there is only one δ-disjoint journey from s to d_k, i.e., $\mathsf{MaxFlow}_\delta = 1$ for any $\delta \geq 2$ in $\mathcal{G}_{k+1}$.

As for MinCut_δ, we notice that $\mathsf{MinCut}_\delta \leq k+1$ in $\mathcal{G}_{k+1}$ since removing all the contacts starting from s can separate s and d_{k+1}. Then it is sufficient to prove $\mathsf{MinCut}_\delta \geq k+1$, i.e., any k δ-removals cannot disconnect s and d_{k+1} when $\delta \geq 2$. To show this point, we first denote by C_1 the set of inner contacts in levels $1, 2, \ldots, k$ and all cross-level contacts, and denote by C_2 the inner contacts in level $k+1$. Then we discuss three possible scenarios.

- Case 1: all the k δ-removals are in C_1. In this case, it is still possible to travel from s to d_{k+1} within level $k+1$, i.e., $s \rightarrow v_{k+1,1} \rightarrow v_{k+1,2} \rightarrow \cdots \rightarrow d_{k+1}$.
- Case 2: all the k δ-removals are in C_2. It can be observed that there are $k+1$ δ-disjoint journeys from d_k to d_{k+1}, so even with k δ-removals of the inner contacts in level $k+1$, there is still one journey from d_k to d_{k+1}, thus preserving a journey $s \rightarrow d_k \rightarrow d_{k+1}$.
- Case 3: there are $i > 0$ δ-removals in C_1 and $k-i > 0$ δ-removals in C_2. In this case, we first notice that with $i < k$ δ-removals in C_1, there is still a journey from s to d_k (by the induction in $\mathcal{G}_k$) and there are at least $k+1-i$ δ-disjoint journeys from d_k to d_{k+1} even after the i δ-removals in C_1. As a result, with $k-i > 0$ δ-removals in C_2, at least one journey from d_k to d_{k+1} is preserved, thus also preserving a journey $s \rightarrow d_k \rightarrow d_{k+1}$.

Therefore, we can conclude that $\mathsf{MinCut}_\delta = k+1$ in $\mathcal{G}_{k+1}$ for any $\delta \geq 2$, which completes the induction proof.

The first proposition in Theorem 1 implies that $\mathsf{MaxFlow}_1$ and MinCut_1 can be efficiently computed. We provide two possible approaches here. The first approach is to directly solve the LP relaxation. Alternatively, we can first derive the Line Graph of the original time-varying graph and then apply traditional MaxFlow algorithms such as Ford-Fulkerson algorithm to find the maximum number of *node-disjoint paths* in the Line Graph. The correctness of this approach can be easily verified by observing that $\mathsf{MaxFlow}_1$ equals the maximum number of node-disjoint paths in the Line Graph. This also gives the result for MinCut_1 since $\mathsf{MinCut}_1 = \mathsf{MaxFlow}_1$.

The second proposition in Theorem 1 demonstrates that Menger's Theorem could break down in time-varying graphs, which highlights a key difference between time-varying and static graphs. Due to this fundamental difference, the traditional techniques used to compute MaxFlow or MinCut in static networks, such as the Ford-Fulkerson algorithm, cannot be applied to time-varying graphs to compute $\mathsf{MaxFlow}_\delta$ or MinCut_δ. In the next section, we will further discuss the computation of the two metrics.

4 Computational Consideration

In this section, we study the computational complexity and related algorithms for computing $\mathsf{MaxFlow}_\delta$ and MinCut_δ in time-varying networks.

4.1 Computation of $\mathsf{MaxFlow}_\delta$

We start with the computation of $\mathsf{MaxFlow}_\delta$ for an *arbitrary value* of δ, referred to as the **δ-MAXFLOW** problem. We first show that δ-MAXFLOW is NP-hard to approximate within $O(|E|^{1/2-\epsilon})$ for any $\epsilon > 0$ and then develop an algorithm that achieves $O(\sqrt{|E|})$-approximation. Finally, a special case where δ-MAXFLOW may be solved in polynomial time is discussed.

4.1.1 Computational Complexity

The following theorem shows that δ-MAXFLOW is even NP-hard to approximate.

Theorem 2. *δ-MAXFLOW is NP-hard. It is even NP-hard to achieve $O(|E|^{1/2-\epsilon})$-approximation for any $\epsilon > 0$.*

Proof. The proof is based on a reduction from the Bounded-Length Edge-Disjoint Paths (BLEDP) problem which is NP-hard [22].

- Problem: BLEDP.
- Instance:
  - A weighted digraph $G' = (V', E')$, where the weight on edge e indicates its length (denoted by l_e). The length of each edge is a positive integer.
  - The source-destination pair (s, d).
  - An integer $L > 0$ indicating the length bound.
- Question: Find the maximum number of edge-disjoint paths from s to d in G' such that the length of each of these paths is upper-bounded by L. □

Here we make an additional assumption that there exists no edge with its length greater than L in G'. We also assume that there are no isolated nodes in G'. These assumptions do not change the complexity of BLEDP because we can simply remove these isolated nodes or long edges from G' without any influence on the optimal solution.

The high-level idea of the reduction is to transform the "spatial length bound" into a "temporal length bound". Note that in our model, a natural temporal bound T exists so we set $T = L$. In addition, we also need to make sure that whenever edge e is crossed, a "temporal distance" of l_e slots is traversed. Since it is assumed that edge-traversal delay is one time slot, we can expand each edge in series such that extra delay is incurred. To be more specific, if the length of edge e is l_e, we replace this single edge by l_e edges that are concatenated in series; each of the catenated edges has one-slot traversal delay and is active in the entire time span. An example is illustrated in Fig. 6. It is trivial to check that BLEDP is equivalent to solving δ-MAXFLOW in the constructed time-varying graph for $\delta = T$. Hence, δ-MAXFLOW is NP-hard.

It remains to investigate the hardness of approximating δ-MAXFLOW. Guruswami et al. [22] proved that BLEDP is NP-hard to approximate within $O(|E'|^{1/2-\epsilon})$ for any $\epsilon > 0$. Moreover, in their constructed graph (underlying the BLEDP problem) for proving the inapproximability bound, the sum of arc lengths is $\sum_{e \in E'} l_e = O(|E'|)$. In our constructed time-varying graph, we have $|E| = \sum_{e \in E'} l_e = O(|E'|)$, which implies $|E'| = \Omega(|E|)$. Therefore, it is also NP-hard to achieve $O(|E|^{1/2-\epsilon})$-approximation for δ-MAXFLOW.

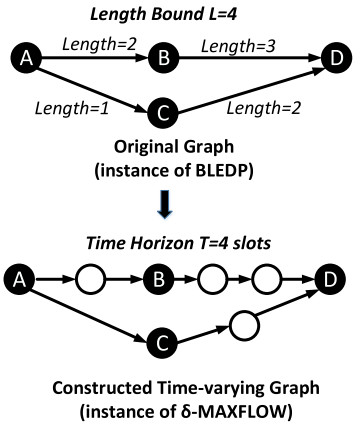


Fig. 6. Illustration of the reduction from BLEDP to δ-MAXFLOW. Note that in the constructed time-varying graph, edge traversal delay is one time slot and each edge is active in the entire time span $\{1, 2, 3, 4\}$.

4.1.2 Approximation Algorithm

Next, we propose a greedy algorithm that achieves $O(\sqrt{|E|})$ approximation. Such an approximation ratio is near-optimal as compared to the inapproximability bound of $O(|E|^{12-\epsilon})$ (see Theorem 2). The greedy algorithm is inspired by the Blocking Flow Algorithm [24] for the network flow problem. Intuively, it iteratively calculates the shortest journey that is δ-disjoint from those previously selected journeys. Before we move on to the detailed algorithm description, it is necessary to introduce a short-hand term called *interfering contact*.

Definition 12 (Interfering Contact). *Consider a journey J. A contact (e, t) is said to be an interfering contact of journey J if there exists a contact (e, t') used by J such that $|t - t'| < \delta$.*

If J is one of the δ-disjoint journeys, then its interfering contacts cannot be used by any other δ-disjoint journey.

Now we are ready to present a greedy algorithm for δ-MAXFLOW, shown as Algorithm 1. It first computes the Line Graph (see Section 2.3) of the original time-varying graph and then finds an s-d path with the least number of nodes in the Line Graph. By the property of Line Graphs (see Observation 1 in Section 2.3), this path corresponds to a journey in the original time-varying graph; then we add this journey to the set of δ-disjoint journeys. The next operation is to remove all the interfering contacts of this journey from the time-varying graph and reconstruct the Line Graph from the *remaining time-varying graph*. If s and d are still connected in the Line Graph, the above procedure is repeated until s and d are disconnected. From the definition of interfering contacts, we can easily verify that the obtained journeys are δ-disjoint.

Now we estimate the time complexity of this greedy algorithm. In each iteration (steps 1-1), we need to compute the Line Graph and the path with the least number of nodes. Recall that we denote $|C|$ the total number of contacts in the time-varying graph. Then it takes $O(|C|^2)$ time to construct the Line Graph and $O(|C|^2)$ time to compute the path with the least number of nodes (suppose BFS is used). Also note that the total number of iterations is at most $|C|$ since the number of δ-disjoint journeys cannot exceed $|C|$ and each iteration adds one δ-disjoint journey. Consequently, the overall time complexity of the greedy algorithm is $O(|C|^3)$.

Algorithm 1. Greedy Algorithm for δ-MAXFLOW

Input:
$\mathcal{G}$: the time-varying graph;
(s, d): the source-destination pair;
δ: the degree of temporal disjointness;
Output:
$J_1, \ldots, J_m$: a set of δ-disjoint journeys.
1: Initialize $m = 0$;
2: Compute the Line Graph of $\mathcal{G}$;
3: **if** s and d is disconnected in the Line Graph **then**
4: Go to step 10;
5: **end if**
6: $m \leftarrow m + 1$;
7: In the Line Graph, find an $s - d$ path P_m that passes the least number of nodes (the corresponding journey is denoted by J_m);
8: Remove all the interfering contacts of J_m from $\mathcal{G}$;
9: Go to step 2;
10: END.

The approximation ratio of this greedy algorithm is given in the following theorem.

Theorem 3. *The greedy algorithm attains $O(\sqrt{|E|})$ approximation for δ-MAXFLOW, i.e., $\frac{\mathsf{OPT}}{\mathsf{ALG}} = O(\sqrt{|E|})$.*

Proof. If the the destination is unreachable from the source, both the optimal solution and the greedy algorithm will yield a result of zero, where no approximation gaps exist. Hence, it is enough to consider the scenario where the destination is reachable from the source.

Before the detailed proof, it is essential to define the notions of *short paths* and *long paths* in the Line Graph. Let k be an *arbitrary* positive integer. A short path consists of at most k nodes while a long path is made up of more than k nodes. Their corresponding journeys are called the *short journey* (traversing at most k edges) and the *long journey* (traversing more than k edges), respectively. Denote $\mathcal{J}^* = \{J_1^*, \ldots\}$ the optimal solution and $\mathcal{J} = \{J_1, \ldots\}$ the solution obtained by the greedy algorithm.

We first prove that the number of long journeys in $\mathcal{J}^*$ is at most $\frac{|E|(\frac{T}{\delta}+1)}{k}$. Indeed, since journeys in $\mathcal{J}^*$ are δ-disjoint, each edge can be traversed by at most $\lceil \frac{T}{\delta} \rceil$ journeys in $\mathcal{J}^*$. At the same time, each of the long journeys in $\mathcal{J}^*$ traverses more than k edges so the total number of long journeys in $\mathcal{J}^*$ can be at most $\lfloor \frac{\lceil \frac{T}{\delta} \rceil |E|}{k} \rfloor \leq \frac{|E|(\frac{T}{\delta}+1)}{k}$.

Then we prove that the number of short journey in $\mathcal{J}^*$ is at most $2k \times |\mathcal{J}|$. To show this point, we first prove that each short journey (say J_j^*) in $\mathcal{J}^*$ is interfered by some short journey (say J_i) in $\mathcal{J}$ (i.e., J_j^* and J_i use the same edge within δ slots). Note that each short journey in $\mathcal{J}^*$ must be interfered by at least one journey in $\mathcal{J}$ otherwise the greedy algorithm is not finished. Let $J_i \in \mathcal{J}$ be the journey that interferes with some journey $J_j^* \in \mathcal{J}^*$ *for the first time*, i.e., journeys constructed in the greedy algorithm before J_i do not interfere with J_j^*. In other words, when the greedy algorithm is constructing journey J_i, journey J_j^* is also a candidate journey. Since J_i is selected rather than J_j^*, it implies that the number of edges traversed by J_i is less or

equal to that of J_j^*. Due to the fact that J_j^* is a short journey, we can conclude that J_i is also a short journey.

Meanwhile, each short journey in $\mathcal{J}$ can interfere with at most $2k$ δ-disjoint journeys because any short journey in $\mathcal{J}$ contains at most k contacts and each of these contacts can interferes with at most 2 δ-disjoint journeys. Hence, the total number of δ-disjoint journeys that can be interfered by the short journeys in $\mathcal{J}$ is at most $2k \times |\mathcal{J}|$. Since we have shown that each short journey in $\mathcal{J}^*$ is interfered by at least one short journey in $\mathcal{J}$, it is safe to conclude that the number of short journeys in $\mathcal{J}^*$ is upper-bounded by $2k \times |\mathcal{J}|$, which means that

$$|\mathcal{J}^*| = |\mathcal{J}^*_{long}| + |\mathcal{J}^*_{short}| \leq \frac{|E|(\frac{T}{\delta}+1)}{k} + 2k \times |\mathcal{J}|. \quad (1)$$

Now we set k to be the integer such that $\sqrt{|E|(\frac{T}{\delta}+1)} \leq k < \sqrt{|E|(\frac{T}{\delta}+1)}+1$. Then it follows that

$$\begin{aligned}|\mathcal{J}^*| &< \sqrt{|E|(\frac{T}{\delta}+1)} + 2\Big(\sqrt{|E|(\frac{T}{\delta}+1)}+1\Big)|\mathcal{J}| \\ &\leq \sqrt{|E|(\frac{T}{\delta}+1)}|\mathcal{J}| + 2\Big(\sqrt{|E|(\frac{T}{\delta}+1)}+1\Big)|\mathcal{J}| \\ &= \Big(3\sqrt{|E|(\frac{T}{\delta}+1)}+2\Big)|\mathcal{J}|,\end{aligned}$$

where the first inequality follows from the setting of k and the second inequality holds because of our premise that $|\mathcal{J}| \geq 1$ (i.e., the destination is reachable from the source). Since T is a bounded integer and $\delta \leq T$, we can finally conclude that Algorithm 1 achieves $O(\sqrt{|E|})$-approximation. □

In practice, the greedy algorithm also performs extremely well, as is demonstrated by the following numerical results.

Numerical Results for the Greedy Algorithm. In order to understand the performance of the greedy algorithm, we compare it with the optimal solution to δ-MAXFLOW. In our experiment, 1000 random time-varying graphs are tested. Each network has 20 nodes and the underlying static graph is a random scale-free graph. The time horizon is $T = 20$ slots and we assume each link is active with a probability $p = 0.5$ in each slot. The source-destination pair is also randomly selected. The optimal solution to δ-MAXFLOW is derived by directly solving its ILP formulation. Fig. 7 shows the comparison, where the approximation gap is calculated by $\frac{\text{OPT}-\text{ALG}}{\text{ALG}}$. We can observe that the approximation gap is usually less than 8 percent, much better than the theoretical bound in Theorem 3.

4.1.3 Special Case: Fixed-Parameter Tractability

Sometimes we may only want to find a fixed number k δ-disjoint journeys (called the ***δ*-FIXFLOW(*k*)** problem) instead of the maximum number. For example, traditional disjoint-path protection usually exploits two disjoint paths, one as the primary path and the other as the backup path. Unfortunately, δ-FIXFLOW(k) problem is still NP-hard in general. This hardness result can be easily derived from the work of Kleinberg et al. [21]. They showed that a special case of δ-FIXFLOW(k) problem is NP-hard, where each link is active for exactly one slot and spatial disjointness (i.e., $\delta = T$) is assumed. Hence, δ-FIXFLOW(k) problem is also NP-hard. However, the good news is that δ-FIXFLOW(k) is solvable in polynomial time if the underlying graph is a Directed Acyclic Graph (DAG). In other words, the problem in this special case is Fixed-Parameter-Tractable (FPT) with respect to parameter k. This can be achieved by modifying the classic *Pebbling Game* [25]. The detailed procedures are as follows.

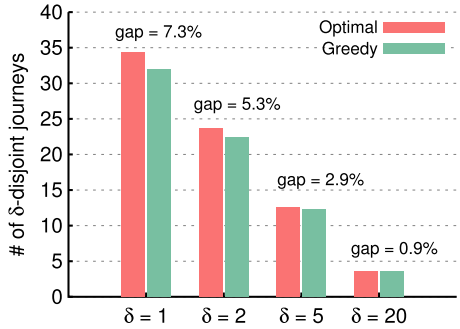


Fig. 7. Comparison between the greedy algorithm (Algorithm 1) and the optimal solution to δ-MAXFLOW.

Suppose we are given a fixed integer $k > 0$ and need to find k δ-disjoint journeys. Denote by $L(\mathcal{G})$ the Line Graph of *the input time-varying graph* $\mathcal{G}$ and let $L(G)$ be the Line Graph of *the underlying graph* G. Then the pebbling game executes the following operations.

- Perform topological sorting over $L(G)$. Note that if there is an edge $u \to v$ in $L(G)$, then the level of u is higher than the level of v. Also note that each node in $L(G)$ represents an edge in G, so after the topological sorting we get the level for each edge in G. Denote l_e the level of edge $e \in E$.
- In $L(\mathcal{G})$, associate node $v_{e,t}$, which corresponds to contact (e, t), with level l_e.
- The pebbling game is run over $L(\mathcal{G})$ with the following rules. Initially, there are k pebbles at the source s. In each round of the game, we decide whether pebbles can be moved. A pebble can be move from node $v_{e,t}$ to $v_{e',t'}$ in $L(\mathcal{G})$ if (i) there is an edge between $v_{e,t}$ and $v_{e',t'}$ in $L(\mathcal{G})$, (ii) there are no other pebbles resided in any node $v_{e',t''}$ such that $|t'' - t'| < \delta$, and (iii) the level of node $v_{e,t}$ is higher than or equal to any other nodes that are resided by a pebble. Note that if a pebble is moved from node s, rule (iii) can be ignored; if a pebble is moved to node d, rule (ii) can be neglected. When all the pebbles are moved to the destination d, the pebbling game is won.

Note that when the game ends, if all the k pebbles are moved to the destination, we can easily find k δ-disjoint journeys by using the trajectories of pebbles (by Observation 1); otherwise the game is lost and it is impossible to find k δ-disjoint journeys. The correctness of the pebbling game is given by Theorem 4.

Theorem 4. *For a given constant k, the pebbling game is won if and only if there are k δ-disjoint journeys from s to d.*

Proof. When the pebbling game is won, all of the k pebbles are moved to the destination d along k paths $P_1, \ldots, P_k$ in the line graph, which corresponds to k journeys in the original time-varying graph: $J_1, \ldots, J_k$. Then we prove that these journeys are δ-disjoint.

Suppose two of these journeys (say J_i and J_j) are not δ-disjoint, i.e., they use the same edge (say e) within δ slots. Assume J_i uses edge e in slot t_i and J_j uses e at time t_j. Then we have $|t_i - t_j| < \delta$ by the assumption. Without loss of generality, we let pebble p_i reaches v_{e,t_i} first. Then pebble p_i must leave node v_{e,t_i} before pebble p_j reaches v_{e,t_j} since $|t_i - t_j| < \delta$ (by the second rule of the pebbling game). Denote $v_{e',t'}$ the node that p_j resides in when pebble p_i moves away from v_{e,t_i}. It is obvious that pebble p_j visits node $v_{e',t'}$ before node v_{e,t_j}, so the level of $v_{e',t'}$ is higher than the level of v_{e,t_j}, i.e., $l_{e'} \geq l_e$. By the third rule of the pebbling game, for pebble p_i to be able to move away from node v_{e,t_i}, the level of node v_{e,t_i} must be higher than the level of node $v_{e',t'}$, which means that $l_e > l_{e'}$, thus resulting in a contradiction.

Conversely, it is clear that if there are k δ-disjoint journeys from s to d, then the k pebbles can be moved along the paths corresponding to these journeys, which makes the pebbling game won. □

The analysis of time complexity for the pebbling game is similar to [21]. Recall that the time-varying graph $\mathcal{G}$ has $|C|$ contacts, so there are $|C|+2$ nodes in the Line Graph (including the source node s and the destination node d). Varying the positions of the k pebbles in $L(\mathcal{G})$, we can obtain $(|C|+2)^k$ patterns. The starting pattern is the one where all the pebbles reside in node s and the ending pattern is the one where all the pebbles reach node d. Hence, we will go through at most $(|C|+2)^k$ patterns before the game ends, and the overall time complexity of the pebbling game is $O(|C|^k)$. It is easy to see that when k is a fixed constant, the pebbling game is polynomial-time but becomes exponential when k is a part of the input parameters.

4.2 Computation of MinCut_δ

In this section, we study the computation of MinCut_δ for an arbitrary value of δ, referred to as the **δ-MINCUT** problem.

4.2.1 Computational Complexity

The complexity of δ-MINCUT is given in Theorem 5.

Theorem 5. *δ-MINCUT is NP-hard.*

Proof. Kempe et al. [21] showed that in a special type of time-varying graphs, where each link is active for only one slot, it is NP-hard to determine whether there exists a set of k nodes whose permanent removals can disconnect the source-destination pair. This is obviously a restricted instance of the node-version δ-MINCUT problem, which implies that the node-version δ-MINCUT is NP-hard. Moreover, it can be verified that node-version problems are just a special case of edge-version problems by using node splitting (see [19], Chapter 7.2). Hence, the edge-version δ-MINCUT problem is also NP-hard. □

4.2.2 Approximation Algorithm

In this section, we present an approximation algorithm (referred to as the *min-weight algorithm*) for δ-MINCUT. The algorithm is inspired by the fact that MinCut_δ can be efficiently computed for $\delta = 1$ (see Section 3.3). It first computes the smallest one-cut and then constructs a feasible solution to δ-MINCUT out of the one-cut. The detailed procedures are as follows.

- *Step 1:* Assign a weight to each contact according to its "temporal closeness" to other contacts. Intuitively, if there are more contacts in the "temporal neighborhood" of the given contact, then a δ-removal (i.e., a δ-slot failure) of this contact will disable more neighboring contacts at the same time. Hence, this contact should be given a smaller weight such that it has a higher priority of being removed. We let the weight of a contact be inversely proportional to the number of its "neighboring" contacts (see SETWEIGHT in Algorithm 2).
- *Step 2:* Compute MinCut_1 over the weighted time-varying graph. Note that Property (I) in Theorem 1 still holds in weighted time-varying graphs, so MinCut_1 can be efficiently computed (e.g., by solving the LP relaxation). After this step, we obtain a set of contacts S^* with the smallest sum of weights whose removals will disconnect the source-destination pair.
- *Step 3:* Compute the δ-cover of S^*, i.e., the *smallest* set of δ-removals needed to cover all the contacts in S^*. For example, suppose $S^* = \{(e_1,1),(e_1,2),(e_2,2),(e_2,4)\}$ and $\delta = 2$. Then we need at least three δ-removals to cover all the contacts in S^*: one for $(e_1,1)$ and $(e_1,2)$, one for $(e_2,2)$ and one for $(e_2,4)$; this means that $|\mathsf{Cover}_\delta(S^*)| = 3$. Finally, the δ-cover of S^* is returned as a feasible solution to δ-MINCUT.

The above steps are summarized in Algorithm 2. We briefly discuss the complexity of this algorithm. The time complexity used for setting weights (step 1) is $O(|C|\delta)$ where $|C|$ is the number of contacts in the time-varying network. The time complexity of step 2 depends on the way we compute MinCut_1 (see Section 3.3 for different approaches). For example, if Ford-Fulkerson Algorithm is used over the Line Graph to compute MinCut_1, then step 2 of Algorithm 2 consumes $O(|C|^3)$ time. The computation of the δ-cover (step 3) consumes $O(|C|)$ time. Hence, the overall time complexity of the min-weight algorithm (Algorithm 2) is $O(|C|^3)$.

Algorithm 2. Min-Weight Algorithm for δ-MINCUT

1: Call SETWEIGHT to compute the weight for each contact;
2: Compute MinCut_1 over the weighted time-varying graph, where we obtain a set of contacts S^* with the smallest sum of weights whose removals will disconnect the source-destination pair;
3: Return the δ-cover of S^* as the solution.
4: **Procedure:** SETWEIGHT
5: **for** each contact (e,t) **do**
6: Scan all the δ-slot windows containing (e,t), and find the one that contains the maximum number of contacts (say containing $K_{e,t}$ contacts);
7: Set $\omega_{e,t} = \frac{1}{K_{e,t}}$;
8: **end for**

The approximation ratio of the above min-weight algorithm is given in the following theorem.

Theorem 6. *The min-weight algorithm (Algorithm 2) achieves δ-approximation for δ-MINCUT, i.e., $\frac{\mathsf{ALG}}{\mathsf{OPT}} \leq \delta$.*

Proof. We make two simple observations regarding the weights. The first is that $\omega_{e,t} \geq \frac{1}{\delta}$ since $K_{e,t} \leq \delta$. The second is that the sum of weights that can be removed by one δ-removal is less than or equals to 1. Indeed, consider a certain δ-removal that deletes contacts $(e,t_1),(e,t_2),\ldots,(e,t_m)$. It should be obvious that $K_{e,t_i} \geq m$ for any $1 \leq i \leq m$, which means $\sum_{i=1}^{m} \omega_{e,t_i} = \sum_{i=1}^{m} \frac{1}{K_{e,t_i}} \leq \sum_{i=1}^{m} \frac{1}{m} = 1$. Then we introduce the following lemma. □

Lemma 2. *Let C be an arbitrary set of contacts whose removals disconnect the source-destination pair. The following result holds*

$$\sum_{(e,t)\in C} \omega_{e,t} \leq |\mathsf{Cover}_\delta(C)| \leq \delta \sum_{(e,t)\in C} \omega_{e,t}.$$

Proof. The lower bound directly follows from the second observation mentioned above. Then we get down to proving the upper bound. Denote E_c the set of underlying edges in C. For each edge $e \in E_c$, suppose we need n_e δ-removals to completely delete e from C, and the corresponding removal heads are $(e,t_1),(e,t_2),\ldots,(e,t_{n_e})$, where we assume $1 \leq t_1 < t_2 < \cdots < t_{n_e} \leq T$. Denote $C_{e,i}$ the set of contacts deleted by the δ-removal with head (e,t_i) and define

$$W_{e,i} = \sum_{(e,t)\in C_{e,i}} \omega_{e,t}, \ \forall e \in E_c \text{ and } 1 \leq i \leq n_e. \tag{2}$$

Then we have

$$|\mathsf{Cover}_\delta(C)| = \sum_{e\in E_c} n_e = \sum_{e\in E_c} \sum_{i=1}^{n_e} \frac{\sum_{(e,t)\in C_{e,i}} \omega_{e,t}}{W_{e,i}}, \tag{3}$$

where the last equality is due to Equation (2). We also notice that for any $e \in E_c$ and $1 \leq i \leq n_e$

$$\sum_{(e,t)\in C_{e,i}} \omega_{e,t} \geq \omega_{e,t_i},$$

because contact (e,t_i) is included in $\mathcal{C}_{e,i}$. By simple transformations, we obtain

$$\frac{\sum_{(e,t)\in C_{e,i}} \omega_{e,t}}{\omega_{e,t_i}} \geq 1 = \frac{\sum_{(e,t)\in C_{e,i}} \omega_{e,t}}{W_{e,i}}.$$

Since $\omega_{e,t_i} \geq \frac{1}{\delta}$, we have

$$\delta \sum_{(e,t)\in C_{e,i}} \omega_{e,t} \geq \frac{\sum_{(e,t)\in \mathcal{C}_{e,i}} \omega_{e,t}}{\omega_{e,t_i}} \geq \frac{\sum_{(e,t)\in C_{e,i}} \omega_{e,t}}{W_{e,i}}.$$

Taking the above inequality into (3), we obtain

$$|\mathsf{Cover}_\delta(C)| \leq \delta \sum_{e\in E_c} \sum_{i=1}^{n_e} \sum_{(e,t)\in C_{e,i}} \omega_{e,t} = \delta \sum_{(e,t)\in C} \omega_{e,t}.$$

The last equality holds because $C = \bigcup_{e\in E_c} \bigcup_{i=1}^{n_e} C_{e,i}$ and any two sets in the collection $\{C_{e,i} | e \in E_c, 1 \leq i \leq n_e\}$ do not intersect.

With the above lemma, we are ready to prove the approximation ratio for the min-weight algorithm. Suppose C_{ALG} is the set of contacts disabled by the solution of the min-weight algorithm and C^* is the set of contacts disabled by the optimal solution to δ-MINCUT. Then according to Lemma 2, we have

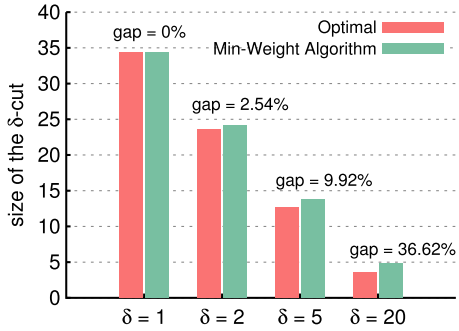


Fig. 8. Comparison between the min-weight algorithm (Algorithm 2) and the optimal result to δ-MINCUT.

$$|\mathsf{Cover}_\delta(C_{ALG})| \leq \delta \sum_{(e,t)\in C_{ALG}} \omega_{e,t}.$$

Since the min-weight algorithm first finds the minimum number of 1-removals that can disconnect the source-destination pair in the weighted time-varying graph, we have

$$\sum_{(e,t)\in C_{ALG}} \omega_{e,t} \leq \sum_{(e,t)\in C^*} \omega_{e,t}.$$

This implies that

$$|\mathsf{Cover}_\delta(C_{ALG})| \leq \delta \sum_{(e,t)\in C^*} \omega_{e,t} \leq \delta |\mathsf{Cover}_\delta(C^*)|,$$

where the last inequality is due to the lower bound in Lemma 2. Therefore, δ-approximation is achieved by the min-weight algorithm. □

Numerical Results for the Min-Weight Algorithm. The simulation setting is the same as that used for Algorithm 1. Fig. 8 shows the comparison between the min-weight algorithm (Algorithm 2) and the optimal solution to δ-MINCUT. We notice that the min-weight algorithm is close to the optimum: the approximation gap[2] is less than 10 percent for a relatively small value of δ; in particular, the approximation gap is zero when $\delta = 1$. The final observation is that the approximation gap becomes larger with the increase in δ; this tendency is consistent with the theoretical approximation ratio of δ.

5 Application in Bus Communication Networks

In this section, we demonstrate how to use our survivability framework to facilitate the design of robust networks in practice. To be more specific, we exploit δ-disjoint journeys to design a *survivable routing* protocol for a real-world bus communication network [2]. Each bus in the network has a predesigned route and is equipped with an 802.11 radio that constantly scans for other buses. Since the route of each bus is designed in advance, we can make a *coarse prediction* about bus mobility and the evolution of their communication topology. As a result, we can convert this bus communication network into a time-varying graph whose topology changes according to the estimated bus mobility. However, the prediction may be imperfect due to various reasons such as unexpected obstacles, traffic accidents, traffic jam, etc. The

2. The approximation gap is calculated by $\frac{\text{ALG}-\text{OPT}}{\text{OPT}}$.

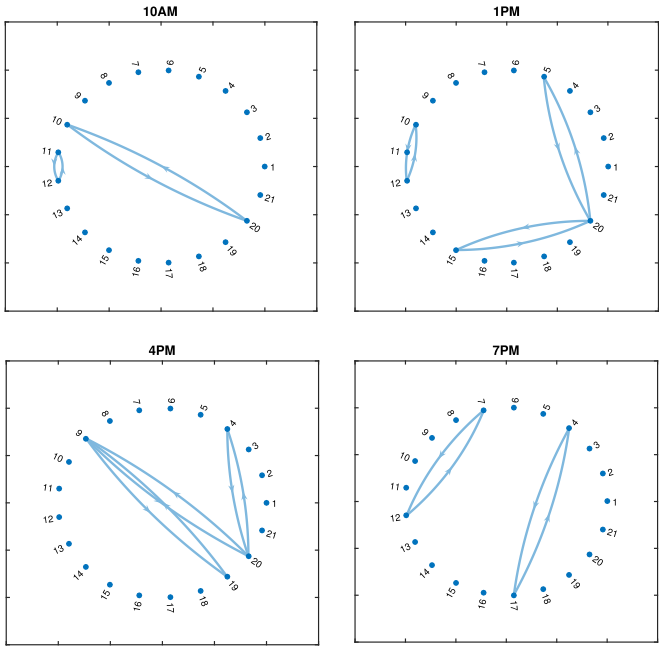


Fig. 9. Snapshots for the topology of the bus communication network. Note that the position of each node does not correspond to its physical location.

goal of survivable routing is to reduce the packet loss rate due to these unpredictable failures. In the rest of this section, we first present the design of the survivable routing protocol using δ-disjoint journeys. Then we discuss trace statistics, simulation settings and results.

5.1 Survivable Routing Protocol: DJR

The basic idea of this protocol is to replicate each packet at the source and send these copies along multiple δ-disjoint journeys obtained by solving δ-MAXFLOW. When at least one of these copies reaches the destination, the original packet is successfully delivered. This replication-based protocol is referred to as Disjoint-Journey Routing (DJR).

5.2 Traces

We use the trace from UMassDieselNet [2] where a public bus transportation system was operated around Amherst, Massachusetts. The trace records the contacts among 21 buses in nine days, which roughly reflects bus mobility over the predesigned bus routes. We use such contact information as a coarse prediction for the states of bus-to-bus links in the nine-day period. However, we assume that the prediction is imperfect and unpredictable failures may disable these contacts (the failure model will be introduced in the next section).

Fig. 9 shows the snapshots of the network topology at different instants on Day 1. Note that the position of each node does not correspond to its physical location due to the lack of geographical information. It can be observed that this bus communication network is sparsely and intermittently connected: at each point of time, there exist very few contacts in the network and the topology could be very different at different instants. This highlights the difficulty of exploiting spatially-disjoint paths for robust transmission as in [26], [27].

Fig. 10 illustrates the statistics of the bus communication network and reveals two important features of the network. The first is the "bursty" structure of contacts between any two

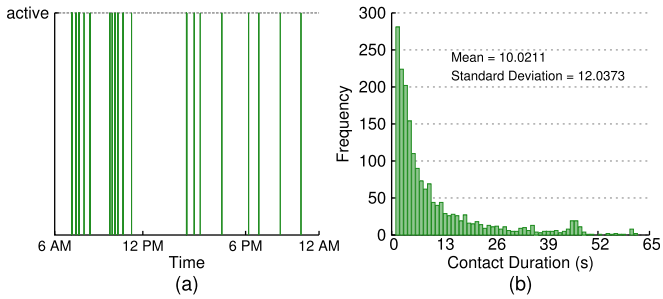


Fig. 10. Statistical structures of the bus communication network. (a) The bursty pattern of the contacts between a typical pair of buses. (b) Histogram for contact durations. Most contacts only last for a short period of time.

buses; that is, buses only communicate with each other occasionally. Fig. 10a illustrates such a bursty structure for a typical pair of buses. The second observation is that most connections in this network last for only a short period of time. As is shown in Fig. 10b, most contacts span less than 20s.

5.3 Simulation Settings

In our simulation, the slot length is identical to the trace resolution, i.e., one second. According to the measurement in [2], the average transmission rate is about 1.64 Mbps. If the packet size is set to be 1 KB, the transmission time of one packet is nearly negligible as compared to the slot length, which implies zero link-traversal delay. Each packet has a deadline (DDL) after which it will be dropped from the network; naturally, the packet deadline can be modeled by the time horizon T of the corresponding time-varying graph. A packet is generated between a random source-destination pair immediately after the previous packet expires or gets delivered. In addition, at most n copies are allowed, meaning that we can use at most n δ-disjoint journeys to send these copies. Algorithm 1 is used to compute δ-disjoint journeys.

Since it is impossible to precisely predict future topology changes, we impose random failures on the time-varying graph generated from the trace. For each link, we let failures occur in each slot with a certain probability p, and the duration of each failure is uniformly distributed within $[0, d]$ seconds. The performance metric is the packet loss rate, i.e., the fraction of packets that fail to reach the destination before the deadline.

5.4 Total Number of δ-Disjoint Journeys

We first look at the maximum number of δ-disjoint journeys in the bus communication network (Fig. 11). First, it can be observed that there exist very few δ-disjoint journeys in this network: less than three δ-disjoint journeys when $\delta \geq 5$. Particularly, only one δ-disjoint journey exists when δ is relatively large, which means that it is almost impossible to find even two journeys that are spatially disjoint (i.e., $\delta = T$). This observation indicates the lack of spatial connectivity in this bus network and implies the inefficiency of traditional Disjoint-Path Routing in networks with intermittent connectivity since such a protocol only relies on spatial diversity. Second, we can observe the diminishing return for the number of δ-disjoint journeys: beyond a certain value of δ, the increase of δ no longer reduces the number of δ-disjoint journeys. Such a tendency is due to the bursty contact structure in this network (see Section 5.2). The final observation is that extending the packet deadline increases the total

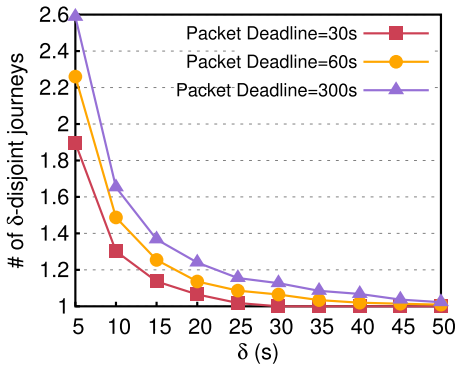


Fig. 11. The total number of δ-disjoint journeys in the network.

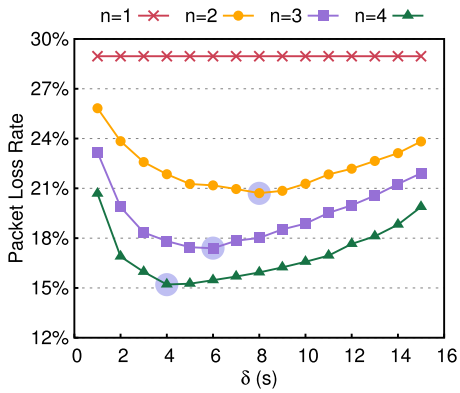


Fig. 12. Influence of n and δ on packet loss rates (DDL=300s, p=0.05, and d=60s).

number of δ-disjoint journeys since there are more transmission opportunities within a longer deadline.

5.5 Tunability of DJR

Next, we study the two-dimensional tunability of DJR (Fig. 12). We first investigate the tunability of n, i.e., the maximum number of copies we are allowed to produce or the maximum number of δ-disjoint journeys we can use. If we are allowed to use only one of the δ-disjoint journeys ($n = 1$), DJR is ineffective and the packet loss rate remains at a high level regardless of the value of δ. If we can use more δ-disjoint journeys, the packet loss rate is significantly reduced (of course, more redundant copies are created).

The influence of δ is more interesting. With the increase of δ, the packet loss rate first goes down and then increases; this tendency can be explained as follows. When δ is small, there exist many δ-disjoint journeys and we can choose any n of them to transmit copies of packets. With a fixed number of disjoint journeys, it is known that larger temporal disjointness makes the network more robust since it can survive failures of longer duration. Hence, the packet loss rate first goes down. However, the increase of δ also leads to the reduction in the number of δ-disjoint journeys (see Fig. 11); beyond a certain value of δ, the number of δ-disjoint journeys becomes smaller than n and we have to send copies over fewer than n disjoint journeys, which means that the network can survive fewer failures. Therefore, although temporal disjointness continues to grow, the reduction in the number of available disjoint journeys makes the loss rate increase. Moreover, we can observe that there exists an "optimal" value of δ which minimizes the packet loss rate (highlighted by shaded circles). In fact, this optimal value is the maximum δ such that $\mathsf{MaxFlow}_\delta \geq n$.

6 Related Work

Time-Varying Graphs. There is extensive literature seeking to define metrics for time-varying graphs, such as connectivity [15], [21], [33], distance [17], diameter [31], [32], etc. The combinatorial properties of time-varying graphs are also an active research area. For example, Kranakis et al. focused on finding connected components in a time-varying graph; Ferreira et al. investigated the complexity for computing the shortest journey [17] and the minimum spanning tree [33] (see the survey [16]).

Survivability in Time-Varying Networks. Despite the extensive research on time-varying graphs, there is very little literature on survivability of time-varying networks. The closest work to ours was done by Berman [20] and Kleinberg et al. [21]. They discussed vulnerability in so-called "edge-scheduled networks" or "temporal networks" where each link is active for exactly one slot and only permanent failures happen. Our work considers a more general graph model while leveraging the temporal features of failures, thus generalizing their results. Scellato et al. [18] investigated a similar problem in random time-varying graphs and proposed a metric called "temporal robustness". By comparison, our framework is deterministic, thus guaranteeing the worst-case survivability. Li et al. [30] studied a related but different problem in time-varying networks; specifically, they proposed heuristic algorithms to find the the min-cost subgraph of a probabilistic time-varying graph such that the probability that the subgraph is temporally connected exceeds a certain threshold.

Time-Varying Graphs and DTNs. An important application scenario of time-varying graphs is Delay Tolerant Networks (DTN), where nodes have intermittent connectivity and can only send packets opportunistically. The primary goal of DTN is to improve the packet delivery ratio via some routing schemes, and there is extensive literature in this area, such as [28], [29], [30]. In contrast, our work does not focus on any specific routing algorithm. Instead, this paper is intended to understand the inherent survivability properties of a time-varying network, which can facilitate the design of survivable routing algorithms in DTNs (e.g., Section 5).

7 Conclusion

In this paper, we propose a new survivability framework for time-varying networks, namely (n, δ)-survivability. In order to evaluate (n, δ)-survivability, two metrics are proposed: MinCut_δ and $\mathsf{MaxFlow}_\delta$. We analyze the fundamental relationship between the two metrics and show that Menger's Theorem only conditionally holds in time-varying graphs. As a result, computing both survivability metrics is NP-hard. To resolve the computational intractability, we develop several approximation algorithms. Finally, we use trace-driven simulations to demonstrate the application of our framework in a real-world bus communication network.

Acknowledgments

A shorter version of this paper [35] appeared in the *Proceedings of the IEEE International Conference on Computer Communications* (INFOCOM), San Francisco, CA, USA, April 10-15, 2016. This work was supported by US National Science Foundation Grants CNS-1116209 and AST-1547331.

REFERENCES

[1] X. Zhang, J. Kurose, B. N. Levine, D. Towsley, and H. Zhang, "Study of a bus-based disruption-tolerant network: Mobility modeling and impact on routing," in *Proc. 13th Annu. ACM Int. Conf. Mobile Comput. Netw.*, 2007, pp. 195–206.

[2] A. Balasubramanianm, B. N. Levine, and A. Venkataramani, "Enabling interactive applications for hybrid networks," in *Proc. 14th ACM Int. Conf. Mobile Comput. Netw.*, 2008, pp. 70–80.

[3] J. Mukherjee and B. Ramamurthy, "Communication technologies and architectures for space network and interplanetary Internet," *IEEE Commun. Surv. Tutorials*, vol. 15, no. 2, pp. 881–897, Apr.–Jun. 2013.

[4] S. Burleigh and A. Hooke, "Delay-tolerant networking: An approach to interplanetary internet," *IEEE Commun. Mag.*, vol. 41, no. 6, pp. 128–136, Jun. 2003.

[5] Y. Wang, H. Dang, and H. Wu, "A survey on analytic studies of delay-tolerant mobile sensor networks: Research articles," *Wirel. Commun. Mobile Comput.*, vol. 7, no. 10, pp. 1197–1208, 2007.

[6] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein, "Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with Zebranet," *ACM SIGOPS Operating Syst. Rev.*, vol. 36, no. 5, pp. 96–107, 2002.

[7] P. Bahl, R. Chandra, T. Moscibroda, R. Murty, and M. Welsh, "White space networking with Wi-Fi like connectivity," in *Proc. ACM SIGCOMM Conf. Data Commun.*, 2009, pp. 27–38.

[8] B. Radunovic, R. Chandra, and D. Gunawardena, "Weeble: Enabling low-power nodes to coexist with high-power nodes in white space networks," in *Proc. 8th Int. Conf. Emerging Netw. Exp. Technol.*, 2012, pp. 205–216.

[9] S. Rangan, T. Rappaport, and E. Erkip, "Millimeter wave cellular wireless networks: Potentials and challenges," *in Proc. IEEE*, vol. 102, no. 3, pp. 366–385, 2014.

[10] Y. Gu and T. He, "Dynamic switching-based data forwarding for low-duty-cycle wireless sensor networks," *IEEE Trans. Mobile Comput.*, vol. 10, no. 12, pp. 1741–1754, Oct. 2011.

[11] L. Chen et al., "Group-based discovery in low-duty-cycle mobile sensor networks," in *Proc. IEEE Commun. Soc. Conf. Sensor Mesh Ad Hoc Commun. Netw.*, 2012, pp. 542–550.

[12] Federal Communications Commission, "In the Matter of unlicensed operation in the TV broadcast bands," *FCC Order DA 11–131*, 2011.

[13] C. Song, Z. Qu, N. Blumm, and A. Barabasi, "Limits of predictability in human mobility," *Science*, vol. 327, pp. 1018–1021, 2010.

[14] V. Srinivasan, M. Motani, and W. T. Ooi, "Analysis and implications of student contact patterns derived from campus schedules," in *Proc. 12th Annu. Int. Conf. Mobile Comput. Netw.*, 2006, pp. 86–97.

[15] J. Whitbeck, M. Amorim, V. Conan, and J. Guillaume, "Temporal reachability graphs," in *Proc. 18th Annu. Int. Conf. Mobile Comput. Netw.*, 2012, pp. 377–388.

[16] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro, "Time-varying graphs and dynamic networks," *Ad-hoc, Mobile Wireless Netw.*, vol. 6811, pp. 346–359, 2011.

[17] B. Xuan, A. Ferreira, and A. Jarry, "Computing the shortest, fastest, and foremost journeys in dynamic networks," *Int. J. Found. Comput. Sci.*, vol. 14, pp. 267–285, 2003.

[18] S. Scellato, I. Leontiadis, C. Mascolo, P. Basuy, and M. Zafer, "Evaluating temporal robustness of mobile networks," *IEEE Trans. Mobile Comput.*, vol. 12, no. 1, pp. 105–117, Jan. 2013.

[19] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization*. Belmont, MA, USA: Athena Sci., 1997.

[20] K. A. Berman, "Vulnerability of scheduled networks and a generalization of Menger's theorem," *Networks*, vol. 28, pp. 125–134, 1996.

[21] D. Kempe, J. Kleinberg, and A. Kumar, "Connectivity and inference problems for temporal networks," in *Proc. 32nd Annu. ACM Symp. Theory Comput.*, 2000, pp. 504–513.

[22] V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, and M. Yannakakis, "Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems," in *Proc. 31st Annu. ACM Symp. Theory Comput.*, 1999, pp. 19–28.

[23] S. Neumayer, G. Zussman, R. Cohen, and E. Modiano, "Assessing the vulnerability of the fiber Infrastructure to disasters," *IEEE/ACM Trans. Netw.*, vol. 19, no. 6, pp. 1610–1623, Dec. 2011.

[24] Y. Dinitz, "Algorithm for solution of a problem of maximum flow in a network with power estimation," *Doklady Akademii Nauk SSSR*, vol. 11, no. 4, pp. 1277–1280, 1970.

[25] S. Fortune, J. Hopcroft, and J. Wyllie, "The directed subgraph homeomorphism problem," *Theoretical Comput. Sci.* vol. 10, pp. 111–121, 1980.

[26] A. Srinivas and E. Modiano, "Minimum energy disjoint path routing in wireless Ad Hoc networks," in *Proc. 9th Annu. Int. Conf. Mobile Comput. Netw.*, 2003, pp. 122–133.

[27] G. Kuperman and E. Modiano, "Disjoint path protection in multihop wireless networks with interference constraints," in *Proc. IEEE Globecom*, 2014, pp. 4472–4477.

[28] S. Jain, M. Demmer, R. Patra, and K. Fall, "Using redundancy to cope with failures in a delay tolerant network," in *Proc. Conf. Appl. Technol. Archit. Protocols Comput. Commun.*, 2005, pp. 109–120.

[29] S. Jain, K. Fall, and R. Patra, "Routing in a delay-tolerant network," in *Proc. Conf. Appl. Technol. Archit. Protocols Comput. Commun.*, 2004, pp. 145–158.

[30] F. Li, S. Chen, M. Huang, Z. Yin, C. Zhang, and Y. Wang, "Reliable topology design in time-evolving delay-tolerant networks with unreliable links," *IEEE Trans. Mobile Comput.*, vol. 14, no. 6, pp. 1301–1314, Jun. 2015.

[31] A. Chaintreau, A. Mtibaa, L. Massoulie, and C. Diot, "The diameter of opportunistic mobile networks," in *Proc. ACM CoNEXT Conf.*, 2007, Art. no. 12.

[32] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: Densification laws, shrinking diameters and possible explanations," in *Proc. 11th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2005, pp. 177–187.

[33] A. Ferreira and A. Jarry, "Minimum-energy broadcast routing in dynamic wireless networks," *J. Green Eng.*, vol. 2, no. 2, pp. 115–123, 2012.

[34] L. W. Beineke, "Characterizations of derived graphs," *J. Combinatorial Theory*, vol. 9, no. 2, pp. 129–135, 1970.

[35] Q. Liang and E. Modiano, "Survivability in time-varying networks," in *Proc. 35th Annu. IEEE Int. Conf.*, 2016, pp. 1–9.

Qingkai Liang received the BE degree (with the highest honor) from the Department of Electronic Engineering, Shanghai Jiao Tong University, China, in 2013 and the MS degree from MIT, in 2015. He is working toward the graduate degree in the Laboratory for Information and Decision Systems, MIT. His research interests include the theory of network control, optimization, and machine learning.

Eytan Modiano received the BS degree in electrical engineering and computer science from the University of Connecticut, Storrs, in 1986 and the MS and PhD degrees, both in electrical engineering, from the University of Maryland, College Park, Maryland, in 1989 and 1992, respectively. He was a Naval Research Laboratory fellow between 1987 and 1992 and a National Research Council Post doctoral fellow during 1992-1993. Between 1993 and 1999, he was with MIT Lincoln Laboratory. Since 1999, he has been on the faculty with MIT, where he is a professor in the Department of Aeronautics and Astronautics and the Laboratory for Information and Decision Systems (LIDS). His research is on communication networks and protocols with emphasis on satellite, wireless, and optical networks. He received the MobHoc 2016 Best Paper Award, the Wiopt 2013 Best Paper Award, and the Sigmetrics 2006 Best Paper Award. He is an editor-at-large for the *IEEE/ACM Transactions on Networking*, and served as an associate editor for the *IEEE Transactions on Information Theory* and the *IEEE/ACM Transactions on Networking*. He was the technical program cochair for IEEE Wiopt 2006, IEEE Infocom 2007, ACM MobiHoc 2007, and DRCN 2015. He is a fellow of the IEEE and an associate fellow of the AIAA, and served on the IEEE fellows committee.