Survivability of Time-varying Networks

by

Qingkai Liang

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2015

Signature redacted

Author ....
Department of Aeronautics and Astronautics
May 13, 2015

Signature redacted

Certified by ....
Eytan Modiano
Professor, Department of Aeronautics and Astronautics
Thesis Supervisor

Signature redacted

Accepted by ....
Paulo C. Lozano
Associate Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Survivability of Time-varying Networks

by

Qingkai Liang

Submitted to the Department of Aeronautics and Astronautics
on May 13, 2015, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

Time-varying graphs are a useful model for networks with dynamic connectivity such as mmWave networks and vehicular networks, yet, despite their great modeling power, many important features of time-varying graphs are still poorly understood. In this thesis, 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 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 several 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.

Thesis Supervisor: Eytan Modiano
Title: Professor, Department of Aeronautics and Astronautics

Acknowledgments

First and foremost, I would like to express my gratitude to my research advisor Professor Eytan Modiano for his invaluable guidance during the past two years. His encouragement has always been the driving force for me when I encounter difficulties, both in my research and in my daily life. He is such an experienced advisor that can always shed light on my research directions and tell me to do the right things at the right time. He is very patient to discuss various research problems and share inspiring ideas. His detailed comments on this thesis greatly improved my academic writing skills as well as the quality of this thesis.

Second, I would like to thank my colleagues in Communications and Networking Research Group (CNRG): Chih Ping Li, Georgios Paschos, Matt Johnston, Marzieh Parandehgheibi, Anurag Rai, Abhishek Sinha, Jianan Zhang, and Thomas Stahlbuhk. They provide me with lots of insightful feedback on this research, which directly contributes to the improvement of several chapters in this thesis.

Third, my appreciation goes to my roommates: Fangchang Ma, Yili Qian, and Rujian Chen. They not only share their ideas on this thesis but also bring a lot of happiness to my daily life at MIT.

Fourth, I would like to thank my parents whose love and support have always been the most important source of energy when I pursue my career dream.

Finally, I would like to express my thanks to National Science Foundation (NSF Grant CNS-1116209) for its generous support for this research.

Contents

List of Figures

List of Tables

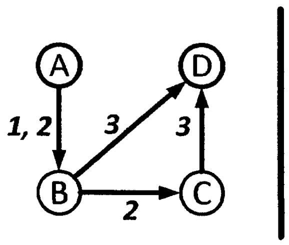
Chapter 1

Introduction

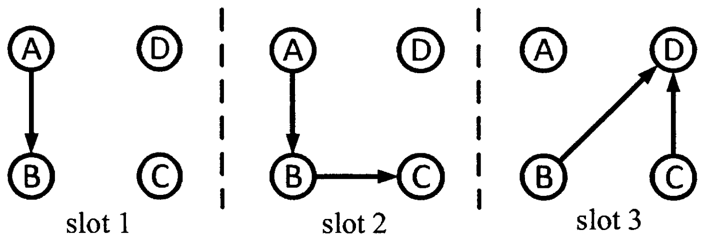
1.1 Background and Motivations

Time-varying graphs have emerged as a useful model for networks with time-varying connectivity, especially in the context of communication networks. For example, in vehicular networks, the network connectivity changes over time with the movement of vehicles; in whitespace networks [16] [17] [18], the state of secondary links will switch between ON/OFF with primary users' channel reclamation/release; in millimeter-wave (mmWave) networks [22], the network topology changes with the adjustment of beam directions of directional antennas. In Figure 1-1, we illustrate a simple time-varying graph and its snapshots within 3 time slots.

In many applications of time-varying networks, transmission reliability is of a great concern. For example, it is crucial to provide robustness against unexpected shadowing in mmWave networks; it is also critical to guarantee transmission reliability in a vehicular network if it is used for emergency surveillance. Unfortunately, time-varying networks are particularly vulnerable due to their constantly changing topology that results from either *predictable* or *unpredictable* interruptions. Predictable interruptions (or *intrinsic* interruptions) usually follow the nature of networks, such as node mobility in mobile networks. In contrast, unpredictable interruptions, such as unexpected obstacles, bring in uncertainty and may greatly degrade network performance. Therefore, the goal of this thesis is to understand the robustness of time-varying networks against unpredictable interruptions (also



(a) A time-varying graph



(b) Snapshots of the time-varying graph

Figure 1-1: (a) The 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.

referred to as **failures**) while treating those predictable interruptions as an inherent feature of the network topology.

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 while a hardware malfunction may influence the network for several days. By comparison, the traditional definition of survivability is intended for a static environment and fails to account for links reappearing. Duration of failures has a crucial impact on the performance of time-varying networks; for example, in the time-varying network shown in Figure 1-1, an 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 $\to$ 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 the three slots.

1.2 Contributions

This thesis tackles the above non-trivial temporal factors by proposing a new survivability framework for time-varying networks. Our framework captures both the *number* and the *duration* of failures, regardless of *where* and *when* failures occur.

Chapter 2 formally introduces the model of time-varying graphs and develops a useful tool called the Line Graph which bridges time-varying and static graphs.

Chapter 3 proposes 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, the proposed framework can generalize many existing survivability models. We further propose two metrics, namely MinCut_δ and $\mathsf{MaxFlow}_\delta$, in order to assess robustness of time-varying networks. Moreover, this chapter presents some graph-theoretic results that highlight the difference between static and time-varying graphs. For example, it is shown that some fundamental survivability-related results such as Menger's Theorem[1] only conditionally hold for time-varying graphs.

Due to the difference between static and time-varying graphs, the evaluation of survivability becomes very challenging in time-varying networks. In Chapter 4, we analyze the complexity of computing the proposed survivability metrics and develop several approximation algorithms with provable approximation ratios.

Chapter 5 presents detailed discussions on applications and extensions of the proposed framework.

Chapter 6 applies the proposed framework in the design of survivable routing protocols for vehicular networks. Real-world traces are used to validate the proposed design. It is

[1] In graph theory, Menger's Theorem is a special case of the maxflow-mincut theorem, which states that the maximum number of edge (node) disjoint paths equals to the size of the minimum edge (node) cut.

shown that our survivability framework has strong modeling power and is more suitable for time-varying networks than existing approaches.

1.3 Related Work

Time-varying Graphs. With the advent of many dynamic networks (e.g., mobile networks), time-varying graphs have been recognized as a powerful modeling tool. There is extensive literature seeking to define graphical metrics for time-varying graphs, such as connectivity [12,26], distance [6,27], centrality [13,20], diameter [5,15], 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 and algorithms for computing the shortest journey [27] (see the survey [4] for more details).

Survivability in Time-varying Networks. Interestingly, despite the extensive research on time-varying graphs, there is very little literature on survivability of time-varying networks. The closest work to this thesis was done by Berman [3] and Kleinberg *et al.* [12]. They discussed network vulnerability for so-called "edge-scheduled networks" or "temporal networks" where each link is active for exactly one slot and only permanent failures happen. By comparison, this research considers a more general graph model while leveraging the temporal features of failures, thus generalizing their results. Scellato *et al.* [23] investigated a similar problem on random time-varying graphs and proposed a metric called temporal robustness. By comparison, the framework proposed in this thesis is deterministic, thus guaranteeing the worst-case survivability.

Time-varying Graph and DTN 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 [10,11,19,25]. Different from DTN literature, this work does not focus on any specific routing algorithms. Instead, this thesis is intended to understand the inherent survivability of a time-varying network, which can facilitate the design of survivable routing algorithms in DTN (as will be demonstrated in Chapter 6). As a result, the framework proposed in this thesis is complementary to previous DTN papers.

Chapter 2

Model of Time-varying Graphs

In this chapter, we formalize the model of time-varying graphs and introduce some important terminology and assumptions that will be frequently used throughout this thesis. 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 [4], 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 (e.g., queuing delay). However, it is trivial to transform node-related functions to edge-related functions by node splitting (see Chapter 5.1 for details); thus, it suffices to consider the above edge-version characterization. Similarly, the above definition only captures directed graphs since the undirected case can be transformed to the directed case (also see Chapter 5.1).

In this thesis, we consider a *discrete and finite* time span, i.e., $\mathcal{T} = \{1, 2, \cdots, T\}$, 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 (or the resolution time) of a time-varying graph is arbitrary as long as it can capture topology changes in sufficient granularity. For example, if link states change frequently due to fast fading, the slot length needs to be very small (milliseconds) to capture such rapid variations; in contrast, if the network topology changes due to node mobility, the slot length may be much larger (seconds or minutes). In general, a smaller slot length yields more modeling accuracy but increases computational complexity.

Under the discrete-time model, the edge-delay function ζ can take values from $\mathbb{N} = \{0, 1, \cdots\}$. Note that *zero* delay means that the time used for crossing an edge is negligible as compared to the slot length. In the rest of this thesis, we consider the case where edge delay is one slot, i.e., $\zeta(e, t) = 1$ for any $e \in E$ and $t \in \mathcal{T}$, and it is trivial to extend the analysis to arbitrary traversal time.

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 satellite network with known orbits, in a whitespace network with planned channel reclamation, in a mmWave network with scheduled beam steering, in a sensor network with pre-designed sleep-wake patterns, etc. In contrast, *unpredictable* topology changes (also referred to as **failures** in this thesis) may include those caused by unexpected shadowing, unscheduled channel reclamation, hardware malfunctions, etc. Note that this model does not require perfect predictions on future topology changes since any prediction errors can be treated as failures.

Finally, we would like to highlight the modeling power of time-varying graphs. In theory, this time-varying graph model generalizes many types of graphs. For example, it generalizes the notion of static graphs since we can view a static graph as a time-varying graph where the time horizon is only one slot ($T = 1$) and the slot length is infinite. This model also generalizes the so-called "edge-scheduled graphs" [3] or "temporal graphs" [12] which can be reduced to a time-varying graph where each edge is active for only one slot. In practice, they can be used to model almost all types of networks with topology changes. Examples include (i) mobile networks (e.g., vehicular or satellite networks), where the network topology varies due to node mobility or scheduled obstacles, (ii) whitespace networks where link states change due to primary users' channel reclamation/release, (iii) mmWave networks with tunable directional antennas where the network topology varies with the dynamic adjustment of beam directions.

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 Figure 1-1, there exists a contact $(\mathsf{AB}, 1)$, showing that link $\mathsf{A} \rightarrow \mathsf{B}$ is active in slot 1.

Definition 3 (Journey [26]). *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) shows that intermediate edges must 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 Figure 1-1: $(\mathsf{AB}, 1) \rightarrow (\mathsf{BC}, 2) \rightarrow (\mathsf{CD}, 3)$ 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 Figure 1-1, meaning that a message from A can reach D within $T = 3$ slots.

2.3 A Useful Tool: Line Graph

In this section, we introduce a useful tool which transforms 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* [2] which illustrates the adjacency between edges. The difference here is that we also need to consider 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.

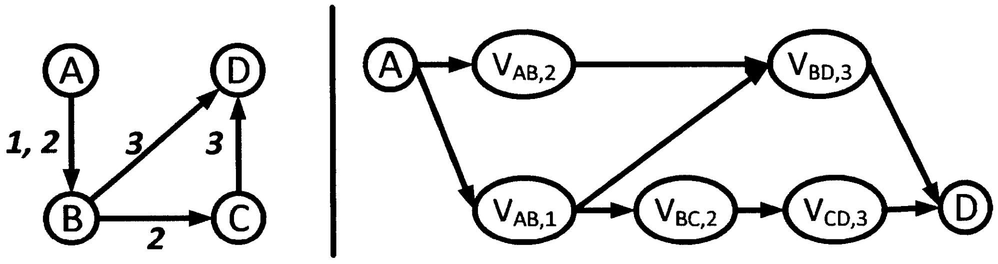
- Add a directed edge from node v_{e_1,t_1} to node v_{e_2,t_2} in the Line Graph if $(e_1, t_1) \rightarrow (e_2, t_2)$ is a feasible journey from $\mathsf{start}(e_1)$ to $\mathsf{end}(e_2)$. Also, add an edge from node s to node $v_{e,t}$ if $\mathsf{start}(e) = s$, and add an edge from node $v_{e,t}$ to node d if $\mathsf{end}(e) = d$.

Note that the construction of the Line Graph only relies on the original time-varying graph and the source-destination pair. An example of the Line Graph is shown in Figure 2-1.

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 Figure 2-1, 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.*

Observation 1 implies that we can handle many journey-related problems in a time-varying graph by looking at paths in the corresponding Line Graph. This allows us to apply path-related results in static graphs, such as Menger's Theorem and Dijkstra Algorithm, to help solve journey-related problems in time-varying graphs.



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

Figure 2-1: Illustration of the Line Graph (src: A, dst: D).

Chapter 3

Survivability Model and Metrics

In this chapter, we begin to investigate robustness of time-varying networks. Specifically, we are interested in their resilience against *unpredictable interruptions* (i.e., failures) such as unscheduled channel reclamation, unexpected obstacles, 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 a challenging task to evaluate robustness of time-varying networks.

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 slots.*

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) in the rest of this thesis.

Discussion on (n, δ)-Survivability: First, 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 any 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.

Second, (n, δ)-survivability is a generalized definition. For example, if $\delta = T$ holds[1], then (n, δ)-survivability reflects the number of *permanent failures* the network can tolerate, which becomes the conventional notion of survivability we use in static networks.

Finally, it should also be mentioned that failures can be either link failures or node failures. As will be shown in Chapter 5.1, node failures can be equivalently converted to link failures, so we will consider link failures unless otherwise stated. Also, failures are assumed to occur at the beginning of slots and last for multiples of the slot length. In Chapter 5.2, we will demonstrate how to accommodate failures happening at arbitrary instants and lasting for arbitrary duration.

[1]In our model, the time span of a time-varying graph is at most T, so $\delta = T$ is enough to imply permanent failures.

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 to 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 survivability metrics for time-varying networks. The fundamental relationship between the two metrics will be discussed in Chapter 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.*

Intuitively, a δ-removal just corresponds to a link failure that lasts for δ slots. For the simplicity of notations, if a δ-removal disables link e in slots $t, t+1, \cdots, t+\delta-1$, we say that contact (e,t) is the **Removal Head** of this δ-removal, meaning that this δ-removal starts to block link e in slot t. Any δ-removal can be uniquely identified by its removal head.

Definition 8 (δ-cut). *A δ-cut (denoted by CUT_δ) 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 survivability metric for time-varying networks, 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-slot failures required to disconnect the time-varying network. If the number of δ-slot failures is strictly fewer than MinCut_δ, the destination is still reachable from the source. 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. Finally, MinCut_δ has many applications in practical problems, which will be discussed in details in Chapter 5.4.

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

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

Here, C is the set of contacts in the time-varying graph and $y_{e,t}$ is a binary decision variable indicating whether contact (e,t) is chosen to be a δ-removal head (note that a δ-removal head uniquely identifies a δ-removal). $\mathcal{J}_{sd}$ is the set of feasible journeys from source s to destination d. For any $J \in \mathcal{J}_{sd}$, we define $R(\delta,J)$ as the set of contacts such that if any of these contacts is selected to be a δ-removal head, then at least one contact used by journey J will be disabled, 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 disabled by at least one of the selected δ-removals, such that d is not reachable from s.

This formulation is concise but has an exponential number of constraints because the number of possible journeys is exponential in the number of contacts. In Appendix A, we present a compact ILP formulation which is less intuitive. The complexity and the algorithm for solving the above ILP will be further discussed in Chapter 4.2.

3.2.2 Survivability Metric: $\mathsf{MaxFlow}_\delta$

The second survivability metric, namely $\mathsf{MaxFlow}_\delta$, 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 $t \in \mathcal{T}$, 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 on any edge within the window. Figure 3-1 gives an example of δ-disjoint journeys for the cases where $\delta = 1$ and $\delta = 2$.

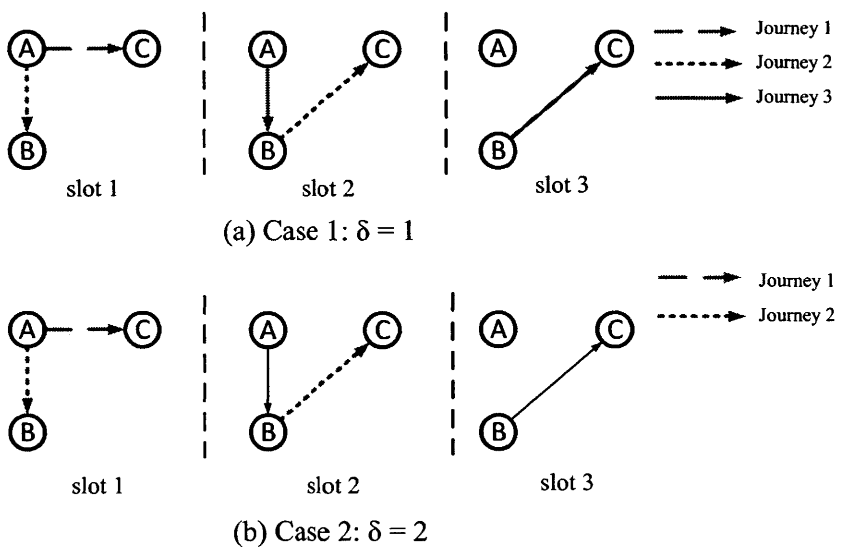


Figure 3-1: Illustration of δ-disjoint journeys. The source-destination pair is (A, C). (a) When $\delta = 1$, any two different δ-disjoint journeys cannot use the same link within the same slot, and there are three δ-disjoint journeys. (b) When $\delta = 2$, only two δ-disjoint journeys exist since any link cannot be used by two δ-disjoint journeys within 2 slots. For example, link $\mathsf{A} \to \mathsf{B}$ has been used by Journey 2 in slot 1, so any other δ-disjoint journey cannot use this link in slot 1 or 2.

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 $\mathsf{MaxFlow}_\delta$.

Definition 11 ($\mathsf{MaxFlow}_\delta$). $\mathsf{MaxFlow}_\delta$ *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 $\mathsf{MaxFlow}_\delta$ (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 a bus communication network (see Chapter 6 for details), we will see that a time-varying network may not have any spatially-disjoint paths. Thus, MaxFlow proposed in static networks is not effective in time-varying networks. By comparison, $\mathsf{MaxFlow}_\delta$ considers δ-disjoint journeys, which allows for *temporal disjointness.* Moreover, $\mathsf{MaxFlow}_\delta$ generalizes MaxFlow since we can simply set $\delta = T$ such that δ-disjoint journeys become spatially disjoint.

Second, $\mathsf{MaxFlow}_\delta$ not only gives us a measure of network survivability but also tells us how to achieve such survivability. The idea is similar to the Disjoint-Path Protection in static networks [24] [14], 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., $\mathsf{MaxFlow}_\delta \geq n$), the transmission can survive any $n-1$ failures that last for δ slots and is thus $(n-1, \delta)$-survivable.

Formulation: $\mathsf{MaxFlow}_\delta$ 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 decision variable indicating whether journey J is added to the set of

δ-disjoint journeys. All the other notations have the same meanings as in the formulation of MinCut_δ. The first constraint checks every edge and forces this edge to be used by at most one of the δ-disjoint journey in any time window of δ slots. The above formulation also has an exponential number of constraints in the number of contacts. A compact formulation is shown in Appendix A. The complexity and the algorithms for solving the above ILP will be further investigated in Chapter 4.1.

3.3 Analysis of Metrics

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

$$\mathsf{MaxFlow}_\delta \le \mathsf{MaxFlow}_\delta^\mathsf{R} = \mathsf{MinCut}_\delta^\mathsf{R} \le \mathsf{MinCut}_\delta.$$

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

Theorem 1. *In a time-varying graph $\mathcal{G}$, Menger's Theorem holds* ($\mathsf{MaxFlow}_\delta = \mathsf{MinCut}_\delta$) *if $\delta = 1$.*

Proof. 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 $\mathcal{G}$ (see Chapter 2.3 for details) 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: Lemma 1 does not holds for $\delta \geq 2$. For example, if $\delta = 2$, there is only one δ-disjoint journey in Figure 2-1(a) 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 finally have $\mathsf{MaxFlow}_1 = \mathsf{MinCut}_1$. □

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 (see Chapter 2.3 for details) 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 following theorem shows that, in general, Menger's Theorem may break down.

Theorem 2. *For any $\delta \geq 2$, there exist instances such that $\mathsf{MaxFlow}_\delta \neq \mathsf{MinCut}_\delta$. Moreover, the gap ratio $\frac{\mathsf{MinCut}_\delta}{\mathsf{MaxFlow}_\delta}$ can grow without bound.*

Proof. 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 k-th graph. The construction for $k = 1, 2, 3$ is shown in Figure 3-2. 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 correspond to $\mathcal{G}_2$. Note that the structure of $\mathcal{G}_2$ is similar to the time-varying graph $\mathcal{H}_1$ shown in Figure 3-3, and the structure of $\mathcal{G}_3$ can also be seen as an "extension" from $\mathcal{H}_1$.

We use inductions to prove $\frac{\mathsf{MinCut}_\delta}{\mathsf{MaxFlow}_\delta} = k$ for any $\delta \geq 2$ in $\mathcal{G}_k$. For brevity, we only demonstrate the induction philosophy from $\mathcal{G}_1$ to $\mathcal{G}_2$ while its formal generalization is presented in Appendix B.

• In $\mathcal{G}_1$, the source-destination pair is (s, d_1). It is obvious that $\mathsf{MaxFlow}_\delta = \mathsf{MinCut}_\delta = 1$.

• 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

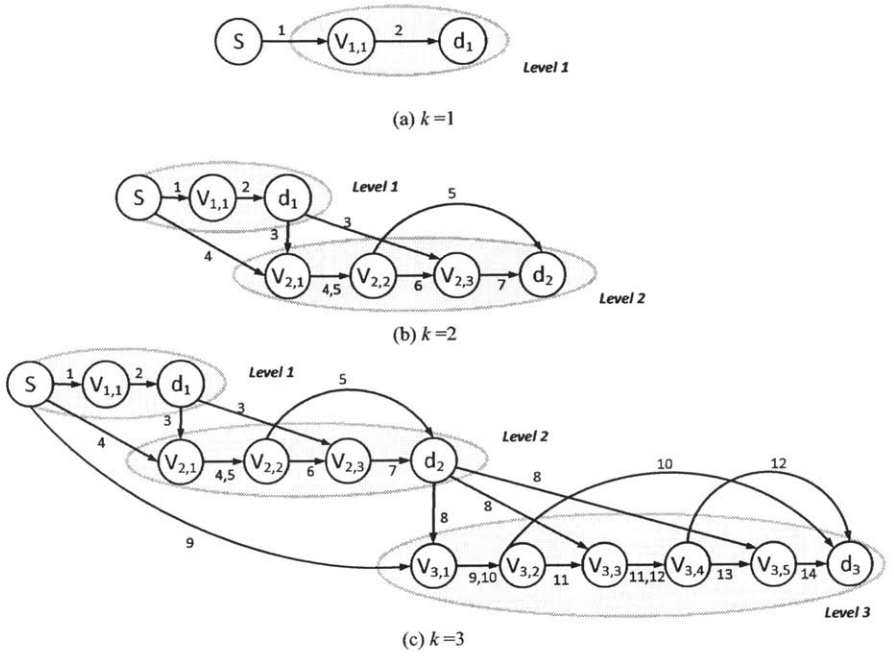
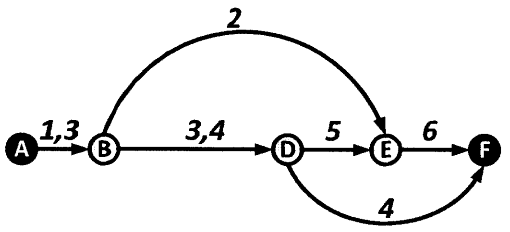


Figure 3-2: Examples used in the proof to Theorem 2. The source-destination pair is (s, d_k) in graph $\mathcal{G}_k$ ($k = 1, 2, 3$).

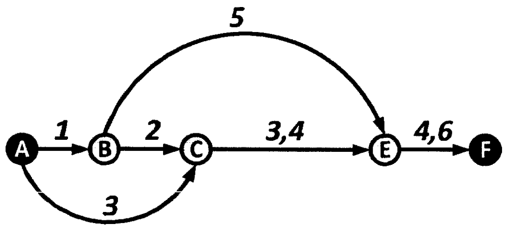
from $\mathcal{G}_1$ that there is only one δ-disjoint journey from s to d_1. For the latter choice, the only possibility is $s \to v_{2,1} \to v_{2,2} \to v_{2,3} \to d_2$ but this journey cannot be δ-disjoint of any journey in the first choice (i.e., via node d_1). Hence, there is only one δ-disjoint 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 or occurs to some "cross-level" contact (say contact $(d_1 \to v_{2,1}, 1)$ or $(d_1 \to v_{2,3}, 1)$), we can still descend to level 2 and reach d_2. If the δ-removal occurs to any "inner" contact within level 2, there is still a journey from s to d_2 via d_1 since there are two spatially disjoint journeys from d_1 to d_2. Hence, we can conclude that any single δ-removal cannot disconnect d_2 from s and $\mathsf{MinCut}_\delta = 2$.

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. □

Figure 3-3 shows two simple examples where $\mathsf{MaxFlow}_\delta \neq \mathsf{MinCut}_\delta$ for $\delta = 2$. It is not hard to see that there is only one δ-disjoint journey from A to F while $\mathsf{MinCut}_\delta = 2$ since any single δ-removal cannot disconnect F from A. In fact, the proof to Theorem 2 also shows that the two simple examples are the basic blocks for constructing the gap between $\mathsf{MaxFlow}_\delta$ and MinCut_δ. We will formalize this observation at the end of this section.



(a) Time-varying Graph $\mathcal{H}_1$



(b) Time-varying Graph $\mathcal{H}_2$

Figure 3-3: Forbidden structure $\mathcal{H}_1$ and $\mathcal{H}_2$. The source-destination pair is A $\rightarrow$ F.

Although Theorem 2 shows the existence of the gap between $\mathsf{MaxFlow}_\delta$ and MinCut_δ for $\delta \geq 2$, it does not tell us *how often* such a gap occurs. We investigate this issue through experiments over random time-varying graphs. The number of nodes is uniformly distributed within [5, 15]; the underlying topology is a random scale-free graph; the time horizon T is uniformly distributed within [2, 5] and each link is active with a random probability within [0.1, 0.9] in each slot. $\mathsf{MaxFlow}_\delta$ and MinCut_δ are computed directly through their ILP formulations.

Table 3.1 adds up $\mathsf{MaxFlow}_\delta$ and MinCut_δ in 50000 random time-varying graphs and shows the frequency of gap occurrence. Surprisingly, it turns out that the gap occurs at an extremely low frequency over these random instances. Hence, it is a rare event that MinCut_δ does not equal to $\mathsf{MaxFlow}_\delta$ for $\delta \geq 2$; this motivates us to investigate the conditions under which the gap occurs. As discussed before, $\mathcal{H}_1$ and $\mathcal{H}_2$ shown in Figure 3-3 might be fundamental to the occurrence of the gap. In the rest of this section, we present *forbidden structures* based on $\mathcal{H}_1$ and $\mathcal{H}_2$ to explain the gap. Before providing the formal result, we

Table 3.1: Frequency of Gap Occurrence

	Total MinCut_δ	Total $\mathsf{MaxFlow}_\delta$	Frequency
$\delta = 2$	153488	153484	0.008%
$\delta = 3$	133503	133493	0.02%
$\delta = 5$	117956	117940	0.032%

introduce two terms (defined over *static* graphs).

- **Edge Contraction:** The contraction of edge e is to first delete both ends of e and then build a new vertex adjacent to all vertices originally adjacent to one or both ends of e. This procedure is illustrated in Figure 3-4.
- **Topological Minor:** A graph G is a topological minor of a graph H if G can be obtained from H by contracting edges.

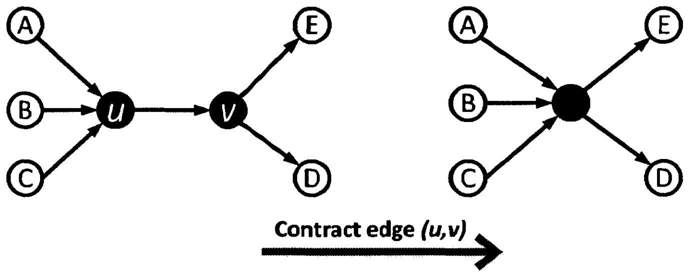


Figure 3-4: Illustration of the contraction for edge (u, v).

Now we are ready to present the following forbidden structure characterization about the gap between $\mathsf{MaxFlow}_\delta$ and MinCut_δ. Unfortunately, we are still unable to prove this result in its full generality.

Conjecture 1. *In a time-varying graph $\mathcal{G}$, Menger's Theorem holds (*$\mathsf{MaxFlow}_\delta = \mathsf{MinCut}_\delta$*) for any $\delta \geq 1$ if the Line Graph of $\mathcal{G}$ does not contain a subgraph that has either $L(\mathcal{H}_1)$ or $L(\mathcal{H}_2)$ as a topological minor, where $L(\mathcal{H}_1)$ or $L(\mathcal{H}_2)$ is the Line Graph of the time-varying graph $\mathcal{H}_1$ or $\mathcal{H}_2$ shown in Figure 3-3.*

In other words, the two forbidden structures $L(\mathcal{H}_1)$ and $L(\mathcal{H}_2)$ are the *only obstacles* to the equivalence between the two metrics. Note that we use the Line Graph instead of the original time-varying graph in the above characterization because it is the **adjacency** between edges and the **time ordering** that matter while the detailed activation slot numbers are not important. The Line Graph preserves such information and is invariant to the detailed activation slot numbers.

Conjecture 1 also shows that the gap between $\mathsf{MaxFlow}_\delta$ and MinCut_δ can only occur under very specific *spatial* and *temporal* structures. In a random time-varying graph, the probability of satisfying both the spatial and the temporal structures is very low. Hence, it is a rare event that $\mathsf{MaxFlow}_\delta$ does not equal to MinCut_δ (just as we observe in Table 3.1). More importantly, it implies that both $\mathsf{MaxFlow}_\delta$ and MinCut_δ may be efficiently computed (e.g., by solving the LP relaxations) in most time-varying graphs. In cases where the gap exists, computing the two metrics may be hard and alternative schemes should be deployed to handle these defective cases. In the next chapter, we will discuss the computational aspects to address this problem.

Chapter 4

Computational Issues

Although the previous chapter shows that both $\mathsf{MaxFlow}_\delta$ and MinCut_δ may be efficiently computed in most cases, it is still necessary to develop algorithms that can handle the computation in the general case where Menger's Theorem may break down. In this chapter, we study the computational complexity and related algorithms for evaluating survivability 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:

δ-**MAXFLOW**

Input: a time-varying network $\mathcal{G}$, a source-destination pair (s, d), and the value of δ;

Output: the value of $\mathsf{MaxFlow}_\delta$ and the corresponding set of δ-disjoint journeys.

4.1.1 Computational Complexity

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

Theorem 3. *δ-MAXFLOW is NP-hard, and it is even NP-hard to achieve $O(|E|^{1/2-\epsilon})$-approximation for any $\epsilon > 0$. Moreover, this bound is tight.*

Proof. The proof is by reducing from the Bounded-Length Edge-Disjoint Paths (BLEDP) problem [8].

- 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).
  - A positive integer L 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 the "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 the edge-traversal delay is one 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 e by l_e edges in series; each edge has one-slot delay and is active in the entire time span. An example is illustrated in Figure 4-1. 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 approximation for δ-MAXFLOW. Guruswami *et al.* [8] proved that it is NP-hard to achieve $O(|E'|^{1/2-\epsilon})$-approximation for any $\epsilon > 0$ for BLEDP. In the constructed time-varying graph, we have $|E| = \sum_{e \in E'} l_e \leq L|E'| = T|E'|$. Since T is a bounded integer, it follows that $|E'| = \Omega(|E|)$. Therefore, it is NP-hard to achieve $O(|E|^{1/2-\epsilon})$-approximation for any $\epsilon > 0$ for δ-MAXFLOW. $\square$

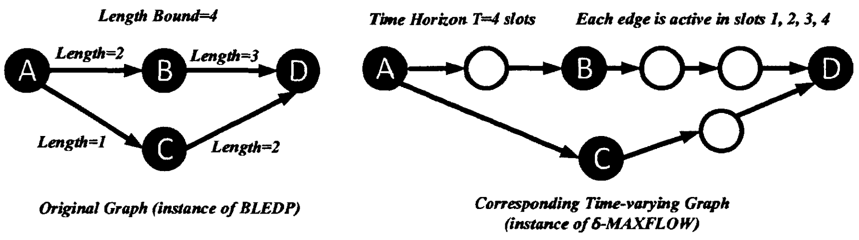

Length Bound=4
Length=2
Length=3
Length=1
Length=2
A
B
C
D
Original Graph (instance of BLEDP)
Time Horizon T=4 slots
Each edge is active in slots 1, 2, 3, 4
Corresponding Time-varying Graph (instance of δ-MAXFLOW)

Figure 4-1: Illustration of the reduction from BLEDP to δ-MAXFLOW.

Note that to prove the tightness of the inapproximability bound, we just need to find an algorithm that achieves $O(|E|^{1/2})$ approximation, which will be demonstrated in the next section.

4.1.2 Optimal Approximation Algorithm

In this section, we propose an approximation algorithm that attains the approximation lower bound in Theorem 3. 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 time-varying graph $\mathcal{G}$ and 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 $|t - t'| < \delta$.*

Intuitively speaking, 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 Chapter 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 Chapter 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 in the Line Graph. From the definition of interfering contacts, we can easily verify that the obtained journeys are δ-disjoint.

Algorithm 1 Greedy Algorithm for δ-MAXFLOW

```
Input:
    $\mathcal{G}$: the time-varying graph;
    $(s, d)$: the source-destination pair
    $\delta$: the degree of temporal disjointness;
Output:
    $J_1, \cdots, J_m$: a set of $\delta$-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.
```

Now we estimate the time complexity of this greedy algorithm. In each iteration (steps 2-8), 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 $O(|C|^3)$.

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

Theorem 4. *The greedy algorithm attains $O(\sqrt{|E|})$ approximation for δ-MAXFLOW, i.e.,* $\frac{\text{OPT}}{\text{ALG}} = O(\sqrt{|E|})$.

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 $\mathcal{J}^* = \{J_1^*, \cdots\}$ the optimal solution and $\mathcal{J} = \{J_1, \cdots\}$ 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

$$|\mathcal{J}^*| = |\mathcal{J}^*_{long}| + |\mathcal{J}^*_{short}| \leq \frac{|E|(\frac{T}{\delta}+1)}{k} + 2k \times |\mathcal{J}| \tag{4.1}$$

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. $\square$

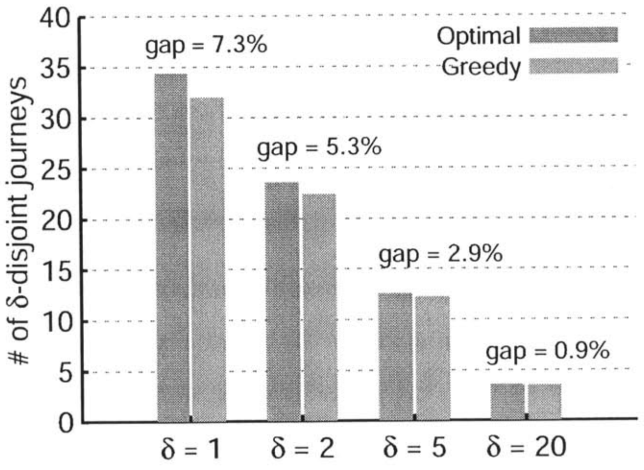


Figure 4-2: Comparison between the greedy algorithm (Algorithm 1) and the optimal solution to δ-MAXFLOW.

It is interesting to observe that the above approximation ratio attains the lower bound in Theorem 3. As a result, the greedy algorithm is the **optimal approximation algorithm** and the inapproximability bound in Theorem 3 is tight. In practice, the greedy algorithm also performs extremely well, as is demonstrated by the following numerical results.

Numerical Results on 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 picked. The optimal solution to δ-MAXFLOW is derived by directly solving its ILP formulation. Figure 4-2 shows the comparison, where the approximation gap is calculated by $\frac{\mathsf{OPT}-\mathsf{ALG}}{\mathsf{ALG}}$. We can observe that the approximation gap is less than 8% even in the worst case, much better than the theoretical bound in Theorem 4.

4.1.3 Special Case

Sometimes we may only want find a fixed number (say k) δ-disjoint journeys (called the **δ-FIXFLOW** 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 problem is still NP-hard in general. This hardness result can be easily derived from the work of Kleinberg *et al.* [12]. They showed that a special case of δ-FIXFLOW 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 problem is also NP-hard. However, the celebrated news is that δ-FIXFLOW is polynomial-time solvable if the underlying graph is a Directed Acyclic Graph (DAG) (note that the Line Graph of such a underlying graph is also a DAG). This is achieved through the modification of classic *Pebbling Games* [7]. The detailed procedures are shown as the following

Suppose we are given a fixed integer $k > 0$ and need to find k δ-disjoint journeys. Denote $L(\mathcal{G})$ the Line Graph of *the input time-varying graph* $\mathcal{G}$ and $L(G)$ 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 $u \rightarrow v$ 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 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 Theorem 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 5.

Theorem 5. *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 $P_1, \cdots, P_k$ in the line graph, which corresponds to k journeys in the original time-varying graph: $J_1, \cdots, J_k$. 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 $l_e > l_{e'}$, which causes 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 [12]. 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:

Input: a time-varying network $\mathcal{G}$, a source-destination pair (s, d), and the value of δ;
Output: the smallest δ-cut CUT^*_δ, and $\mathsf{MinCut}_\delta = |\mathsf{CUT}^*_\delta|$.

In the rest of this section, we first study the complexity of δ-MINCUT and then present a naive algorithm as well as its heuristic improvement.

4.2.1 Computational Complexity

Similar to δ-MAXFLOW, the computation of MinCut_δ for an arbitrary value of δ is also intractable, as is shown in Theorem 6.

Theorem 6. *δ-MINCUT is NP-hard.*

Proof. Kempe *et al.* [12] 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. Note that the node-version δ-MINCUT problem is just a special case of its edge-version counterpart (see Chapter 5.1 for the formal proof). As a result, the edge-version δ-MINCUT problem is also NP-hard. □

It is still unknown how hard it is to approximate for δ-MINCUT. However, as will be shown later, it is possible to find algorithms that achieve near-constant approximations, so we would expect δ-MINCUT to be a more tractable problem than δ-MAXFLOW.

4.2.2 Naive Algorithm

In this section, we present a naive approximation algorithm for δ-MINCUT. Before the detailed introduction, it is necessary to introduce the following concept.

Definition 13 (δ-cover). *For a given set of contacts S, its δ-cover (denoted by $\mathsf{Cover}_\delta(S)$) is the smallest set of δ-removals required to disable 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 disable 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 $|\mathsf{Cover}_\delta(S)| = 3$. It is trivial to prove that the value of $\mathsf{Cover}_\delta(S)$ is uniquely determined once the contact set S is given. A simple approach for calculating $\mathsf{Cover}_\delta(S)$ is shown in Appendix C.

The basic idea of the naive algorithm is to first compute MinCut_1, which has been shown to be computationally tractable in Chapter 3.3. Note that the computation of MinCut_1 gives us the smallest **1-cut** CUT_1^*, and if we compute the δ-cover of CUT_1^*, a feasible solution to δ-MINCUT will be derived. The above procedure is shown in Algorithm 2. Note that the solution found in the naive algorithm is only optimal when we input $\delta = 1$ and approximation gaps exist when $\delta \geq 2$. Surprisingly, this naive algorithm has a good approximation ratio, as is shown in Theorem 7.

Algorithm 2 Naive Algorithm for δ-MINCUT

```
1: Compute the smallest 1-cut $\mathsf{CUT}_1^*$;
2: Return the $\delta$-cover of $\mathsf{CUT}_1^*$ as the solution.
```

Theorem 7. *The naive algorithm (Algorithm 2) achieves δ-approximation for δ-MINCUT, i.e., $\frac{\mathsf{ALG}}{\mathsf{OPT}} \leq \delta$.*

Proof. Denote S the set of contacts disabled by the 1-removals found in the first step of the naive algorithm. It is obvious that $|\mathsf{Cover}_\delta(S)| \leq \mathsf{MinCut}_1$ for any $\delta \geq 1$. At the same time, we notice that each δ-removal found in the optimal solution to δ-MINCUT contains at most δ contacts, which corresponds to at most δ 1-removals. Thus, we can disconnect the source-destination pair with at most $\delta\mathsf{MinCut}_\delta$ 1-removals. According to the minimality of MinCut_1, we have $\mathsf{MinCut}_1 \leq \delta\mathsf{MinCut}_\delta$, which implies that

$$|\mathsf{Cover}_\delta(S)| \leq \mathsf{MinCut}_1 \leq \delta\mathsf{MinCut}_\delta,$$

i.e., δ-approximation is achieved by the naive algorithm for δ-MINCUT. □

The overall time complexity of Algorithm 2 depends on the way we compute MinCut_1 (see Chapter 3.3 for different approaches). For example, if Ford-Fulkerson Algorithm is used over the Line Graph to compute MinCut_1, then the first step of Algorithm 2 consumes $O(|C|^3)$ time. The computation of the δ-cover consumes $O(|C|)$ time. Hence, the overall time complexity of the naive algorithm is $O(|C|^3)$.

4.2.3 Heuristic Algorithm

Although the above naive algorithm achieves a good approximation ratio, it has an obvious drawback as is illustrated in Figure 4-3. Suppose $\delta = 3$. If the naive algorithm is used, we first calculate MinCut_1, where contacts $(\mathsf{AB}, 4)$ and $(\mathsf{AC}, 4)$ are removed; thus, we need two δ-removals to disable the above two contacts when $\delta = 3$, meaning that the naive algorithm will output a δ-cut of size two. However, it is obvious that the optimal cut should remove contacts $(\mathsf{SA}, 1)$, $(\mathsf{SA}, 2)$ and $(\mathsf{SA}, 3)$, requiring only one δ-removal when $\delta = 3$. This bad scenario occurs because the naive algorithm uses the result when $\delta = 1$, which only considers one-slot removals and fails to distinguish the temporal correlation between contacts. For example, although removing (SA, 1), (SA, 2) and (SA, 3) requires more one-slot removals, they are temporally closed to each other and thus can be removed together by a single δ-removal when δ is large. By comparison, contacts (AB, 4) and (AC, 4) are spatially disjoint and poorly correlated, which means that they cannot be removed together no matter how large δ is.

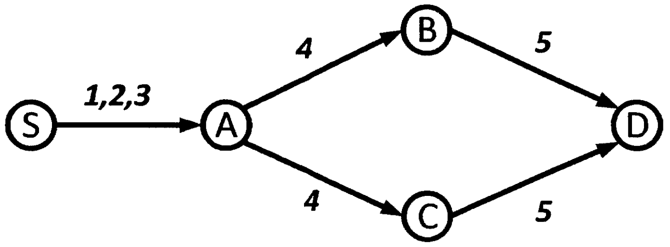


Figure 4-3: Example where the naive algorithm (Algorithm 2) fails to find the optimal solution. Suppose $\delta = 3$ and the source-destination pair is (S, D). The naive algorithm will remove contacts (AB, 4) and (AC, 4), which needs two δ-removals. By comparison, the optimal solution is to remove contacts (SA, 1), (SA, 2) and (SA, 3), which only needs one δ-removal when $\delta = 3$.

Observing the above drawback, we propose a heuristic improvement (Algorithm 3) so as to leverage the time correlation between contacts. The idea is to first assign weights to contacts according to their time correlation and then call the naive algorithm over the weighted time-varying graph. Note that approaches for computing MinCut_1 over an unweighted time-varying graph is still valid in the weighted case.

The key part is the weight assignment. Intuitively, if there are more contacts in the "temporal neighborhood" of a given contact, its time correlation to other contacts is larger and a single δ-removal is likely to disable more contacts together with this given contact. Hence, such a contact should be given a smaller weight such that it has a higher priority of being removed. Keeping this in mind, we propose the following weight assignment scheme. Suppose we are assigning the weight for contact (e, t). We first scan all the δ-slot windows containing this contact and then find a window that contains the maximum number of contacts (say it contains K contacts). The weight of contact (e, t) is just the reciprocal of K. This procedure is illustrated in Figure 4-4.

Algorithm 3 Heuristic Algorithm for δ-MINCUT

```
1: Call SetWeight to compute the weight for each contact;
2: Call the naive algorithm (Algorithm 2) over the weighted time-varying graph;
3: Procedure: SetWeight
4: for each contact $(e,t)$ do
5:     Scan all the $\delta$-slot windows containing $(e,t)$, and find the one that contains the maximum number of contacts (say containing $K_{e,t}$ contacts);
6:     Set $\omega_{e,t} = \frac{1}{K_{e,t}}$;
7: end for
```

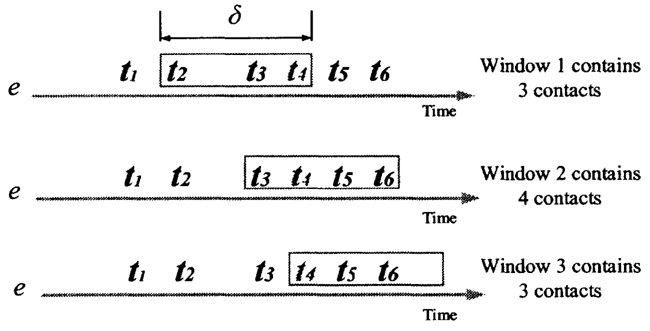

δ
e
t1 t2 t3 t4 t5 t6
Time
Window 1 contains 3 contacts
Window 2 contains 4 contacts
Window 3 contains 3 contacts

Figure 4-4: Illustration of the weight assignment. Suppose edge e is active in slots $t_1 < t_2 < \cdots < t_6$ and we are assigning the weight for contact (e, t_4). We first scan all the windows (with a length of δ slots) that contain the target contact (e, t_4) and then check the number of contacts containing in these windows. It turns out that the second window contains the maximum number of contacts (4 contacts). Hence, we know $K_{e,t_4} = 4$ and its weight is $\frac{1}{K_{e,t_4}} = \frac{1}{4}$.

Applying the heuristic algorithm to the example in Figure 4-3, we would give a weight of $\frac{1}{3}$ to contacts $(\mathsf{SA}, 1)$, $(\mathsf{SA}, 2)$ and $(\mathsf{SA}, 3)$, and a weight of 1 to contacts $(\mathsf{AB}, 4)$ and $(\mathsf{AC}, 4)$. In this case, MinCut_1 will correspond to the removals of contacts $(\mathsf{SA}, 1)$, $(\mathsf{SA}, 2)$ and $(\mathsf{SA}, 3)$ since their weights only sum to one while removing $(\mathsf{AB}, 4)$ and $(\mathsf{AC}, 4)$ yields the total weights of 2. Thus, the optimal solution is found with the heuristics.

Finally, we investigate the performance of the heuristic algorithm. Unfortunately, it turns out the theoretical approximation ratio is still δ, as is shown in Theorem 8. However, the subsequent numerical results indicate that the practical performance of the heuristic algorithm is much better. The time complexity used for setting weights is merely $O(|C|\delta)$, so the heuristic improvement does not increase the overall time complexity.

Theorem 8. *The heuristic algorithm (Algorithm 3) 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 or equals to 1. Indeed, consider a certain δ-removal that deletes contacts $(e, t_1), (e, t_2), \cdots, (e, t_n)$. It should be obvious that $K_{e,t_i} \geq n$ for any $1 \leq i \leq n$, which means $\sum_{i=1}^{n} \omega_{e,t_i} = \sum_{i=1}^{n} \frac{1}{K_{e,t_i}} \leq \sum_{i=1}^{n} \frac{1}{n} = 1$. 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 $(e, t_1), (e, t_2), \cdots, (e, t_{n_e})$, 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

$$W_{e,i} = \sum_{(e,t)\in C_{e,i}} \omega_{e,t}, \ \forall e \in E_c \text{ and } 1 \leq i \leq n_e. \tag{4.2}$$

Then we have

$$|\mathsf{Cover}_\delta(C)| = \sum_{e\in E_c} n_e = \sum_{e\in E_c} \sum_{i=1}^{n_e} \frac{\sum_{(e,t)\in C_{e,i}} \omega_{e,t}}{W_{e,i}}, \tag{4.3}$$

where the last equality is due to equation (4.2). 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 $C_{e,i}$. 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

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

Taking the above inequality into (4.3), we obtain

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

where 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 heuristic algorithm. Suppose C_{ALG} is the set of contacts disabled by the solution of the heuristic algorithm and C^* is the set of contacts disabled by the optimal solution to δ-MINCUT. Then according to Lemma 2, we have

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

Since the heuristic 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 heuristic algorithm. $\square$

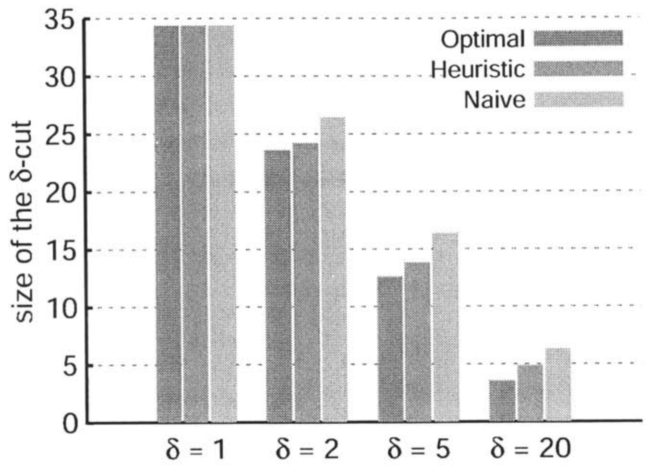


Figure 4-5: Comparison among the naive algorithm (Algorithm 2), the heuristic algorithm (Algorithm 3) and the optimal result to δ-MINCUT.

Table 4.1: Approximation gaps of different algorithms

Approximation Gap	$\delta = 1$	$\delta = 2$	$\delta = 5$	$\delta = 20$
Naive Algorithm	0%	11.92%	29.95%	77.22%
Heuristic Algorithm	0%	2.54%	9.92%	36.62%

Numerical Results on the Heuristic Algorithm: The simulation environment is the same as that used for Algorithm 1. Figure 4-5 shows the comparison between the naive and the heuristic algorithms, and Table 4.1 gives their detailed approximation gaps[1]. As is expected, the heuristic algorithm significantly outperforms the naive algorithm; the only exception is when $\delta = 1$ since both algorithms give the optimal solution in this case. We also notice that the heuristic algorithm is very close to the optimum: the approximation gap is less than 10% in most cases and less than 40% even in the worst case; this is much better than the theoretical bound in Theorem 8. The final observation is that the approximation gaps of both algorithms become larger with the increase in δ; this tendency is consistent with their theoretical approximation ratio of δ.

[1]The approximation gap is calculated by $\frac{\mathsf{ALG}-\mathsf{OPT}}{\mathsf{OPT}}$.

Chapter 5

Discussion

In this chapter, we discuss some important extensions and applications of the proposed framework. First, we discuss how to extend the graph and failure model used in the previous chapter to a more general setting. Then some extended survivability metrics are investigated. Finally, we demonstrate the applications of these metrics.

5.1 Reductions Between Variants

In this section, we discuss how to reduce different variants of the proposed problems to our existing framework.

(1) Node vs. Edge. Sometimes we may need to handle node-related problems. In our context, we are interested in

- Node-version δ-MAXFLOW: find the maximum number of δ-node-disjoint[1] journeys in a time-varying graph.
- Node-version δ-MINCUT: find the minimum number of δ-node-removals[2] that can render the destination unreachable from the source.

The above node-version problems can be easily reduced to edge-version problems by using the *split graph* (which is also a time-varying graph). The basic idea of split graphs is

[1]The definition of δ-node-disjoint journeys is similar to its edge-version counterpart, requiring that any two of these journeys do not use the same *node* within δ slots.

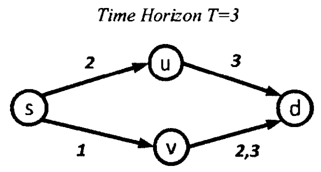
[2]A δ-node-removal means that we disable a *node* for consecutive δ slots.

to split each node in the original time-varying graph into two nodes and replace the original node by an edge between the two splitting nodes. To be more specific, for a time-varying graph $\mathcal{G}$, the construction of its split graph $S(\mathcal{G})$ is as follows.

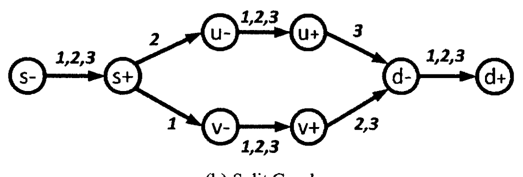
- Split any node v in $\mathcal{G}$ into two nodes v^- and v^+; add an edge $v^- \rightarrow v^+$ (i.e., *splitting edge*) in the split graph. The active slots and traversal time of this edge are dependent on the presence and delay functions of node v (see the remark below).

- For each contact (uv, t) in $\mathcal{G}$, create a contact (u^+v^-, t) in the split graph. Note the one-to-one correspondence: $(uv, t) \leftrightarrow (u^+v^-, t)$.

Remark on split graphs. It should be clear that any node features (e.g., node presence and delay) are transformed to edge features of the corresponding splitting edges. If node v is active in slots $t_1, t_2, \cdots$, then its splitting edge $v^- \rightarrow v^+$ should also be active in these slots. Similarly, if there is a delay of D slots when passing node v, then the traversal delay over its splitting edge $v^- \rightarrow v^+$ should also be D slots. It should be pointed out that if we do not consider node delay or presence functions, each splitting edge is active in the entire time span $1, \cdots, T$ and the traversal delay is zero. An example of split graphs is shown in Figure 5-1.



(a) Original Time-varying Graph



(b) Split Graph

Figure 5-1: Illustration of the split graph. Note that we do not consider node presence or delay functions here so each splitting edge is active in the entire time span and its traversal delay is zero. Also note that the source-destination pair becomes (s^+, d^-) in the split graph.

Split graphs are very useful in solving the node-version δ-MINCUT and δ-MAXFLOW problems, as is shown in the following theorem.

Theorem 9. *Solving the node-version δ-MAXFLOW and δ-MINCUT problems in a time-varying graph $\mathcal{G}$ is equivalent to solving the edge-version δ-MAXFLOW and δ-MINCUT problems in its split graph $S(\mathcal{G})$.*

Proof. For a journey J in the original time-varying graph $\mathcal{G}$:

$$J : s \to (sv_1, t_1) \to (v_1v_2, t_2) \to \cdots \to d,$$

we can find the corresponding *split journey* $\widehat{J}$ in the split graph $S(\mathcal{G})$:

$$\begin{aligned} \widehat{J} : s^+ &\to (s^+v_1^-, t_1) \to (v_1^-v_1^+, t_2) \to \\ &\to (v_1^+v_2^-, t_2) \to (v_2^-v_2^+, t_2) \to \cdots \to d^-. \end{aligned}$$

It should be mentioned that the inverse mapping, i.e., $\widehat{J} \mapsto J$, also holds, so there is an one-to-one correspondence between a journey J in $\mathcal{G}$ and its split journey $\widehat{J}$ in $S(\mathcal{G})$, i.e., $\widehat{J} \leftrightarrow J$. Besides, it's obvious that a certain journey visits node v in slot t if and only if its split journey visits contact (v^-v^+, t).

We first prove for δ-MAXFLOW. Consider an arbitrary set of journeys $J_1, \cdots, J_m$ in $\mathcal{G}$ and their split journeys $\widehat{J}_1, \cdots, \widehat{J}_m$ in $S(\mathcal{G})$. It can be easily verified that $J_1, \cdots, J_m$ are δ-node-disjoint if and only $\widehat{J}_1, \cdots, \widehat{J}_m$ are δ-edge-disjoint. Hence, finding the maximum number of δ-node-disjoint journeys in $\mathcal{G}$ is equivalent to finding the maximum number of δ-edge-disjoint journeys in its split graph $S(\mathcal{G})$.

As for δ-MINCUT, we first notice that any δ-node-cut $\mathsf{CUT}_\delta^{node}$ in $\mathcal{G}$ naturally corresponds to a δ-edge-cut $\mathsf{CUT}_\delta^{edge}$ in its split graph $S(\mathcal{G})$, where

$$\mathsf{CUT}_\delta^{edge} = \left\{(v^-v^+, t) | (v, t) \in \mathsf{CUT}_\delta^{node}\right\}.$$

As a result, the size of the minimum δ-edge-cut in $S(\mathcal{G})$ is less or equal to the size of the minimum δ-node-cut in $\mathcal{G}$. To show the inverse direction, suppose $\mathsf{CUT}_\delta^{edge}$ is an arbitrary δ-edge-cut in the split graph. Then we construct a set of δ-node-removals (denoted by R)

in $\mathcal{G}$ from $\mathsf{CUT}_{\delta}^{edge}$. For any contact $(v^- v^+, t) \in \mathsf{CUT}_{\delta}^{edge}$, we add (v, t) to R. For any contact $(u^+ v^-, t) \in \mathsf{CUT}_{\delta}^{edge}$, if $u \neq s$, we add (u, t) to R; if $u = s$, we add (v, t) to R. It can be easily observed that R forms a legitimate δ-node-cut in $\mathcal{G}$. Therefore, the size of the minimum δ-node-cut in $\mathcal{G}$ is less or equal to the size of the minimum δ-edge-cut in $S(\mathcal{G})$, which completes our proof. □

Note that the source-destination pair in the split graph $S(\mathcal{G})$ becomes (s^+, d^-). It essentially implies that the node-version δ-MAXFLOW and δ-MINCUT problems are just a special case of the corresponding edge-version problems which have been extensively studied in the previous chapters.

(2) Directed vs. Undirected. Sometimes the underlying graph is undirected and we need to transform it into the directed case in order to apply our framework. Now we consider two possible scenarios. In the first case, the two directions of an undirected edge are failure-independent; that is, if one direction fails, the other direction is not affected. For example, in cognitive radio networks, the downlink and the uplink may be assigned two independent channels, where the reclamation of one channel does not influence the other. In this case, it is enough to replace the undirected edge with two directed edges (Figure 5-2(b)). On the other hand, it is often the case that the two directions are failure-dependent, which means both directions are ON or OFF simultaneously. For instance, if an unexpected obstacle appears in the middle of two nodes, then transmissions in both directions are disabled. In this case, we cannot simply use two directed edges, and the gadget shown in Figure 5-2(c) will be helpful. It can be observed that no matter which direction is used when we cross the undirected edge $u - v$, the same directed edge $w_1 \rightarrow w_2$ is traversed. Conversely, disabling edge $w_1 \rightarrow w_2$ is enough to disable the two directions. Hence, with such a gadget, we can easily solve δ-MAXFLOW and δ-MINCUT in the undirected and failure-dependent case.

5.2 Arbitrary Failures

Our previous discussion about (n, δ)-survivability assumes that failures occur at the beginning of slots and last for multiples of the slot length. In this section, we show how to

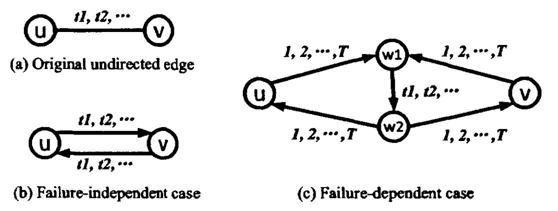


Figure 5-2: Illustration of the gadgets that reduces an undirected edge $u - v$ to directed edges. In the failure-independent case, we only need to replace it by two directed edges. In the failure-dependent case, we first add two additional nodes w_1, w_2 and then add directed edges $u \to w_1$, $w_2 \to v$, $v \to w_1$, $w_2 \to u$ and $w_1 \to w_2$. The active slots and traversal delay of $w_1 \to w_2$ are the same as those of the original undirected edge $u - v$. The remaining directed edges are active in the entire time span and have zero traversal delay.

relax this assumption and accommodate failure happening at arbitrary instants or lasting for arbitrary duration.

Denote l the slot length used for the time-varying graph. Suppose a failure with arbitrary duration $D \in \mathbb{R}^+$ occurs at instant $t + \epsilon$, where t is the beginning of a certain slot (say slot n) and $0 < \epsilon < l$. It is easy to see that this failure influences the transmission in slots $n, n+1, \cdots, n + \lceil \frac{D+\epsilon}{l} \rceil - 1$, which is the same as a failure happening at instant t and lasting for $\lceil \frac{D+\epsilon}{l} \rceil$ slots with respect to their destruction to the graph.

5.3 Extended Metrics

In this section, we discuss two other survivability metrics that are extended from MinCut_δ and $\mathsf{MaxFlow}_\delta$.

Definition 14 ($\mathsf{MinTime_n}$). *Given any positive integer n,* $\mathsf{MinTime_n}$ *is the minimum value of δ such that we can render the destination unreachable from the source with at most n δ-removals.*

Intuitively, $\mathsf{MinTime_n}$ indicates the minimum failure duration needed to disconnect the network given *a fixed number of failures*. If $\mathsf{MinTime_n} = \delta$, the source-destination pair can survive any n failures that last for fewer than δ slots and is thus $(n, \delta - 1)$-survivable. Note that $1 \leq \mathsf{MinTime_n} \leq T$. If we cannot disconnect the network with n permanent failures, this metric is null. It should also be pointed out that $\mathsf{MinTime_n}$ is the sibling metric of MinCut_δ since the latter evaluates the minimum number of failures needed to disconnect the network given *fixed duration of failures*.

Applying the similar trick to $\mathsf{MaxFlow}_\delta$, we can define the following metric.

Definition 15 ($\mathsf{MaxTime_n}$). *Given any positive integer n,* $\mathsf{MaxTime_n}$ *is the maximum value of δ such that there are at least n δ-disjoint journeys from the source to the destination.*

When $\mathsf{MaxTime_n} = \delta$, there are at least n δ-disjoint journeys, which means the source-destination pair is $(n-1, \delta)$-survivable. If there exist no n δ-disjoint journeys even when $\delta = 1$, then $\mathsf{MaxTime_n}$ is null.

From the definitions of $\mathsf{MinTime_n}$ and $\mathsf{MaxTime_n}$, it is not hard to make the following observation regarding their computations.

Observation 2. *The computations of* $\mathsf{MinTime_n}$ *can be reduced to the computation of* MinCut_δ *in polynomial time, and vice versa. The same argument applies to* $\mathsf{MaxTime_n}$ *and* $\mathsf{MaxFlow}_\delta$.

Proof. We only show the reductions from $\mathsf{MinTime_n}$ to MinCut_δ while the analysis for other reductions is similar.

Suppose we have an oracle that can output the value of MinCut_δ if we input δ. To compute $\mathsf{MinTime_n}$, we can first feed $\delta = 1$ to the oracle and get $\mathsf{MinCut_1}$. If $\mathsf{MinCUt_1} \leq n$, then $\mathsf{MinTime_n} = 1$; otherwise we increase δ by 1 and repeat the above procedure until $\mathsf{MinCut}_{\delta^*} \leq n$ for some δ^*, where we obtain $\mathsf{MinTime_n} = \delta^*$. Since the above procedure repeats for at most T rounds, the computation of $\mathsf{MinTime_n}$ can be reduced to finding MinCut_δ in polynomial time. □

This observation has two important implications. First, computing $\mathsf{MinTime_n}$ or $\mathsf{MaxTime_n}$ is as hard as computing MinCut_δ or $\mathsf{MaxFlow}_\delta$, which is NP-hard. Second, the approximation algorithms we develop for computing MinCut_δ and $\mathsf{MaxFlow}_\delta$ can be used to solve $\mathsf{MinTime_n}$ or $\mathsf{MaxTime_n}$. Therefore, we can see the fundamental role of MinCut_δ and $\mathsf{MaxFlow}_\delta$ in understanding other metrics.

5.4 Applications of MinCut_δ

Besides being a survivability metric, MinCut_δ can also be applied to many practical problems. For example, we may need to block the spread of rumors or viruses in time-varying networks (e.g., evolving social networks or multi-robot systems) [9]. By calculating MinCut_δ, we can easily identify crucial contacts (or spread blockers) and block rumors or viruses by disabling only a small number of edges for δ slots. An extension of this application is the so-called Flow Inhibition/Interdiction Problem [21] in time-varying networks, where we need to disconnect the network at the minimum cost.

Chapter 6

Application: Bus Communication Networks

In this chapter, 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 [1]. Each bus in the network has a pre-designed 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 where topology changes are caused by the estimated bus mobility. However, the prediction may not be perfect due to various reasons such as unexpected obstacles, traffic accidents, traffic jam, etc. The goal of survivable routing is to reduce the packet loss rate due to these unpredictable failures.

In the rest of this chapter, we first present the design of the survivable routing protocol using δ-disjoint journeys. Then we discuss trace statistics, simulation settings and results.

6.1 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). The advantages of DJR over other reliable transmission protocols are outlined as follows.

• *Simplicity of Deployment in Time-varying Networks:* Static networks usually deploy ARQ at the data link layer and TCP at the transport layer for error recovery. However, due to the lack of connectivity, it is not only difficult to get timely ACK at the sender but also hard to find opportunities for retransmissions. In contrast, DJR does not require any feedback, which greatly simplifies the data link layer and the transport layer (no need for error recovery). In addition, as a network-layer protocol, it can be combined with FEC codes at the physical layer (e.g., erasure code [10]) to achieve a better performance.

• *Temporal Diversity*: Many traditional survivable routing protocols rely on spatial diversity, such as Disjoint-Path Routing (DPR) [24] [14], where spatially-disjoint paths are used to recover packets. However, spatial diversity is a demanding requirement in networks with sparse and intermittent connectivity. We will demonstrate that it is hard to find even two spatially-disjoint paths in the bus network. By comparison, DJR exploits temporal diversity to combat failures and is well suited for time-varying networks, especially when failures are transient.

• *Two-fold Tunability*: Our survivability framework has two natural parameters, namely n and δ. Hence, the tunability of DJR is also in two dimensions: we can both tune the number of δ-disjoint journeys to use, and also adapt the degree of temporal disjointness. By comparison, existing survivable routing protocols (e.g., [11, 19, 25]) lack such flexibility.

6.2 Trace Statistics

We use the trace from UMassDieselNet [1] where a public bus transportation system was operated around Amherst, Massachusetts. The trace records the contacts among 40 buses in 9 weekdays, which roughly reflects the bus mobility over those pre-designed bus routes. We will use this estimated mobility to coarsely predict the topology changes of this bus communication network (yet, we will assume that the estimation is imperfect due to obstacles, traffic jam, etc.).

To facilitate our subsequent discussion, we pre-process the raw trace and observe two important features of this bus communication network. The first observation is the "bursty" structure of contacts between any two buses; that is, buses only communicate with each other occasionally. Figure 6-1(a) 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 Figure 6-1(b), most contacts span less than 20s.

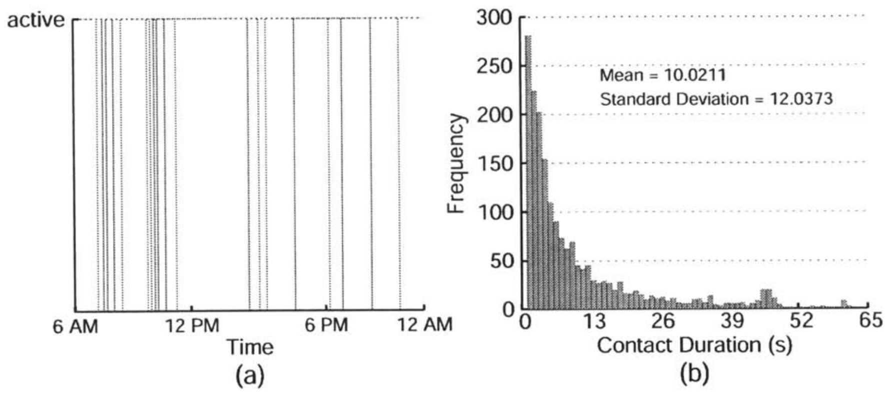


Figure 6-1: Statistical structures of the bus communication network. (a) The bursty pattern for the contacts between a typical pair of buses. (b) Histogram for the contact duration. Most contacts only last for a short period of time (less than 20s).

6.3 Simulation Settings

In our simulation, the slot length is identical to the trace resolution, i.e., one second. According to the measurement in [1], the average transmission rate is about 1.64Mbps. If the packet size is set to be 1KB, 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 packets is generated between a random source-destination pair immediately after the previous packet becomes expired or delivered. In addition, at most n copies are allowed, meaning that we can use at most n δ-disjoint journeys to send these copies. In addition, we resort to Algorithm 1 to compute δ-disjoint journeys.

Since it is impossible to precisely predict the topology changes in the future, we impose random failures on the estimated 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.

6.4 Total Number of δ-Disjoint Journeys

We first look at the maximum number of δ-disjoint journeys in the bus communication network (Figure 6-2). 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. 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 (we will validate this later). 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 short and bursty contacts in this bus network (see Chapter 6.2). For example, if a contact only lasts for 20s while the next contact occurs 1 hour later, the temporal distance between any two journeys that use this link is at most 20s within the deadline (unless the packet deadline is greater than 1 hour), and any larger δ actually prevents any two δ-disjoint journeys from using the same link within the deadline, which essentially becomes spatial disjointness. The final observation is that extending the packet deadline increases the total number of δ-disjoint journeys since there are more transmission opportunities within a longer deadline.

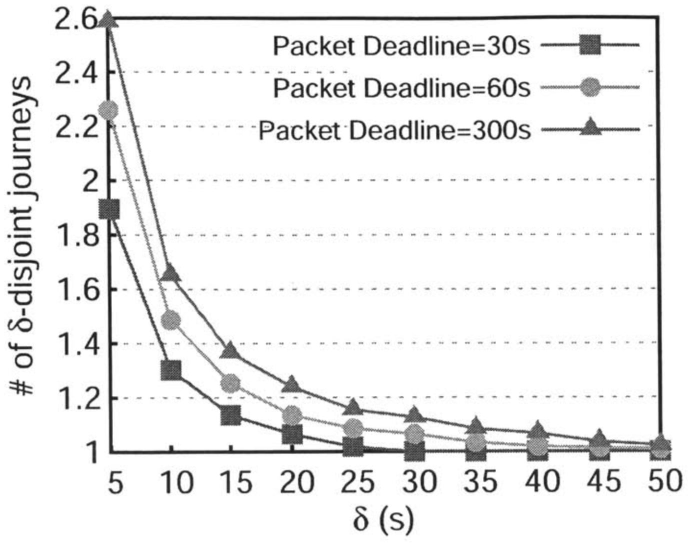


Figure 6-2: The total number of δ-disjoint journeys in the bus communication network.

6.5 Tunability of DJR

Next, we study the two-fold tunability of DJR (Figure 6-3). 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 produced).

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 relatively small, there exist many δ-disjoint journeys and we can choose any n of them to use. 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 goes down with the increase of δ initially. However, the increase of δ also leads to the reduction in the number of δ-disjoint journeys (see Figure 6-2); 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 $\mathsf{MaxFlow}_\delta \geq n$, i.e., $\mathsf{MaxTime}_n$ we introduce in Chapter 5.3.

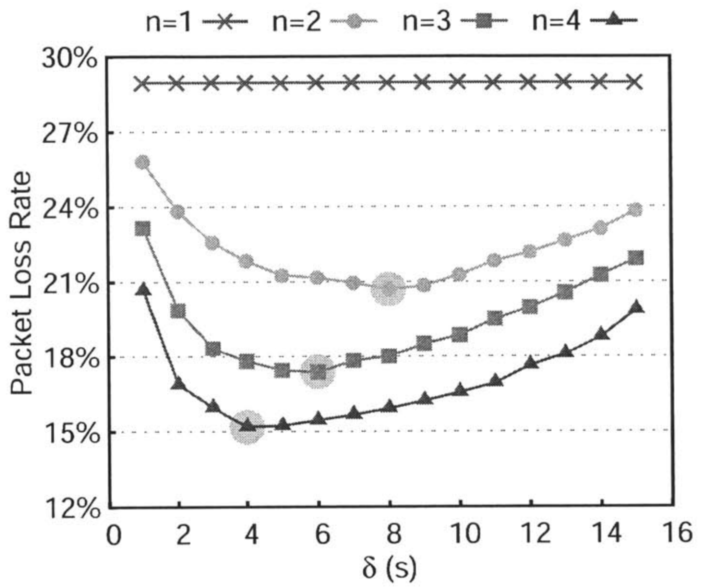


Figure 6-3: Influence of n and δ on packet loss rates (DDL=300s, p=0.05, d=60s).

6.6 Comparison with DPR

To numerically demonstrate the necessity of exploiting temporal diversity in time-varying networks, we numerically study the comparison between Disjoint-Journey Routing (DJR) and Disjoint-Path Routing (DPR) [24] [14]. Note that DPR relies on spatial diversity and only sends copies along spatially-disjoint paths; in a sense, DPR is a special case of DJR when $\delta = T$. In the comparison, we let DJR use the "optimal" value of δ as previously seen in Figure 6-3. We also consider different failure duration, ranging from tens of seconds (transient failures) to infinity (permanent failures).

The comparison between DJR and DPR is shown in Figure 6-4. In almost all the cases, the network experiences significantly higher packet loss rates (around 2x more) if DPR is used rather than DJR. The underlying reason has been argued: it is almost impossible to find even two paths that are spatially-disjoint due to the sparse and intermittent connectivity. Hence, traditional DPR is not very effective in time-varying networks. By comparison, DJR exploits temporal diversity, where we can find a reasonable number of journeys to use despite the intermittent connectivity. As a result, DJR is more suitable for time-varying networks that are intermittently connected, especially when most failures are transient.

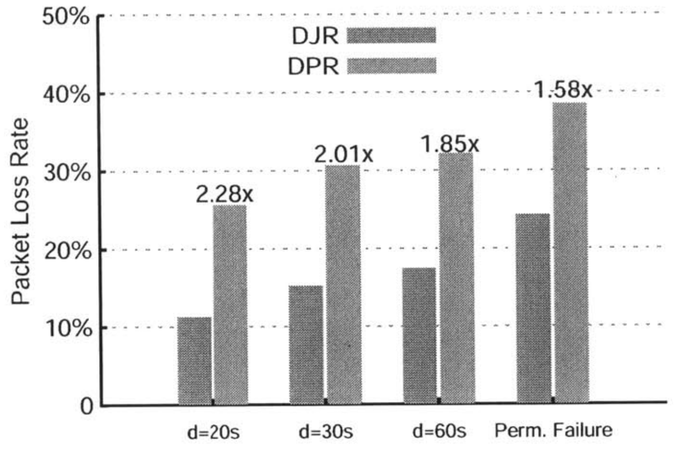


Figure 6-4: Comparison between DJR and DPR (DDL = 300s, p=0.05, n=3).

Chapter 7

Conclusion

In this thesis, a new survivability framework is proposed for time-varying networks, namely (n, δ)-survivability. In order to evaluate (n, δ)-survivability, two metrics are proposed: MinCut_δ and $\mathsf{MaxFlow}_\delta$. These metrics are analogous to MinCut and MaxFlow in static graphs but account for the temporal features of time-varying networks, such as failure duration. We analyze the fundamental relationship between the two metrics and show that Menger's Theorem only conditionally holds in time-varying graphs. Due to the difference, it is NP-hard to compute both survivability metrics. To resolve the computational intractability, we develop several approximation algorithms. Finally, we use trace-driven simulations to demonstrate the application of our framework in the robust design of a real-world bus communication network.

Appendix A

Compact Formulation

In this appendix, we provide a compact formulation for $\mathsf{MaxFlow}_\delta$ and MinCut_δ. For the simplicity of demonstration, we only provide the compact formulation for the case with one-slot traversal delay; other cases can be formulated in a similar way. Suppose the original time-varying graph is $\mathcal{G}$ (its underlying graph is $G = (V, E)$ and the time span is $\mathcal{T} = \{1, \cdots, T\}$). The source-destination pair is (s, d) and the corresponding Line Graph is $L(\mathcal{G})$. Except node s and node d, every node in $L(\mathcal{G})$ has unit capacity; edges in $L(\mathcal{G})$ have infinite capacity.

The compact formulation of $\mathsf{MaxFlow}_\delta$ is defined over the Line Graph $L(\mathcal{G})$. The basic idea is to formulate it as a classic maxflow problem in the Line Graph plus additional constraints on the integrality and the δ-disjointness. Since node capacity is considered in the Line Graph, we resort to the split graph of $L(\mathcal{G})$ (see Chapter 5.1), which is denoted by $S(L(\mathcal{G})) = (V', E')$. Note that the capacity of every splitting edge has unit capacity and any other edge has infinite capacity. Also note that the source-destination pair becomes (s^+, d^-) in the split graph. In Figure A-1, we give an example for the above transformations: $\mathcal{G} \rightarrow L(\mathcal{G}) \rightarrow S(L(\mathcal{G}))$.

Denote F the exogenous flow into the split graph and $f(x, y)$ the flow over edge $(x, y) \in E'$ in the split graph. Also let $I(x)$ and $O(x)$ be the set of ingoing and outgoing neighbors of node x in the split graph, respectively, i.e., $I(x) = \{y|(y, x) \in E'\}$ and $O(x) = \{y|(x, y) \in E'\}$. The compact formulation for $\mathsf{MaxFlow}_\delta$ is as follows.

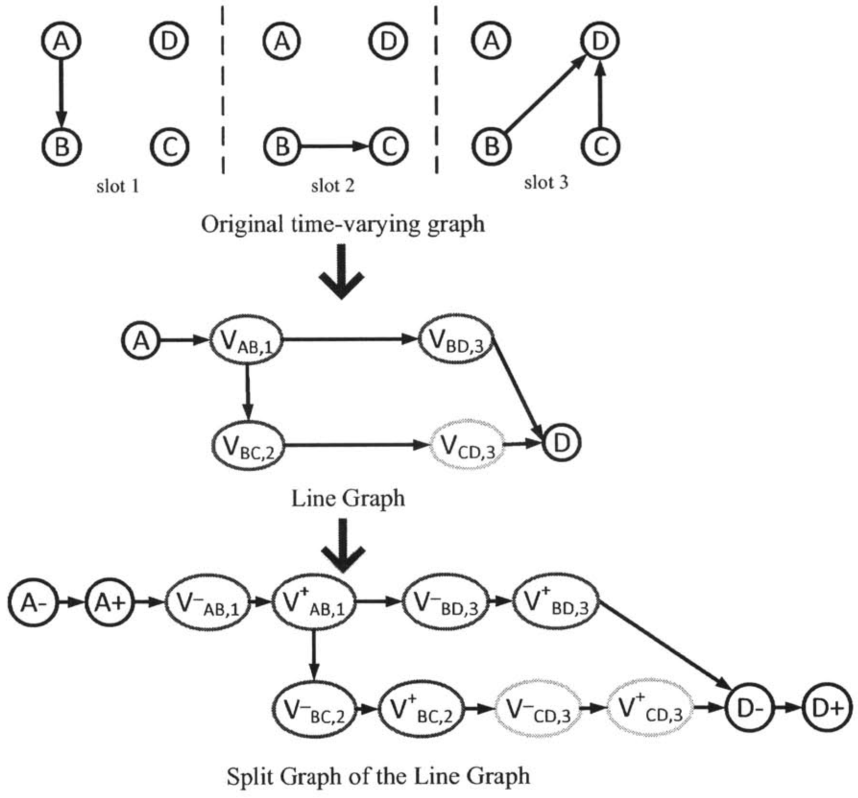


Figure A-1: Illustration of the transformations: $\mathcal{G} \to L(\mathcal{G}) \to S(L(\mathcal{G}))$. The source-destination pair is (A, D) in $\mathcal{G}$ and $L(\mathcal{G})$, and is $(\mathsf{A}^+, \mathsf{D}^-)$ in $S(L(\mathcal{G}))$.

$$\begin{aligned}
\max \quad & F \\
\text{s.t.} \quad & \sum_{y \in O(x)} f(y,x) - \sum_{y \in I(x)} f(x,y) = \begin{cases} F, & x = s^+ \\ -F, & x = d^- \\ 0, & x \in V' \backslash \{s^+, d^-\} \end{cases} \\
& \sum_{t'=t}^{t+\delta-1} f(v_{e,t'}^-, v_{e,t'}^+) \le 1, \; \forall \text{ splitting edge } (v_{e,t}^-, v_{e,t}^+) \in E' \\
& f(x,y) \in \{0, 1, \cdots\}, \; \forall\, (x,y) \in E'.
\end{aligned}$$

The first constraint is just the flow conservation constraint. The second constraint forces any splitting edge, which corresponds to a contact in the original time-varying graph, to be used by at most one journey within a window of δ slots. Note that the unit-capacity constraint for splitting edges has already been included in the second constraint.

It is interesting to observe that when $\delta = 1$ the second constraint becomes $f(v_{e,t}^-, v_{e,t}^+) \leq 1$, which we recognize as the capacity constraint for splitting edges. In this case, the above formulation just becomes a standard maxflow problem and the integrality gap disappears. This is consistent with Theorem 1.

The above formulation only gives the value of $\mathsf{MaxFlow}_\delta$. If we need to know the detailed δ-disjoint journeys, we can label different flows by their first-hop destination and consider the conservation constraints separately.

As for MinCut_δ, we can similarly formulate it as a standard mincut problem in the split Line Graph plus additional constraints on δ-removals. It turns out that the LP relaxation of the compact formulation for MinCut_δ can be formulated as the dual problem of the LP relaxation for the above $\mathsf{MaxFlow}_\delta$ formulation. Thus, obtaining the compact formulation of MinCut_δ reduces to mechanical procedures of LP dualization (omitted here).

Appendix B

Formal Proof to Theorem 2

Recall that our goal is to 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 k-th graph. The constructions for $k = 1, 2, 3$ have been shown in Figure 3-2. In general, graph $\mathcal{G}_k$ has k levels. Different levels share the same source s and have their own "virtual destinations" $d_1, \cdots, d_k$. Note that the real source-destination pair in $\mathcal{G}_k$ is (s, d_k). Besides, the i-th level has $2i - 1$ inner nodes, where we denote $v_{i,j}$ the j-th inner node at level i. The inner contacts in level i include the following.

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

Besides, there are also cross-level edges from lower levels to higher levels, i.e., $d_i \dashrightarrow v_{i+1,j}$ which is active in slot $(i + 1)^2 - 1$ for any $1 \leq i \leq k - 1$ and odd j.

Then we prove by induction that $\mathsf{MaxFlow}_\delta = 1$ and $\mathsf{MinCut}_\delta = k$ for any $\delta \geq 2$ in $\mathcal{G}_k$. When $k = 1$, it is obvious that $\mathsf{MinCut}_\delta = 1$ and $\mathsf{MaxFlow}_\delta = 1$. Now suppose the claim holds in $\mathcal{G}_1, \cdots, \mathcal{G}_k$ 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., $s \dashrightarrow d_k \dashrightarrow d_{k+1}$) and the other is directly descend to the $(k + 1)$-th level (i.e., $s \dashrightarrow v_{k+1,1} \dashrightarrow v_{k+1,2} \dashrightarrow \cdots \dashrightarrow d_{k+1}$). 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 any journey in the latter choice (i.e., $s \dashrightarrow v_{k+1,1} \dashrightarrow v_{k+1,2} \dashrightarrow \cdots \dashrightarrow d_{k+1}$) for any $\delta \geq 2$. As a result, there is only one δ-disjoint journey from s to d_k, i.e., $\mathsf{MaxFlow}_\delta = 1$ 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} if $\delta \geq 2$. To show this point, we first denote C_1 the set of inner contacts in levels $1, 2, \cdots, k$ and all cross-level contacts, and denote 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 obvious that we can still travel within level $k+1$, i.e., $s \dashrightarrow v_{k+1,1} \dashrightarrow v_{k+1,2} \dashrightarrow \cdots \dashrightarrow d_{k+1}$.

• 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}; thus, 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 $s \dashrightarrow d_k \dashrightarrow d_{k+1}$.

• 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 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 $s \dashrightarrow d_k \dashrightarrow d_{k+1}$.

Therefore, we can conclude that $\mathsf{MinCut}_\delta = k+1$, completing the induction proof.

Appendix C

Computation of δ-cover

We describe an algorithm for calculating the δ-cover for a given set of contacts S, i.e., $\mathsf{Cover}_\delta(S)$. Suppose S includes n edges $e_1, \cdots, e_n$ and denote $\mathsf{Cover}_\delta(s_i)$ the δ-cover for all the contacts in S that is associated with edge e_i. Then it is easy to see that $\mathsf{Cover}_\delta(S) = \bigcup_{i=1}^n \mathsf{Cover}_\delta(s_i)$, so it is enough to know the method of calculating the δ-cover for each edge. Now, suppose edge e is active in slots $t_1^{(i)} < t_2^{(i)}, < \cdots < t_{k_i}^{(i)}$ in the contact set S. Then the algorithm for computing $\mathsf{Cover}_\delta(S)$ is shown as Algorithm 4. Note that $\mathsf{Flag}(e,t) = 0$ indicates that contact (e,t) has not been disabled by existing δ-removals. Intuitively, this algorithm just keeps on imposing a δ-removal to the earliest contact that has not been disabled by any exiting δ-removals yet (i.e., $\mathsf{Flag}(e,t) = 0$).

Algorithm 4 Compute the δ-cover for a given set of contacts S

```
 1: Initialize $\mathsf{Cover}_\delta(s_i) = \varnothing$ for any $1 \le i \le n$;
 2: Initialize $\mathsf{Flag}(e,t) = 0$ for any $(e,t) \in S$;
 3: for $i = 1 : n$ do
 4:   for $j = 1 : k_i$ do
 5:     $t = t_j^{(i)}$
 6:     if $\mathsf{Flag}(e_i,t) = 0$ then
 7:       Add contact $(e_i,t)$ to $\mathsf{Cover}_\delta(s_i)$;
          // Note that $(e_i,t)$ becomes a removal head
 8:       $\mathsf{Flag}(e_i,t') = 1$ for any $t \le t' \le t+\delta-1$;
 9:     end if
10:   end for
11: end for
12: $\mathsf{Cover}_\delta(S) = \bigcup_{i=1}^n \mathsf{Cover}_\delta(s_i)$.
```

Bibliography

[1] Aruna Balasubramanian, Brian Neil Levine, and Arun Venkataramani. Enhancing interactive web applications in hybrid networks. In *Proceedings of the 14th ACM International Conference on Mobile Computing and Networking*, MobiCom '08, pages 70–80, New York, NY, USA, 2008. ACM.

[2] Lowell W. Beineke. Characterizations of derived graphs. *Journal of Combinatorial Theory*, 9(2):129 – 135, 1970.

[3] Kenneth A. Berman. Vulnerability of scheduled networks and a generalization of menger's theorem. *Networks*, 28(3):125–134, 1996.

[4] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. In *Proceedings of the 10th International Conference on Ad-hoc, Mobile, and Wireless Networks*, ADHOC-NOW'11, pages 346–359, Berlin, Heidelberg, 2011. Springer-Verlag.

[5] Augustin Chaintreau, Abderrahmen Mtibaa, Laurent Massoulie, and Christophe Diot. The diameter of opportunistic mobile networks. In *Proceedings of the 2007 ACM CoNEXT Conference*, CoNEXT '07, pages 12:1–12:12, New York, NY, USA, 2007. ACM.

[6] A. Ferreira. Building a reference combinatorial model for manets. *Network, IEEE*, 18(5):24–29, Sept 2004.

[7] Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, 1980.

[8] Venkatesan Guruswami, Sanjeev Khanna, Rajmohan Rajaraman, Bruce Shepherd, and Mihalis Yannakakis. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. In *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, STOC '99, pages 19–28, New York, NY, USA, 1999. ACM.

[9] Habiba Habiba, Yintao Yu, Tanya Y. Berger-Wolf, and Jared Saia. Finding spread blockers in dynamic networks. In *Proceedings of the Second International Conference on Advances in Social Network Mining and Analysis*, SNAKDD'08, pages 55–76, Berlin, Heidelberg, 2010. Springer-Verlag.

[10] Sushant Jain, Michael Demmer, Rabin Patra, and Kevin Fall. Using redundancy to cope with failures in a delay tolerant network. In *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '05, pages 109–120, New York, NY, USA, 2005. ACM.

[11] Sushant Jain, Kevin Fall, and Rabin Patra. Routing in a delay tolerant network. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '04, pages 145–158, New York, NY, USA, 2004. ACM.

[12] David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. In *Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 504–513, New York, NY, USA, 2000. ACM.

[13] Hyoungshick Kim and Ross Anderson. Temporal node centrality in complex networks. *Phys. Rev. E*, 85:026107, Feb 2012.

[14] G. Kuperman and E. Modiano. Disjoint path protection in multi-hop wireless networks with interference constraints. In *Global Communications Conference (GLOBECOM), 2014 IEEE*, pages 4472–4477, Dec 2014.

[15] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, pages 177–187, New York, NY, USA, 2005. ACM.

[16] Q. Liang, S. Han, F. Yang, G. Sun, and X. Wang. A distributed-centralized scheme for short- and long-term spectrum sharing with a random leader in cognitive radio networks. *Selected Areas in Communications, IEEE Journal on*, 30(11):2274–2284, December 2012.

[17] Q. Liang, X. Wang, and Z. Feng. Singleton spectrum mobility games with incomplete information. In *Global Communications Conference (GLOBECOM), 2012 IEEE*, pages 5608–5613, Dec 2012.

[18] Q. Liang, X. Wang, X. Tian, F. Wu, and Q. Zhang. Two-dimensional route switching in cognitive radio networks: A game-theoretical framework. *Networking, IEEE/ACM Transactions on*, PP(99):1–1, 2014.

[19] Anders Lindgren, Avri Doria, and Olov Schelén. Probabilistic routing in intermittently connected networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 7(3):19–20, July 2003.

[20] Raj Kumar Pan and Jari Saramäki. Path lengths, correlations, and centrality in temporal networks. *Phys. Rev. E*, 84:016105, Jul 2011.

[21] Cynthia A. Phillips. The network inhibition problem. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 776–785, New York, NY, USA, 1993. ACM.

[22] S. Rangan, T. S. Rappaport, and E. Erkip. Millimeter-wave cellular wireless networks: Potentials and challenges. *Proceedings of the IEEE*, 102(3):366–385, March 2014.

[23] Salvatore Scellato, Ilias Leontiadis, Cecilia Mascolo, Prithwish Basu, and Murtaza Zafer. Evaluating temporal robustness of mobile networks. *IEEE Transactions on Mobile Computing*, 12(1):105–117, January 2013.

[24] Anand Srinivas and Eytan Modiano. Minimum energy disjoint path routing in wireless ad-hoc networks. In *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking*, MobiCom '03, pages 122–133, New York, NY, USA, 2003. ACM.

[25] A. Vahdat and D. Becker. Epidemic routing for partially connected ad hoc networks. *Technical Report, Department of Computer Science, Duke University*, 2000.

[26] John Whitbeck, Marcelo Dias de Amorim, Vania Conan, and Jean-Loup Guillaume. Temporal reachability graphs. In *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, Mobicom '12, pages 377–388, New York, NY, USA, 2012. ACM.

[27] B. Xuan, A. Ferreira, and A. Jarry. Computing the shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14:267–285, 2003.