Chapter 1

Introduction

1.1 Mobile Crowdsensing

Mobile crowdsensing (MCS) is a "sensing paradigm that empowers ordinary citizens to contribute data sensed or generated from their mobile devices" that is aggregated "in the cloud for crowd intelligence extraction and human-centric service delivery" [?]. Over the past decade, substantial research has been conducted on defining and classifying MCS applications [? ? , and references therein]. While not discussed in previous work [4, 5], ShareTrace is a MCS application. The following characterization of ShareTrace assumes the four-layered architecture of a MCS application [?], which offers a comprehensive set of classification criteria that To offer a clear comparison, **??** follows the same structure as [?]. Some aspects of the architecture, namely sampling frequency and sensor activity, are marked according to how ShareTrace is described in previous work, rather than how it would function to optimize for energy efficiency. More detail is provided below in this regard. When classifying ShareTrace as an MCS application, the following description is helpful:

> *ShareTrace is a decentralized, delay-tolerant contact-tracing application that estimates infection risk from proximal user interactions and user symptoms.*

1.1.1 Application Layer

Application Tasks

Task Scheduling *Proactive scheduling* allows users to decide when and where they contribute sensing data, while *reactive scheduling* requires that "a user receives a request, and upon acceptance, accomplishes a task" [**?**]. ShareTrace follows proactive scheduling where the sensing task is to detect proximal interactions with other users. Naturally, the scheduling of this task is at the discretion of the ShareTrace user.

Task Assignment *Centralized assignment* assumes that "a central authority distributes tasks among users." Conversely, with *decentralized assignment*, "each participant becomes an authority and can either perform the task or forward it to other users. This approach is very useful when users are very interested in specific events or activities" [**?**]. The latter naturally aligns with ShareTrace in which each user is responsible for their own interactions that are both temporally and spatially specific.

Task Execution With *single-task execution*, MCS applications assign one type of task to users, while *multi-task execution* assigns multiple types of tasks [**?**]. ShareTrace only involves the single task of sensing proximal interactions. Alternatively, ShareTrace could be defined more abstractly as sensing infection risk through interactions and user symtpoms. In this case, ShareTrace would follow a multi-tasked execution model, where the task of sensing user symptoms is achieved through user reporting or bodily sensors.

Application Users

User Recruitment *Volunteer-based recruitment* is when citizens can "join a sensing campaign for personal interests. . . or willingness to help the community," while *incentive-based recruitment* promotes participation and offers control over the recruitment rate. . . These strategies are not mutually exclusive and users can first volunteer and then be rewarded for quality execution of sensing tasks" [**?**]. ShareTrace assumes volunteer-based recruitment. However, as a decentralized application (dApp) that aligns with the principles of self-soverignty, an incentive structure that rewards users with verifiable, high-quality data is plausible.

User Selection *User-centric selection* is when "contributions depend only on participants['] willingness to sense and report data to the central collector, which is not responsible for choosing them." *Platform-centric selection* is "when the central authority directly decides data contributors. . . Platform centric decisions are taken according to the utility of data to accomplish the targets of the campaign" [**?**]. ShareTrace employs user-centric selection, because the purpose of the application is to passively sense the user's interactions and provide them with the knowledge of their infection risk.

User Type A *contributor* "reports data to the MCS platform with no interest in the results of the sensing campaign" and "are typically driven by incentives or by the desire to help the scientific or civil communities." A *consumer* joins "a service to obtain information about certain application scenario[s] and have a direct interest in the results of the sensing campaign" [**?**]. For ShareTrace, users can be either a consumer or a contributor but are likely biased toward the former.

1.1.2 Data Layer

Data Management

Data Storage *Centralized storage* involves data being "stored and maintained in a single location, which is usually a database made available in the cloud. This approach is typically employed when significant processing or data aggregation is required." *Distributed storage* "is typically employed for delay-tolerant applications, i.e., when users are allowed to deliver data with a delayed reporting" [**?**]. For sensing human interaction, ShareTrace relies on distributed storage in the form of Dataswift Personal Data Accounts. Moreover, ShareTrace is delay-tolerant, so distributed storage is most appropriate. However, for reporting the population-level risk distribution, it is likely that centralized storage would be used.

Data Format *Structured data* is standardized and readily analyzable. *Unstructured data*, however, requires significant processing before it can be used [**?**]. ShareTrace deals with structured data (e.g., user symptoms, actor URIs).

Data Dimensionality *Single-dimension data* typically occurs when a single sensor is used, while *multi-dimensional data* arises with the use of multiple sensors. ShareTrace data is one-dimensional because it only uses Bluetooth to sense nearby users.

Data Processing

Data Pre-processing *Raw data output* implies that no modification is made to the sensed data. *Filtering and denoising* entail "removing irrelevant and redundant data. In addition, they help to aggregate and make sense of data while reducing at the same time the volume to be stored" [**?**]. ShareTrace only retains the actor URIs that correspond to valid contacts (i.e., lasting at least 15 minutes).

Data Analytics *Machine learning* (ML) *and data mining analytics* are not real-time. They "aim to infer information, identify patterns, or predict future trends." On the contrary, *real-time analytics* consist of "examining collected data as soon as it is produced by the contributors" [**?**]. ShareTrace aligns with the former category since it aims to infer the infection risk of users.

Data Post-processing *Statistical post-processing* "aims at inferring proportions given quantitative examples of the input data. *Prediction post-processing* aims to determine “future outcomes from a set of possibilities when given new input in the system” [**?**]. ShareTrace applies predictive post-processing via risk propagation.

1.1.3 Communication Layer

Communication Technology

Infrastructured Technology *Cellular* connectivity “is typically required from sensing campaign[s] that perform real-time monitoring and require data reception as soon as it is sensed.” *WLAN* “is used mainly when sensing organizers do not specify any preferred reporting technologies or when the application domain permits to send data” at “a certain amount of time after the sensing process” [**?**]. Infrastructured technology is also referred to as the *infrastructured transmission paradigm* [**?**]. ShareTrace does not require cellular infrastructure, because it is delay-tolerant and thus only requires WLAN.

Infrastructure-less Technology *Infrastructure-less technologies* “consists of device-to-device (D2D) communications that do not require any infrastructure. . . but rather allow devices in the vicinity to communicate directly.” Technologies include *WiFi-Direct*, *LTE-Direct*, and *Bluetooth* [**?**]. Infrastructure-less technology is also called the *opportunistic transmission paradigm* [**?**]. ShareTrace uses Bluetooth because of its energy efficiency and short range.

Data Reporting

Upload mode With *relay uploading*, "data is delivered as soon as collected." *Store-and-forward* "is typically used in delay-tolerant applications when campaigns do not need to receive data in real-time" [?]. Because ShareTrace is delay-tolerant, it uses store-and-forward uploading.

Methology *Individualized sensing* is "when each user accomplishes the requested task individually and without interaction with other participants." *Collaborative sensing* is when "users communicate with each other, exchange data[,] and help themselves in accomplishing a task or delivering information to the central collector. Users are typically grouped and exchange data exploiting short-range communication technologies, such as WiFi-[D]irect or Bluetooth... Note that systems that create maps merging data from different users are considered individual because users do not interact between each other to contribute" [?]. The methodology of sensing is similar to the *sensing scale* which is typically dichotomized as *personal* [? ?] (i.e., individualized) and *community* [?] or *group* [?] (i.e., collaborative). ShareTrace is inherently collaborative, relying on mobile devices to exchange actor URIs to estimate infection risk. Thus, collaborative sensing is used.

Timing *Timing* is based on whether devices "should sense in the same period or not." *Synchronous timing* "includes cases in which users start and accomplish at the same time the sensing task. For synchronization purposes, participants communicate with each other." *Asynchronous timing* occurs "when users perform sensing activity not in time synchronization with other users" [**?**]. ShareTrace requires synchronous timing, because contact sensing inherently requires synchronous communication between the involved devices.

1.1.4 Sensing Layer

Sensing Elements

Sensor Deployment *Dedicated deployment* involves the use of "non-embedded sensing elements," typically for a specific task. *Non-dedicated deployment* utilizes sensors that "do not require to be paired with other devices for data delivery but exploit the communication capabilities of mobile devices" [**?**]. ShareTrace relies on non-dedicated deployment since it relies on Bluetooth that is ubiquituous in modern-day mobile devices.

Sensor Activity *Always-on sensors* "are required to accomplish mobile devices['] basic functionalities, such as detection of rotation and acceleration... Activity recognition [i.e., context awareness]... is a very important feature that accelerometers enable." *On-demand sensors* "need to be switched on by users or exploiting an application running in the background. Typically, they serve more complex applications than always-on sensors and consume a higher amount fo energy" [**?**]. ShareTrace uses Bluetooth, which may be considered on-demand. While energy efficient, users do control when it is enabled. Ideally, ShareTrace would also use always-on sensors to enable Bluetooth with context awareness (i.e., that the user is carrying or nearby the device).

Acquisition *Homogeneous acquisition* "involves only one type of data and it does not change from one user to another one," while *heterogeneous acquisition* "involves different data types usually sampled from several sensors" [**?**]. ShareTrace is homogeneous, because all users sense the same data from one type of sensor.

Data Sampling

Sampling Frequency *Continuous sensing* "indicates tasks that are accomplished regularly and independently [of] the context of the smartphone or the user['s] activities." *Event-based sensing* is "data collection [that] starts after a certain event has occurred. In this context, an event can be seen as an active action from a user or the central collector, but also a given context awareness" [**?**]. ShareTrace sensing is continuous but would ideally be event-based to conserve device energy.

Sensing Responsibility When the *mobile device* is responsible, "devices or users take sampling decisions locally and independently from the central authority... When devices take sampling decisions, it is often necessary to detect the context [of the] smartphones and wearable devices... The objective is to maximize the utility of data collection and minimize the cost of performing unnecessary operations." When the *central collector* is responsible, they make "decisions about sensing and communicate them to the mobile devices" [**?**]. Given the human-centric nature of the ShareTrace sensing task, mobile devices are responsible.

User involvement *Participatory involvement* "requires active actions from users, who are explicitly asked to perform specific tasks. They are responsible to consciously meet the application requests by deciding when, what, where, and how to perform sensing tasks." *Opportunistic involvement* means that "users do not have direct involvement, but only declare their interest in joining a campaign and providing their sensors as a service. Upon a simple handshake mechanism between the user and the MCS platform, a MCS thread is generated on the mobile device (e.g., in the form of a mobile app), and the decisions of what, where, when, and how to perform the sensing are delegated to the corresponding thread. After having accepted the sensing process, the user is totally unconscious with no tasks to perform and data collection is fully automated... The smartphone itself is context-aware and makes decisions to sense and store data, automatically determining when its context matches the requirements of an application. Therefore, coupling opportunistic MCS systems with context-awareness is a crucial requirement" [**?**]. Earlier works on MCS refer to user involvement as the *sensing paradigm* [**? ? ?**]. ShareTrace is opportunistic, ideally with context-awareness.

of research in mobile contact tracing solutions, most notably being the joint effort by Apple and Google [**?**]. External reviews and surveys provide extensive comparison of existing solutions through the lenses of privacy, security, ethics, adversarial models, data management, scalability, interoperability, and more. References [**?**] and [**?**] provide thorough reviews of existing mobile contact tracing solutions with discussion of the techniques, privacy, security, and adversarial models. The former offers additional detail on the system architecture (i.e., centralized, decentralized, and hybrid), data management, and user concerns of existing solutions. Other notable reviews with similar discussion include [? ? ? ? ?]. Reference [?] provides a formal framework for defining aspects of privacy for proximity-based contact tracing. of different aspects of contact tracing [? ? ?]. A common finding across these surveys is that privacy and security continue to be of top concern for users, but contains some interesting nuance. For example, [?] surveyed over 10,000 individuals and found that there was over a 60-percent willingness to install a contact tracing mobile application. In a longitudinal study, [?] found that user preferences regarding privacy were stable over time. Moreover, they found that users had fewer privacy concerns for proximity-based contact tracing, in comparison to location-based contact tracing, but that there was security concerns for proximity-based tracking. Contrary to much of the developed techniques that emphasize a decentralized approach, [?] observed that mobile contact tracing applications that implement a centralized design are significantly more likely to be installed at the country level. Additionally, they found that individuals are generally more comfortable with their location data and identity information accessible to health-, state-, and federal-level authorities, compared to application developers, and the general public. former is the original white paper, which focuses on the motivation, design, and engineering details. Exlusive to [4] is a discussion on privacy, network roaming, protocol interoperability, and the usage of geolocation data. Furthermore, it includes more detail on the system model and data flow than [5]. The latter work formalizes risk propagation in a centralized setting and compares its efficacy to the Apple-Google framework [?]. A concise reiteration of the system model and deployment model is also provided. of the concepts will be reiterated in this work, it is assumed that the reader has read [4, 5]. This work offers the following contributions to the ShareTrace project and, more generally, to the development of next-generation applications that utilize self-soverign technologies to empower individuals take ownership of their personal data: utilizes the actor model to achieve scalable performance; passing on a temporal network; and their risk of infection. The two main advantages that ShareTrace offers over other digital contact-tracing approaches is its use of personal data stores that provide privacy by architectual design and its use of the contact network to infer the infection risk from indirect contact. While this thesis covers many aspects of ShareTrace, it is not comprehensive. The focus of this work is propagation as first proposed in [CITE] data Unlike other approaches that rely on device proximity to detect human interaction, ShareTrace executes iterative message passing on a factor graph to estimate a user's marginal posterior probability of infection (MPPI). To indicate its similarity to belief propagation, we refer to the ShareTrace algorithm as *risk propagation*. By considering both direct and indirect contact, [5] demonstrates that risk propagation is more effective than other proximity-based methods that only consider former. scalable formulation of risk propagation[1] that utilizes asynchronous, concurrent message passing on a temporal graph [16?]. While message passing has been studied under specific epidemiological models [? ?], our formulation allows us to contextualize risk propagation as a novel usage of a temporal graph that

[1] `https://github.com/share-trace`

does not require such assumptions to infer the transmission of disease. As a result, we introduce a form of reachability that can uniquely characterize the dynamics of message passing on a temporal graph. Our formulation of risk propagation aligns with its distributed extension, as introduced by [5], which has connections to the actor model of concurrent computing [**? ?**] and the "think-like-a-vertex" model of graph algorithms [22]. synthetic and real-world temporal graphs; (2) validate the accuracy of our new form of reachability on both synthetic and real-world graphs; (3) and briefly quantify the scalability of this implementation of risk propagation on synthetic graphs. To keep the scope of this work focused, we defer to [5] on the privacy and security aspects of ShareTrace.

Application	Task	Scheduling	Proactive	✓
			Reactive	
		Assignment	Centralized	
			Decentralized	✓
		Execution	Single task	✓
			Multi-tasking	
	User	Recruitment	Voluntary	✓
			Incentivized	
		Selection	Platform-centric	
			User-centric	✓
		Type	Consumer	✓
			Contributor	✓
Data	Management	Storage	Centralized	
			Distributed	✓
		Format	Structured	✓
			Unstructured	
		Dimension	Single dimension	✓
			Multi-dimensional	
	Processing	Pre-processing	Raw data	
			Filtering and denoising	✓
		Analytics	ML and data mining	✓
			Real-time	
		Post-processing	Statistical	
			Prediction	✓
Communication	Technologies	Infrastructured	Cellular	✓
			WLAN	✓
		Infrastructure-less	LTE-Direct	
			WiFi-Direct	
			Bluetooth	✓
	Reporting	Upload mode	Relay	
			Store and forward	✓
		Methodology	Individual	
			Collaborative	✓
		Timing	Synchronous	✓
			Asynchronous	
Sensing	Elements	Deployment	Dedicated	
			Non-dedicated	✓
		Activity	Always-on*	
			On-demand	✓
		Acquisition	Homogeneous	✓
			Heterogeneous	
	Sampling	Frequency	Continuous	✓
			Event-based*	
		Responsibility	Mobile device	✓
			Central collector	
		User involvement	Participatory	
			Opportunistic	✓

Table 1.1: ShareTrace classification using the four-layered architecture of an MCS application [?]. *Ideally, ShareTrace would operate with context-awareness.

Chapter 2

Actor-Based Contact Tracing

This chapter introduces an approach to privacy-preserving, proximity-based digital contact tracing. The approach is a realization of the distributed extension to the ShareTrace algorithm, *risk propagation*, that was first proposed in [5]. While this work includes enough detail to understand the original details about the system architecture and algorithm that define ShareTrace, the reader is encouraged to consult prior work [4, 5] to gain a complete understanding.

After outlining the system model, the original description of risk propagation is presented as both context and motivation for this work. Following this discussion is the main part of the chapter, which frames risk propagation as an online algorithm and an application of the actor model. Lastly, message reachability is introduced as a generalization of temporal reachability that can estimate the complexity of message-passing on temporal networks. Refer to Appendix A for the typographical conventions used in this chapter.

2.1 Global Risk Propagation

In risk propagation, computing infection risk is an inference problem in which the objective is to estimate a user's *marginal posterior infection probability* (MPIP). The prior infection probability of a user is derived from their symptoms [23], so it is called a *symptom score*. Because the posterior infection probability accounts for direct and indirect contact with other users[1], it is called an *exposure score*. In general, a *risk score* s_t is a timestamped infection probability where $s \in [0, 1]$ and $t \in \mathbb{N}$ is the time of its computation.

Computing the full joint probability distribution is intractable as it scales exponentially with the number of users. To circumvent this challenge, risk propagation uses a variation of message passing on a factor graph. Formally, let $G = (V, F, E)$ be a factor graph where V is the set of variable vertices, F is the set of factor vertices, and E is the set of edges incident between them [19]. A *variable vertex* v_i is a random variable such that $v_i : \Omega \to \{0, 1\}$, where the sample space is $\Omega = \{\text{healthy}, \text{infected}\}$ and

$$v_i(\omega) = \begin{cases} 0 & \text{if } \omega = \text{healthy} \\ 1 & \text{if } \omega = \text{infected.} \end{cases}$$

Thus, $p_t(v_i) = s_t$ is a risk score of user i. A *factor vertex* $f(v_i, v_j) = f_{ij}$ represents contact between users i, j such that f_{ij} is adjacent to v_i, v_j. Figure 1.1 depicts a factor graph that reflects the problem constraints. While the max-

[1]While previous work on ShareTrace [5] defines a *contact* as a contiguous 15 minutes in which the devices of two users are proximal, the Centers for Disease Control and Prevention technically considers these 15 minutes *over a 24-hour time window* [8].

sum algorithm aims to maximize the *joint* distribution [7, pp. 411–415], risk propagation aims to maximize *individual* MPIPs [5].

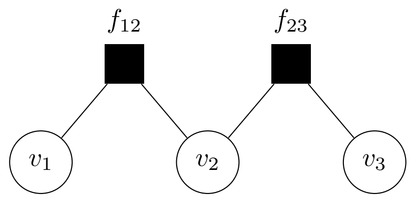

Figure 2.1: A factor graph of 3 variable vertices and 2 factor vertices.

2.2 System Model

The system model is similar to that in previous work [4, 5]. Figure 1.2 illustrates the corresponding data flow[2].

- Each user owns a *personal data store* (PDS), a form of cloud storage that empowers the user with ownership and access control over their data.
- Symptom scores are computed in a user's PDS to support integrating multiple streams of personal data [4]. While local symptom-score computation [4, 5] is more privacy-preserving, it is assumed that the user's PDS is a trusted entity.
- User device interactions serve as a proxy for proximal human interactions. This work does not assume a specific protocol, but does assume that the protocol can approximate the duration of contact with relative

[2] A *data-flow diagram* consists of data processors (circles), directed data flow (arrows), data stores (parallel lines), and external entities (rectangles) [14, pp. 437–438].

Global-Risk-Propagation(S, C)

$(V, F, E) \leftarrow$ Factor-Graph(C)
$n \leftarrow 1$
$\delta^{(n-1)} \leftarrow \infty$
for each $v_i \in V$
    $R_i^{(n-1)} \leftarrow$ top K of S_i
while $n \leq N$ **and** $\delta^{(n-1)} > \delta_{min}$
    for each $\{v_i, f_{ij}\} \in E$
        $m_{v_i \to f_{ij}}^{(n)} \leftarrow R_i^{(n-1)} \setminus \{\, m_{f_{ij} \to v_i}^{(k)} \mid k \in [1 \,..\, n-1] \,\}$
    for each $\{v_i, f_{ij}\} \in E$
        $m_{f_{ij} \to v_j}^{(n)} \leftarrow \max(\{0\} \cup \{\, \alpha \cdot s_t \mid s_t \in m_{v_i \to f_{ij}}^{(n)}, t \leq t_{ij} + \beta \,\})$
    for each $v_i \in V$
        $R_i^{(n)} \leftarrow$ top K of $\{\, m_{f_{ij} \to v_i}^{(k)} \mid k \in [1 \,..\, n], f_{ij} \in N_i \,\}$
    $\delta^{(n)} \leftarrow 0$
    for each $v_i \in V$
        $\delta^{(n)} \leftarrow \delta^{(n)} + \max R_i^{(n)} - \max R_i^{(n-1)}$
    $n \leftarrow n + 1$
return $\{\, \max R_i^{(n)} \mid v_i \in V \,\}$

accuracy and that communication with the actors of those contacted users can be established in a privacy-preserving manner.

- No geolocation data is collected [4]. As a decentralized, proximity-based solution, it is not necessary to collect user geolocation data. See **??** for a discussion of a geolocation-based design that was considered.

- Actor-based risk propagation is a distributed, online algorithm [9, pp.

791–818]. Previous work [4, 5] (see also **??**) formulates risk propagation as a centralized, offline algorithm that periodically aggregates all user data to estimate infection risk. To improve the privacy, scalability, and responsivenes of ShareTrace, this work designs risk propagation to avoid data aggregation and to estimate infection risk in near real-time.

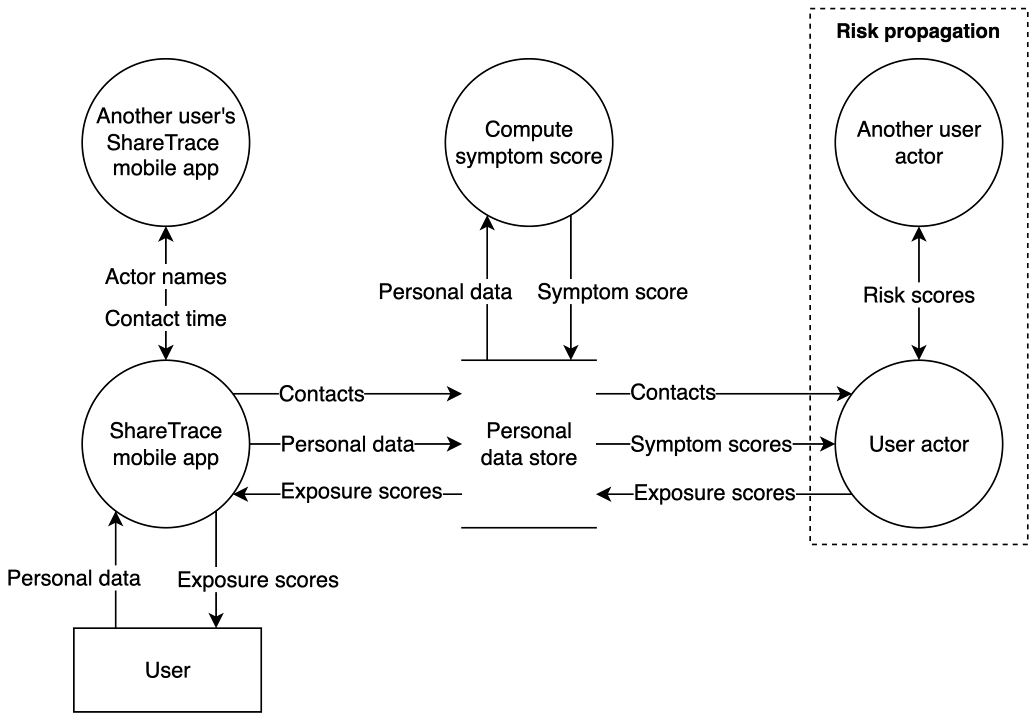

Figure 2.2: Actor-based ShareTrace data flow. *Contacts* include the actor name and contact time of all users with which the user came into close proximity. *Personal data* includes the user's demographics, reported symptoms, and diagnosis. It may also include machine-generated biomarkers and electronic health record data [4].

2.3 Local Risk Propagation

The only purpose of a factor vertex is to compute and relay messages between variable vertices. Thus, one-mode projection onto the variable vertices can be applied such that variable vertices $u, v \in V$ are adjacent if the factor vertex $f(u, v) \in F$ exists [27]. Figure 1.3 shows the modified topology. To send a mes-

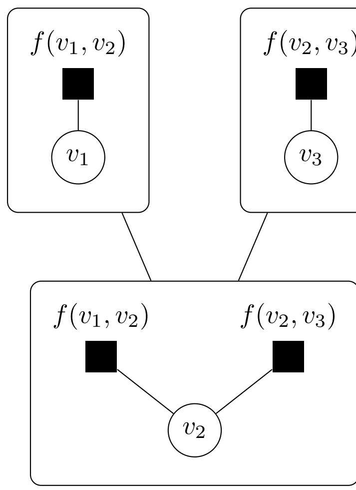

Figure 2.3: One-mode projection onto variable vertices of Figure 1.1. While the projected graph contains only variable vertices, the original factor vertices are also included to show how the original topology is modified.

sage to variable vertex v, variable vertex u applies the computation associated with the factor vertex $f(u, v)$. This modification differs from the distributed extension of risk propagation [5] in that we do not create duplicate factor vertices and messages in each user's PDS. By storing the contact time between users on the edge incident to their variable vertices, this modified topology is identical to the *contact-sequence* representation of a *contact network*, a kind of *time-varying* or *temporal network* in which a vertex represents a person and an edge indicates that two persons came in contact:

$$\{(u, v, t) \mid u, v \in V; u \neq v; t \geq 0\}, \tag{2.1}$$

where a triple (u, v, t) is called a *contact* [16]. Specific to risk propagation, t is the time at which users u and v *most recently* came in contact (see Section 1.3).

The usage of a temporal network in this work differs from its typical usage in epidemiology which focuses on modeling and analyzing the spreading dynamics of disease [10, 11, 18, 21, 25, 26, 28]. In contrast, this work uses a temporal network to infer a user's MPPI. As a result, Section 1.4 extends temporal reachability to account for both the message-passing semantics and temporal dynamics of the network. As noted by [16], the transmission graph provided by [26] "cannot handle edges where one vertex manages to not catch the disease." Notably, the usage of a temporal network in this work allows for such cases by modeling the possibility of infection as a continuous outcome.

TODO Factor graphs are useful for decomposing complex probability distributions and allowing for efficient inference algorithms.

However, as with risk propagation, and generally any application of a factor graph in which the variable vertices represent entities of interest (i.e., of which the marginal probability of a variable is desired), applying one-mode projection is a .

2.3.1 ShareTrace Actor System

As a distributed algorithm, risk propagation is specified from the perspective of an *actor*, which is equivalent to the paradiagm of *think- like-a-vertex* graph-processing algorithms [22]. Some variation exists on exactly how actor behavior is defined [1, 3, 12]. Perhaps the simplest definition is that the *behavior of an actor* is both its *interface* (i.e., the types of messages it can receive) and *state* (i.e., the internal data it uses to process messages) [12]. An *actor system*[3] is defined as the set of actors it contains and the set of unprocessed messages[4] in the actor mailboxes. An expanded definition of an actor system also includes a *local states function* that maps mail addresses to behaviors, the set of *receptionist actors* that can receive communication that is external to the actor system, and the set of *external actors* that exist outside of the actor system [1, 3]. Practically, a local states function is unnecessary to specify, so the narrower definition of an actor system is used. The remainder of this section describes the components of the ShareTrace actor system.

User Actor Behavior

Each user corresponds to an actor that participates in the message-passing protocol of risk propagation. Herein, the user of an actor will only be referred to as an *actor*. The following variant of the concurrent, object-oriented actor model is assumed to define actor behavior [2, 3].

[3] This is technically referred to as an *actor system configuration*.

[4] Formally, a *message* is called a *task* and is defined by a *tag*, a unique identifier; a *target*, the mail address to which the message is delivered; and a *communication*, the message content [1].

- An actor follows the *active object pattern* [12, 20] and the *Isolated Turn Principle* [12]. Specifically, the state change of an actor is carried out by instance- variable assignment, instead of the canonical Become primitive that provides a functional construct for pipelining actor behavior replacement [1–3]. The interface of a user actor is fixed in risk propagation, so the more general semantics of Become is unnecessary.

- The term “name” [1, 15] is preferred over “mail address” [1–3] to refer to the sender of a message. Generally, the mail address that is included in a message need not correspond to the actor that sent it. Risk propagation, however, requires this actor is identified in a risk-score message. Therefore, to emphasize this requirement, “name” is used to refer to both the identity of an actor and its mail address.

- An actor is allowed to include a loop with finite iteration in its behavior definition; this is traditionally prohibited in the actor model [1, 2].

- Because the following pseudocode to describe actor behavior mostly follows the conventions of [9] (see Appendix A), the behavior definition is implied by all procedures that take as input an actor.

The Create-Actor operation initializes an actor, which is equivalent to the *new expression* [1] or Create primitive [2, 3] with the exception that it only specifies the attributes (i.e., state) of an actor. As mentioned earlier, the behavior description of an actor is implied by the procedures that require an actor as input. Every actor A has the following attributes.

- $A.name$: an identifier that enables actors to communicate with it [1, 15].

- *A.contacts*: a dictionary (see Appendix A) that maps an actor name to a timestamp. That is, if user i of actor A_i comes in contact with user j of actor A_j, then the *contacts* of A_i contains the name of A_j along with the most recent contact time for users i and j. The converse holds for user j. This is an extension of the *acquaintance vector* [15] or *acquaintance list* [1, 3].

- *A.score*: the user's current exposure score. This attribute is either a default risk score (see Default-Risk-Score), a risk score sent by an actor of some contacted user[5], or a symptom score of the user.

- *A.cache*: a dictionary that maps a time interval to a risk score; used to tolerate synchronization delays between a user's device and actor (see Section 1.3.1).

Create-Actor

$A.name \leftarrow$ Create-Name
$A.contacts \leftarrow \emptyset$
$A.score \leftarrow$ Default-Risk-Score(A)
$A.cache \leftarrow \emptyset$
return A

Risk scores and contacts have finite relevance, which is parametrized by a *liveness* or *relevance duration* $\Delta t_s, \Delta t_c > 0$, respectively. The relevance of risk scores and contacts is important, because it influences how actors pass messages. For example, actors do not send irrelevant risk scores or relevant

[5] As discussed later (see Handle-Contact-Message in Section 1.3.1), it possible for an actor i to send a message to an actor j but not be a contact of actor j.

DEFAULT-RISK-SCORE(A)

1: $s.value \leftarrow 0$

2: $s.t \leftarrow 0$

3: $s.sender \leftarrow A.name$

4: **return** s

risk scores to irrelevant contacts. The *time-to-live* (TTL) of a risk score or contact is the remaining duration of its relevance. In the following operations, $s.t$ denotes the time at which the risk score was originally computed, $c.t$ is the contact time, and GET-TIME returns the current time.

SCORE-TTL(s)

1: **return** $\max\{0, \Delta t_s - (\text{GET-TIME} - s.t)\}$

CONTACT-TTL(c)

1: **return** $\max\{0, \Delta t_c - (\text{GET-TIME} - c.t)\}$

The interface of a user actor is defined by two types of messages: contact messages and risk-score messages. A *contact message* c contains the name $c.name$ of the actor whose user was contacted and the contact time $c.t$. A *risk-score message* s is simply a risk score along with the actor's name $s.sender$ that sent it. A risk score previously defined as the ordered pair (r, t) (see Section 1.3) is represented as the attributes r and $s.t$. The following sections discuss how a contact message and risk-message are processed by an actor.

Handling Contact Messages There are two ways in which a user actor can receive a contact message. The first, technically correct approach is for a receptionist actor to mediate the communication between the user actor and the PDS so that the user actor can retrieve its user's contacts. The second approach is to relax this formality and allow the user actor to communicate with the PDS directly[6].

The HANDLE-CONTACT-MESSAGEoperation defines how a user actor processes a contact message. A contact is assumed to have finite relevance which is parametrized by the *contact time-to-live* $\Delta t_c > 0$. A contact whose contact time occurred no longer than Δt_c time ago is said to be *alive*. Thus, a user actor only adds a contact if it is alive. Regardless of whether the

HANDLE-CONTACT-MESSAGE(A, c)

if CONTACT-TTL$(c) > 0$
    $c.key \leftarrow c.name$
    $c.threshold \leftarrow 0$
    INSERT$(A.contacts, c)$
    START-CONTACTS-REFRESH- TIMER(A)
    SEND-CURRENT-OR-CACHED(A, c)

START-CONTACTS-REFRESH- TIMER(A)

$oldest \leftarrow$ MINIMUM$(A.contacts)$
if $oldest \neq$ NIL
    $x.key \leftarrow$ "contacts"
    START-TIMER$(x,$ CONTACT-TTL$(oldest))$

contact is alive, the user actor attempts to send a risk-score message that is derived from its current exposure score or a cached risk score (see Section 1.3.1):

[6] If the PDS itself is an actor, then a push-oriented dataflow could be implemented, where the user actor receives contact messages (and symptom-score messages). This would be more efficient and timely than a pull-oriented dataflow in which the PDS is a passive data store that requires the user actor or receptionist to poll it for new data.

```
Handle-Contacts-Refresh- Timer(x)
for each c ∈ A.contacts
    if Contact-Ttl(c) ≤ 0
        Delete(A.contacts, c)
Start-Contacts-Refresh- Timer(A)
```

Like contacts, each risk score is assumed to have finite relevance

```
Send-Current-Or-Cached(A, c)
if Should-Receive(c, A.score)
    Send(A, c, A.score)
else
    s ← Cache-Max(A.cache, c.t + β)
    if s ≠ nil and Score-Ttl(s) > 0
        Send(A, c, s)
```

```
Send(A, c, s)
s′.value ← α · s.value
s′.t ← s.t
s′.sender ← A.name
Send(c, s′)
Update-Threshold(c, s′)
```

```
Should-Receive(c, s)
return c.threshold < s.value
 and c.t + β ≥ s.t
 and Score-Ttl(s) > 0
 and c.name ≠ s.sender
```

UPDATE-THRESHOLD(c, s)

```
threshold ← γ · s.value
if threshold > c.threshold
    c.threshold ← threshold
    START-THRESHOLD-TIMER(c, s)
```

START-THRESHOLD-TIMER(c, s)

```
x.key ← c.name + " threshold"
x.name ← c.name
START-TIMER(x, SCORE-TTL(s))
```

HANDLE-THRESHOLD-TIMER(A, x)

```
c ← SEARCH(A.contacts, x.name)
if c ≠ NIL
    s ← CACHE-MAX(A.cache, c.t + β)
    if s ≠ NIL and SCORE-TTL(s) > 0
        c.threshold ← γ · s.value
        START-THRESHOLD-TIMER(c, s)
    else
        c.threshold ← 0
```

that is parametrized by the *score time-to-live* $\Delta t_s > 0$ and evaluated by the operation IS-SCORE-ALIVE. To send an actor's current exposure score, the contact must be sufficiently recent. It is assumed that risk scores computed after a contact occurred have no effect on the user's exposure score.

To account for the disease incubation period, a delay in reporting symptoms, or a delay in establishing actor communication, a time buffer $\beta \geq 0$ is considered. That is, a risk score is not sent to a contact if IS-CONTACT-

Recent returns false.

The Transmitted operation is used to generate risk scores that are sent to other actors. It is assumed that contact only implies an incomplete transmission of risk between users. Thus, when sending a risk score to another actor, the value of the risk score is scaled by the *transmission rate* $\alpha \in (0, 1)$. Notice that the time of the risk score is left unchanged; the act of sending a risk-score message is independent of when the risk score was first computed.

If line 1 of Send-Current-Or-Cached evaluates to false, then the actor attempts to retrieve the maximum cached risk-score message based on the buffered contact time. If such a message exists and is alive, a risk-score message is derived and sent to the contact. The Send operation follows the semantics of the Send-To primitive [1, 2] or the *send command* [3].

Handling Risk-Score Messages Upon receiving a risk-score message, an actor executes the following operation.

```
HANDLE-RISK-SCORE-MESSAGE(A, s)
CACHE-INSERT(A.cache, s)
if s.value > A.score.value
    previous ← A.score
    A.score ← s
    if previous ≠ DEFAULT-RISK-SCORE(A)
        START-SCORE-REFRESH-TIMER(A)
PROPAGATE(A, s)
```

The Update-Actor operation is responsible for updating an actor's state, based on a received risk-score message. Specifically, it stores the message inside the actor's interval cache $A.cache$, assigns the actor a new exposure score and send coefficient (discussed below) if the received risk-score value exceeds that of the current exposure score, and removes expired contacts. In previous

Start-Score-Refresh-Timer(A)

$x.key \leftarrow$ "score"
Start-Timer(x, Score-Ttl($A.score$))

Handle-Score-Refresh-Timer(A, x)

$s \leftarrow$ Cache-Max(Get-Time)
if $s =$ nil
    $s \leftarrow$ Default-Risk-Score(A)
$A.score \leftarrow s$
if $s \neq$ Default-Risk-Score(A)
    Start-Score-Refresh-Timer(A)

work [5], risk propagation assumes synchronous message passing, so the notion of an iteration or inter-iteration difference threshold can be used as stopping conditions. However, as a streaming algorithm that relies on asynchronous message passing, such stopping criteria are unnatural. Instead, the following heuristic is applied and empirically optimized to minimize accuracy loss and maximize efficiency. Let $\gamma > 0$ be the *send coefficient* such that an actor only sends a risk-score message if its value exceeds the actor's *send threshold* $A.threshold$ (see line **??** in Propagate).

Assuming a finite number of actors, any positive send coefficient γ guarantees that a risk-score message will be propagated a finite number of times. Because the value of a risk score that is sent to another actor is scaled by the transmission rate α, its value exponentially decreases as it propagates away from the source actor with a rate constant $\log \alpha$.

As with the Send-Current-Or-Cached operation, a risk-score message must be alive and relatively recent for it to be propagated. As in previous work [5], factor marginalization is achieved by not propagating the received message to the actor who sent it. The logic of Propagate differs, however, in two ways. First, it is possible that no message is not propagated to a contact. The intent of sending a risk-score message is to update the exposure score of other actors. However, previous work [5] required that a “null” message with a risk-score value of 0 is sent. Sending such ineffective messages incurs additional communication overhead. The second difference is that only the *most recent* contact time is used to determine if a message should be propagated to a contact. Contact times determine what messages are relevant. Given two contact times t_1, t_2 such that $t_1 \leq t_2$, then any risk score with time $t \leq t_1 + \beta$ also satisfies $t \leq t_2 + \beta$. Thus, storing multiple contact times is unnecessary.

Propagate(A, s)

```
for each c ∈ A.contacts
    if Should-Receive(c, s)
        Send(A, c, s)
```

Risk Score Caching

For two actors to communicate, each must have the other actor in their contacts (see Section 1.3.1). Recall that an actor must retrieve these contacts from the user’s PDS, which subsequently requires synchronization with the user’s mobile device (see Figure 1.2). While the user's device can locally store contacts from proximal devices and symptoms of the user, an internet connection is needed to synchronize with the PDS and thus the user's actor. Therefore, it is not only possible but a reality that the user's mobile device and actor will not always be synchronized.

In the best case, this "lag" may only be a few seconds; in the worst case, the user's device is offline for several days. If δ_i (δ_j) is the delay between when the device of user i (ref. j) records a contact with user j (ref. i) and when its actor receives the corresponding contact message, then the delay between when actors A_i and A_j can communicate bidirectionally and when the contact actually occurred is $\delta_{ij} = \max(\delta_i, \delta_j)$. Such dissonance between the "true" state of the world (i.e., when users actually came in contact) and that known to the network of actors could impact the accuracy of risk propagation, which assumes such delays are nonexistent. To address this issue, each actor maintains a cache of received risk-score messages such that it can still send a message that reflects its previous state to a contact that was significantly delayed.

An *interval cache* C is a dictionary that maps a finite time interval (key) to a data element (value). In a typical cache, the *time-to-live* (TTL) of an element is a fixed duration after which the element is removed or *evicted*. In an interval cache, however, the TTL of an element is determined by its associated interval and the current time. That is, an interval cache is like a series of sliding windows, where each window corresponds to an interval that can hold a single element. Thus, the TTL of an element is the duration between the start of its interval and the start of the earliest interval in the cache.

An interval cache maintains N contiguous, half-closed (start-inclusive) intervals, each of duration Δt. An interval cache contains N_a *look-ahead intervals* and N_b *look-back intervals* such that $N = N_b + N_a$. Look- ahead (resp. look-back) intervals allow elements to be associated with intervals whose start times are later (resp. earlier) than the current time t. The *look-back duration* Δt_b and *look-ahead duration* Δt_a are defined as

$$\Delta t_b = N_b \cdot \Delta t$$
$$\Delta t_a = N_a \cdot \Delta t.$$

The *range* of the interval cache is $[C.start, C.end)$, where

$$C.start = C.\mathit{refresh} - \Delta t_b$$
$$C.end = C.\mathit{refresh} + \Delta t_a,$$

and $C.\mathit{refresh}$ is the time at which the cache was last refreshed.

An interval cache is a "live" data structure, so the range must be updated periodically to reflect the advancement of time. Furthermore, intervals and their associated elements that are no longer contained in the range must be evicted. This process of updating the range and evicting cached elements is called *refreshing the cache*. The *refresh period* $T > 0$ of an interval cache is the duration until the range is updated, based on the current time t. Depending on the interval duration Δt and the nature of the data that is being cached, the refresh period may be on the order of seconds or days. To recognize when a refresh is necessary, the cache maintains the attribute $C.refresh$, which is the time of the previous refresh. The operation Cache-Refreshupdates the range if at least T time has elapsed since the previous refresh and then removes all expired elements.

Cache-Refresh(C)

$t \leftarrow$ Get-Time
if $t - C.refresh > T$
  $C.start \leftarrow t - \Delta t_b$
  $C.end \leftarrow t + \Delta t_a$
  $C.refresh \leftarrow t$
  for each $x \in C$
    if $x.key < C.start$
      Delete(C, x)

The operation Cache-Insertrefreshes the cache, if necessary, and merges into the cache the element pointed to by x if its timestamp $x.t$ is in the range. Keys in the interval cache are interval start times and are lazily computed (line 2) to avoid storing intervals with no associated element. By not storing all intervals explicitly, the interval cache only achieves $O(N)$ space complexity when each interval has an associated element. The Mergeoperation (line 7) can be as trivial as replacing the existing value. For risk propagation, the interval cache associates with each interval the newest risk score of maximum value.

Cache-Key(C, x)

return $C.start + \left\lfloor \frac{x.t - C.start}{\Delta t} \right\rfloor \cdot \Delta t$

Cache-Insert(C, x)

if $x.t \in [C.start, C.end)$
  $x.key \leftarrow$ Cache-Key(C, x)
  $x_{old} \leftarrow$ Search(C, x)
  if $x_{old} =$ nil
    $x_{new} \leftarrow x$
  else
    $x_{new} \leftarrow$ Merge(x_{old}, x)
  Insert(C, x_{new})

The intention of sending a cached risk score to a contacted user actor is to account for the delay between when the contact occurred and when the actors establish communication. Therefore, the cached risk score that should be sent is that which would have been the current exposure score at the time the users came into contact. That is, each user actor should send the maximum risk score whose cache interval ends at or before the time of contact, accounting for the time buffer β (see line 4 of Send-Current-Or-Cachedin Section 1.3.1). The operation Cache-Maxis used to carry out this query.

Cache-Max(C, t)

return Maximum$(\{x \in C \mid x.key <$ Cache-Key$(t)\})$

An interval cache is implemented by augmenting a hash table [**?** , pp. 253–285] with the aforementioned attributes and parameters. By using a hash table, the interval cache offers $\Theta(1)$ average-case insert, search, and delete operations. Reference [**?** , pp. 348–354] implements an *interval tree* by augmenting a red-black tree. However, insert, delete, and search operations on a red-black tree require $\Theta(\log N)$ in the average case.

2.4 Message Reachability

A fundamental concept of reachability in temporal networks is the *time-respecting path*: a contiguous sequence of contacts with nondecreasing time. Thus, vertex v is *temporally reachable* from vertex u if there exists a time-respecting path from u to v, denoted $u \to v$ [24]. The following quantities are derived from the notion of a time-respecting path and help quantify reachability in a time-varying network [16].

- The *influence set* $I(v)$ *of vertex* v is the set of vertices that v can reach by a time-respecting path.
- The *source set* $S(v)$ *of vertex* v is the set of vertices that can reach v by a time-respecting path.
- The *reachability ratio* $f(G)$ *of a temporal network* G is the average influence-set cardinality of a vertex v.

Generally, a message-passing algorithm defines a set of constraints that determine when and what messages are sent from one vertex to another. Even if operating on a temporal network, those constraints may be more or less strict than requiring temporal reachability. As a dynamic process, message passing on a time-varying network requires a more general definition of reachability that can account for network topology *and* message-passing semantics [6].

Formally, the *message reachability from vertex u to vertex v* is the number of edges along the *shortest path* P that satisfy the message passing constraints,

$$m(u,v) = \sum_{(i,j)\in P} f(u,i,j,v),$$

where

$$f(u,i,j,v) = \begin{cases} 1 & \text{if all constraints are satisfied} \\ 0 & \text{otherwise.} \end{cases}$$

Vertex v is *message reachable* from vertex u if there exists a shortest path such that $m(u,v) > 0$. The *message reachability of vertex u* is the maximum message reachability from vertex u:

$$m(u) = \max\{m(u,v) \mid v \in V\}. \tag{2.2}$$

The temporal reachability metrics previously defined can be extended to message reachability by only considering the message-reachable vertices:

$$I_m(u) = \{v \in V \mid m(u,v) > 0\}$$
$$S_m(v) = \{u \in V \mid m(u,v) > 0\}$$
$$f_m(G) = \frac{\sum_{v\in V} |I_m(v)|}{|V|}.$$

Let $\mathbf{M}$ be the $|V| \times |V|$ *message-reachability matrix* of the temporal network

G such that vertices are enumerated $1, 2, \ldots, |V|$ and for each $m_{ij} \in \mathbf{M}$,

$$m_{ij} = \begin{cases} 1 & \text{if } m(i,j) > 0 \\ 0 & \text{otherwise.} \end{cases}$$

Then the cardinality of the influence set for vertex i is the number of nonzero elements in the ith row of $\mathbf{M}$:

$$|I_m(i)| = \sum_{j=1}^{|V|} m_{ij}. \tag{2.3}$$

Similarly, the cardinality of the source set for vertex j is the number of nonzero elements in the jth column of $\mathbf{M}$:

$$|S_m(j)| = \sum_{i=1}^{|V|} m_{ij}. \tag{2.4}$$

For risk propagation, let $H(x)$ be the *Heaviside step function*,

$$H(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

Then message reachability is defined as

$$m(u,v) = \sum_{(i,j) \in P} f_r(u,i) \cdot f_c(u,i,j) \tag{2.5}$$

where P is the set of edges along the shortest path $u \to v$ such that the actors are enumerated $0, 1, \ldots, |P| - 1$ and

$$f_r(u, i) = H(\alpha^i \cdot r_u - \gamma \cdot r_i) \tag{2.6}$$

$$f_c(u, i, j) = H(t_{ij} - t_u + \beta) \tag{2.7}$$

are the value and contact-time constraints in the Should-Receive operation (see Section 1.3.1), where (r_i, t_i) is the current exposure score for actor i and t_{ij} is the most recent contact time between actors i and j.

The value of (1.5) can be found by associating with each symptom score a unique identifier. If each actor maintains a log of the risk scores it receives, then the set of actors that receive the symptom score or a propagated risk score thereof can be identified. This set of actors defines the induced subgraph on which to compute (1.5) using a shortest-path algorithm [17].

Regarding efficiency, (1.2) to (1.4) provide the means to quantify the communication overhead of a given message-passing algorithm on a temporal network. Moreover, because such metrics capture the temporality of message passing, they can better quantify complexity than traditional graph metrics.

By relaxing the constraint (1.7), it is possible to estimate (1.5) with (1.6). The *estimated message reachability of vertex u to vertex v*, denoted $\hat{m}(u, v)$, is defined as follows. Based on (1.6),

$$\alpha^{\hat{m}(u,v)} \cdot r_u \leq \gamma \cdot r_v,$$

where the left-hand side is the value of the propagated symptom score for actor u when $\hat{m}(u, v) = 1$, and the right-hand side is the value required by some message-reachable actor v to propagate the message received by actor u or some intermediate actor. Solving for $\hat{m}(u,v)$,

$$\hat{m}(u,v) \leq f(u,v), \tag{2.8}$$

where

$$f(u,v) = \begin{cases} 0 & \text{if } r_u = 0 \\ |P| & \text{if } r_v = 0 \\ \log_\alpha \gamma + \log_\alpha r_v - \log_\alpha r_u & \text{otherwise.} \end{cases}$$

Equation (1.8) indicates that a lower send coefficient γ will generally result in higher message reachability, at the cost of sending possibly ineffective messages (i.e., risk scores that do not update the exposure score of another actor). Equation (1.8) also quantifies the effect of the transmission rate α. Unlike the send coefficient, however, the transmission rate is intended to be derived from epidemiology to quantify disease infectivity and should not be optimized to improve performance.

Given the multivariate nature of message reachability, it is helpful to visualize how it with various combinations of parameter values. Figure 1.4 includes several line plots of estimated message reachability $\hat{m}(u,v)$ with respect to the initial risk score magnitude of actor u.

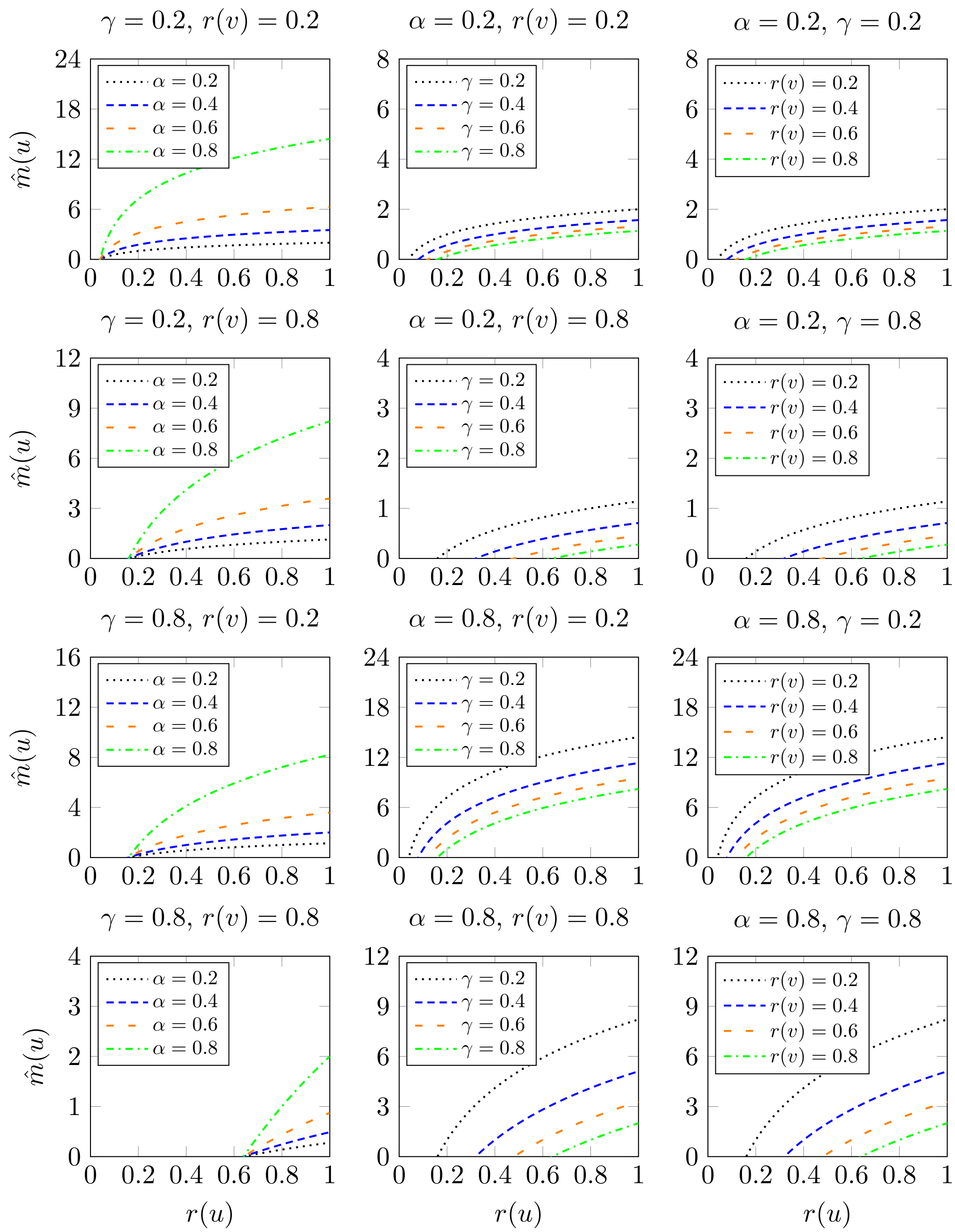

Figure 2.4: Estimated risk propagation message reachability $\hat{m}(u)$ for different values of the transmission rate α, the send tolerance γ, and the initial magnitude $r(v) = r_0(v)/\alpha$ of the destination user v with respect to the initial magnitude $r_u = r_0(u)/\alpha$ of the source user u.

2.5 Complexity

Risk propagation has worst-case time complexity $O(|C|)$ because the number of messages scales with the number of contacts. As noted in Section 1.4, however, this does not capture the temporal constraints of message passing on a time-varying contact network. This is a tighter bound than previous work [5] which states that the worst-case time complexity is $O(n^2)$, where n is the number of actors. The space complexity of risk propagation is also $O(|C|)$ because each actor must store their contacts and the messages received in their mailbox.

user actor receives the corresponding message. Assuming $B \leq L$, there are 3 cases to consider: $\delta \leq B$, $B < \delta \leq L$, and $\delta > L$. contacted user receives. The user will send

Appendix A

Typographical Conventions

A.1 Mathematics

Mathematical typesetting follows the guidance of [13].

A.2 Pseudocode

The pseudocode conventions used in this work mostly follow [9, pp. 21–24].

- Indentation indicates block structure.
- Looping and conditional constructs have similar interpretations to those in standard programming languages.
- Composite data types are represented as *objects*. Accessing an *atttribute* a of an object o is denoted $o.a$. A variable representing an object is a *pointer* or *reference* to the data representing the object. The special value NIL refers to the absence of an object.

- Parameters are passed to a procedure *by value*. That is, the "procedure receives its own copy of the parameters, and if it assigns a value to a parameters, the change is *not* seen by the calling procedure. When objects are passed, the pointer to the data representing the object is copied, but the object's attributes are not" [9, p. 23]. Thus, object attribute assignment "is visible if the calling procedure has a pointer to the same object" [9, p. 24].
- A **return** statement "immediately transfers control back to the point of call in the calling procedure" [9, p. 24].
- Boolean operators **and** and **or** are *short circuiting*.

The following conventions are specific to this work.

- Object attributes may be defined *dynamically* in a procedure.
- Variables are local to the given procedure, but parameters are global.
- The "$\leftarrow$" symbol is used to denote assignment, instead of "$=$".
- The "$=$" symbol is used to denote equality, instead of "$==$", which is consistent with the use of "$\neq$" to denote inequality.
- The "$\in$" symbol is used in **for** loops when iterating over a collection.
- Set-builder notation $\{x \in X \mid \textsc{Predicate}(x)\}$ is used to create a subset of a collection X in place of constructing an explicit data structure.

Bibliography

[1] Gul Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. PhD thesis, MIT, Cambridge, MA, 1985.

[2] Gul Agha. Concurrent object-oriented programming. *Commun. ACM*, 33(9):125–141, 1990.

[3] Gul Agha and Carl Hewitt. Concurrent programming using actors: Exploiting large-scale parallelism. In S. N. Maheshwari, editor, *Foundations of Software Technology and Theoretical Computer Science*, pages 19–41, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.

[4] Erman Ayday, Fred Collopy, Taehyun Hwang, Glenn Parry, Justo Elias Karell, James Kingston, Irene Ng, Aiden Owens, Brian Ray, Shirley Reynolds, Jenny Tran, Shawn Yeager, Youngjin Yoo, and Gretchen Young. Sharetrace: A smart privacy-preserving contact tracing solution by architectural design during an epidemic. Technical report, Case Western Reserve University, 2020.

[5] Erman Ayday, Youngjin Yoo, and Anisa Halimi. ShareTrace: An iterative message passing algorithm for efficient and effective disease risk assessment on an interaction graph. In *Proc. 12th ACM Con. Bioinformatics, Comput. Biology, Health Inform.*, BCB 2021, 2021.

[6] Alain Barrat and Ciro Cattuto. Temporal networks of face-to-face human interactions. In Petter Holme and Jari Saramäki, editors, *Temporal Netw.*, Underst. Complex Syst. Springer, 2013.

[7] Christopher M. Bishop. Pattern recognition and machine learning. In M. I. Jordan, Robert Nowak, and Bernhard Schoelkopf, editors, *Inf. Sci. Stat.* Springer, 2006.

[8] Centers for Disease Control and Prevention. Quarantine and isolation, 2021. `https://www.cdc.gov/coronavirus/2019-ncov/your-health/quarantine-isolation.html`.

[9] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, fourth edition, 2022.

[10] Meggan E. Craft. Infectious disease transmission and contact networks in wildlife and livestock. *Phil. Trans. R. Soc. B*, 370, 2015.

[11] Leon Danon, Ashley P. Ford, Thomas House, Chris P. Jewell, Gareth O. Roberts, Joshua V. Ross, and Matthew C. Vernon. Networks and the epidemiology of infectious disease. *Interdiscip. Perspect. Infect. Dis.*, 2011, 2011.

[12] Joeri De Koster, Tom Van Cutsem, and Wolfgang De Meuter. 43 years of actors: A taxonomy of actor models and their key properties. In *Proceedings of the 6th International Workshop on Programming Based on Actors, Agents, and Decentralized Control*, AGERE 2016, pages 31–40, New York, NY, 2016. Association for Computing Machinery.

[13] Chris Dyer, Kevin Gimpel, and Noah A. Smith. A short guide to typesetting math in NLP papers. `http://demo.clab.cs.cmu.edu/cdyer/short-guide-typesetting.pdf`.

[14] Susan Fowler and Victor Stanwick. *Web Application Design Handbook: Best Practices for Web-Based Software*. Morgan Kaufmann Publishers, San Francisco, CA, 2004.

[15] Carl Hewitt and Henry Baker. Laws for communicating parallel processes. Technical report, Massachusetts Institute of Technology, 1977. `http://hdl.handle.net/1721.1/41962`.

[16] Petter Holme and Jari Saramäki. Temporal networks. *Phys. Rep.*, 519, 2012.

[17] Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24, 1977.

[18] Andreas Koher, Hartmut H. K. Lentz, James P. Gleeson, and Philipp Hövel. Contact-based model for epidemic spreading on temporal networks. *Phys. Rev. X*, 9, 2019.

[19] Frank R. Kschischang, Brendan J. Frey, and Hans A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans. Inf. Theory*, 47, 2001.

[20] R. Greg Lavender and Douglas C. Schmidt. Active object – an object behavioral pattern for concurrent programming, 1996. `https://csis.pace.edu/~marchese/CS865/Papers/Act-Obj.pdf`.

[21] Andrey Y. Lokhov, Marc Mézard, Hiroki Ohta, and Lenka Zdeborová. Inferring the origin of an epidemic with a dynamic message-passing algorithm. *Phys. Rev. E*, 90, 2014.

[22] Robert McCune, Tim Weninger, and Greg Madey. Thinking like a vertex: A survey of vertex-centric frameworks for large-scale distributed graph processing. *ACM Comput. Surveys*, 48, 2015.

[23] Cristina Menni, Ana M Valdes, Maxim B Freidin, Carole H Sudre, Long H Nguyen, David A Drew, Sajaysurya Ganesh, Thomas Varsavsky, M Jorge Cardoso, Julia S El-Sayed Moustafa, Alessia Visconti, Pirro Hysi, Ruth C E Bowyer, Massimo Mangino, Mario Falchi, Jonathan Wolf, Sebastien Ourselin, Andrew T Chan, Claire J Steves, and Tim D Spector. Real-time tracking of self-reported symptoms to predict potential COVID-19. *Nat. Med.*, 26, 2020.

[24] James Moody. The importance of relationship timing for diffusion. *Soc. Forces.*, 81, 2002.

[25] Romualdo Pastor-Satorras, Claudio Castellano, Piet Van Mieghem, and Alessandro Vespignani. Epidemic processes in complex networks. *Rev. of Mod. Phys.*, 87, 2015.

[26] Christopher S. Riolo, James S. Koopman, and Stephen E. Chick. Methods and measures for the description of epidemiologic contact networks. *J. Urban Health*, 78, 2001.

[27] Tao Zhou, Jie Ren, Matú š Medo, and Yi-Cheng Zhang. Bipartite network projection and personal recommendation. *Phys. Rev. E*, 76, 2007.

[28] Lorenzo Zino and Ming Cao. Analysis, prediction, and control of epidemics: A survey from scalar to dynamic network models. *IEEE Circuits Syst. Mag.*, 21, 2021.