SHARETRACE: PROACTIVE CONTACT TRACING WITH ASYNCHRONOUS MESSAGE PASSING

by

RYAN TATTON

Submitted to the Department of Computer and Data Sciences

in partial fulfillment of the requirements for the degree of

Master of Science in Computer and Information Science

at the

CASE WESTERN RESERVE UNIVERSITY

May 2025

CASE WESTERN RESERVE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

We hereby approve the thesis of

Ryan Tatton

candidate for the degree of

Master of Science in Computer and Information Science

Committee Chair

Erman Ayday, PhD

Committee Members

Youngjin Yoo, PhD

Harold Connamacher, PhD

Michael Lewicki, PhD

Date of Defense

7 February 2025

We also certify that written approval has been obtained for any proprietary material contained therein.

Contents

List of Tables **4**

List of Figures **5**

1 Introduction **9**

2 Proposed Design **12**

2.1 Synchronous Risk Propagation 14

2.1.1 Variable Messages 14

2.1.2 Factor Messages 15

2.1.3 Termination 17

2.2 Asynchronous Risk Propagation 18

2.2.1 Actor Behavior 20

2.2.2 Message Reachability 27

2.3 System Design 31

2.3.1 Application Layer 31

2.3.2 Data Layer 34

2.3.3 Communication Layer 36

2.3.4 Sensing Layer 39

3 Evaluation **43**

3.1 Reference Implementation 43

3.2 Experimental Design 47

3.3 Results 56

3.3.1 Experiment 1 56

3.3.2 Experiment 2 64

3.3.3 Experiment 3 68

4 Conclusion **72**

A Prior Designs and Implementations **74**

A.1 Thinking Like a Vertex 75

A.2 Factor Subgraph Actors 77

A.3 Driver-Monitor-Worker Framework 78

A.4 Projected Subgraph Actors 80

A.5 Contact Search 81

B Data Structures **86**

C Pseudocode Conventions **87**

List of Tables

2.1 Classifying ShareTrace as mobile crowdsensing 42

3.1 Default parameter values for experiments 50

3.2 Experiment configurations 51

3.3 Barabasi-Albert n_0 values for Experiment 3 52

3.4 Barabasi-Albert m_0 values for Experiment 3 53

3.5 Random regular d values for Experiment 3 54

3.6 Watts-Strogatz k values for Experiment 3 55

3.7 Accuracy percentiles 59

3.8 Send coefficient optimality percentiles 61

3.9 Relative messages received percentiles 61

3.10 Message reachability percentiles 62

3.11 Contact network degree percentiles 64

3.12 Hypothesis tests for message-passing runtime 67

3.13 Models of message-passing runtime 70

List of Figures

2.1 Factor graph 13

2.2 Exponential decay of risk scores 16

2.3 One-mode projection of the factor graph 19

2.4 Message reachability for asynchronous risk propagation 30

3.1 Arrow diagram of the reference implementation 44

3.2 Cumulative accuracy distributions 57

3.3 Send coefficient optimality distributions 60

3.4 Message-passing efficiency 62

3.5 Message reachability cumulative distributions 63

3.6 Contact network degree distributions 65

3.7 Correlation matrix of Experiment 1 dataset attributes 66

3.8 Message-passing runtimes 69

3.9 Message-passing runtimes with regression lines 71

A.1 ShareTrace batch-processing architecture 76

A.2 Driver-monitor-worker framework 79

A.3 Example of contact search 83

A.4 Central angle of a great circle 84

Acknowledgments

I am sincerely thankful to Profs. Ayday and Yoo for their mentorship and collaboration on the ShareTrace project and the various xLab endeavors. As a result of their generous support, I have learned about the challenges of distributed computing and the exciting prospects of applying decentralized and self-sovereign technologies to solve impactful problems. I believe that my experiences with Profs. Ayday Yoo have significantly contributed to my first career opportunity at Amazon; and for that, I cannot thank them enough.

SHARETRACE: PROACTIVE CONTACT TRACING WITH ASYNCHRONOUS MESSAGE PASSING

by

RYAN TATTON

Abstract

Contact tracing is a non-pharmaceutical intervention that aims to control the spread of disease by identifying and quarantining infected individuals and those with whom they came in close contact. Numerous approaches to digital contact tracing have been proposed in the context of the coronavirus disease 2019 pandemic. Decentralized digital contact tracing limits the sharing of personal data, but no prior work has utilized non-diagnostic information *and* indirect contacts to effectively estimate infection risk. This work improves on prior efforts of ShareTrace by providing an asynchronous message-passing algorithm that permits a fully decentralized deployment. A reference implementation is provided and evaluated for accuracy, efficiency, and scalability.

Chapter 1

Introduction

Contact tracing is a non-pharmaceutical intervention that aims to halt the spread of infectious disease by identifying and quarantining individuals that have been physically proximal with the infected [73]. To combat the pandemic of coronavirus disease 2019 (COVID-19) [29, 86, 104], numerous approaches to *digital contact tracing* (DCT) have been proposed [15, 32, 76, 83, 94]. While the implementation details vary considerably, all approaches to DCT aim to mitigate the spread of infection by automatically informing individuals of their infection risk.

A popular form of DCT is *proximity tracing*, which uses device-to-device (D2D) communication [37] to approximate in-person interactions. While multiple protocols for proximity detection exist, Bluetooth Low Energy (BLE) is typically used because of its relative accuracy, energy efficiency, and broad support in mobile devices [76, 83]. Proximity tracing entails generating and exchanging pseudonyms (i.e., ephemeral identifiers, contact identifiers) between nearby mobile devices. What differentiates DCT applications is how these pseudonyms are generated and utilized. *Decentralized DCT* determines an individual's contacts and infection risk locally, thus avoiding the aggregation of sensitive personal data. Afroogh et al. [1], Oyibo et al. [69], and Simko et al. [85] all find that privacy and security are paramount to the adoption of DCT, which is a key determinant of achieving epidemic control [73]. While decentralization is not sufficient for strong privacy and security guarantees, it generally makes attaining such guarantees more feasible.

In the *broadcast model* of decentralized DCT, an infected individual uploads their information to a central service that allows others to determine if they possess any of the pseudonyms belonging to that individual [76]. A major limitation of the broadcast model is that an individual's infection risk does not account for indirect contacts, which can substantially improve the effectiveness of DCT [73]. Moreover, the broadcast model assumes an accurate diagnostic test exists and is broadly accessible and trusted by the public. To address the limitations of the broadcast model, Ayday, Yoo, and Halimi [3] proposed ShareTrace, which uses a message-passing algorithm that incorporates non-diagnostic information and indirect contacts when estimating infection risk. However, the authors assume a centralized deployment, which exposes the entire contact network and personal data to a single entity.

Other approaches to *message-oriented DCT* [16, 77] provide decentralization, but do not account for non-diagnostic information and indirect contacts. Recently, Cherini et al. [15] proposed extending the broadcast model such that mobile devices also share the pseudonyms of their indirect contacts during

D2D interactions, but still depend on diagnostic testing. Gupta et al. [32] incorporate non-diagnostic information to proactively determine an individual's infection risk, but do not account for indirect contacts.

This work proposes an asynchronous formulation of the message-passing algorithm proposed by Ayday, Yoo, and Halimi [3], which permits a decentralized deployment of ShareTrace. In this way, this work addresses the limitations of other message-oriented DCT designs [16, 77] and more recent works [15, 32] that do not incorporate both non-diagnostic information and indirect contacts when estimating infection risk. While message passing has been studied under specific epidemiological models [45, 56], this work does not require such assumptions to infer the transmission of disease.

The remainder of this work is organized as follows. Chapter 2 begins with the algorithmic foundations of ShareTrace: the risk propagation message-passing algorithm. Risk propagation is first presented as a synchronous, offline algorithm, which is consistent with prior work [3]. Chapter 2 then provides an asynchronous formulation using the actor model. Chapter 3 evaluates a reference implementation of the proposed design. Various data distributions and contact network topologies are used to determine the parameter values that optimize asynchronous risk propagation for accuracy and efficiency. The runtime performance of the reference implementation is similarly evaluated. Chapter 4 concludes with directions for future work. Appendix A includes prior designs and implementations. Appendix B and Appendix C respectively describe the data structures and pseudocode conventions that are used throughout this work.

Chapter 2

Proposed Design

Risk propagation is a message-passing algorithm that estimates an individual's infection risk by considering their demographics, symptoms, diagnosis, and contact with others. Formally, a *risk score* $s_t \in [0, 1]$ is a timestamped infection probability where $t \in \mathbb{N}$ is the time of its computation. Thus, an individual with a high risk score is likely to test positive for the infection and poses a significant health risk to others. There are two types of risk scores: *symptom scores*, or prior infection probabilities, which account for an individual's demographics, symptoms, and diagnosis [10, 62]; and *exposure scores*, or posterior infection probabilities, which incorporate the risk of direct and indirect contact with others.

Given their recent risk scores and contacts, an individual's exposure score is derived by marginalizing over the joint infection probability distribution. Naively computing this marginalization scales exponentially with the number of variables (i.e., individuals). To circumvent this intractability, the joint distribution is modeled as a factor graph, and an efficient message-passing procedure is employed to compute the marginal probabilities with a time complexity that scales linearly in the number of factor vertices (i.e., contacts).

Let $G = (X, F, E)$ be a *factor graph* where X is the set of variable vertices, F is the set of factor vertices, and E is the set of edges incident between them [52]. A *variable vertex* $x : \Omega \to \{0, 1\}$ is a random variable that represents the infection status of an individual, where the sample space is $\Omega = \{\textit{healthy}, \textit{infected}\}$ and

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

Thus, $p_t(x_i) = s_t$ is a risk score of the i-th individual that was computed at time t. A *factor vertex* $f : X \times X \to [0, 1]$ defines the transmission of infection risk between two contacts. Specifically, contact between the i-th and j-th individual is represented by the factor vertex $f(x_i, x_j) = f_{ij}$, which is adjacent to the variable vertices x_i, x_j. Figure 2.1 depicts a factor graph that reflects the domain constraints.

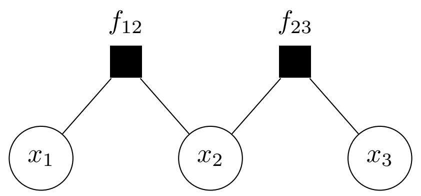

Figure 2.1: A factor graph of 3 variable vertices and 2 factor vertices.

2.1 Synchronous Risk Propagation

Ayday, Yoo, and Halimi [3] first proposed risk propagation as a synchronous, iterative message-passing algorithm that uses the factor graph to compute exposure scores. The first input to Risk-Propagation is the set family S, where

$$S_i = \{\, s_t \mid \tau - t < T_s \,\} \in S \tag{2.1}$$

is the set of recent risk scores of the i-th individual. The second input to Risk-Propagation is the contact set

$$C = \{\, (i, j, t) \mid i \neq j, \tau - t < T_c \,\} \tag{2.2}$$

such that (i, j, t) is the *most recent* contact between the i-th and j-th individual that occurred from time t until at least time $t + \delta$, where $\delta \in \mathbb{N}$ is the *minimum contact duration.* Naturally, risk scores and contacts have finite relevance, so (2.1) and (2.2) are constrained by the *risk score expiry* $T_s \in \mathbb{N}$ and the *contact expiry* $T_c \in \mathbb{N}$, respectively. The *reference time* $\tau \in \mathbb{N}$ defines the relevance of the inputs and is assumed to be the time at which Risk-Propagation is invoked. For notational simplicity in Risk-Propagation, let X be a set. Then $\max X = 0$ if $X = \emptyset$.

2.1.1 Variable Messages

The current exposure score of the i-th individual is defined as $\max S_i$. Hence, a *variable message* $\mu_{ij}^{(n)}$ from the variable vertex x_i to the factor vertex f_{ij} during the n-th iteration is the set of maximal risk scores $R_i^{(n-1)}$ from the previous $n-1$ iterations that were not derived by f_{ij}. In this way, risk propagation is reminiscent of the max-sum algorithm; however, risk propagation aims to maximize *individual* marginal probabilities rather than the joint distribution [9, pp. 411–415].

2.1.2 Factor Messages

A *factor message* $\lambda_{ij}^{(n)}$ from the factor vertex f_{ij} to the variable vertex x_j during the n-th iteration is an exposure score of the j-th individual that is based on interacting with those at most $n-1$ degrees separated from the i-th individual. This population is defined by the subgraph induced in G by

$$\{\, v \in X \cap F \setminus \{x_j, f_{ij}\} \mid d(x_i, v) \leq 2(n-1) \,\},$$

where $d(u, v)$ is the shortest-path distance between the vertices u, v. The computation of a factor message assumes the following.

1. Contacts have a nondecreasing effect on an individual's exposure score.
2. A risk score s_t is *relevant* to the contact (i, j, t_{ij}) if $t < t_{ij} + \beta$, where $\beta \in \mathbb{N}$ is a *time buffer* that accounts for the incubation period, along with the delayed reporting of symptom scores and contacts. The time buffer is also known as the *(contact) tracing window* [75] and is similar to the *notification window* used by other contact tracing applications [55].
3. Risk transmission between contacts is incomplete. Thus, a risk score

decays exponentially along its transmission path in G according to the *transmission rate* $\alpha \in (0, 1)$. Figure 2.2 visualizes this decay, assuming a transmission rate of $\alpha = 0.8$ [33].

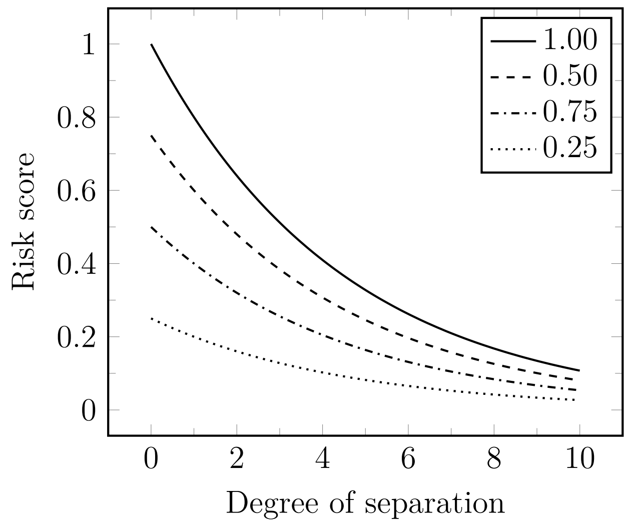

Figure 2.2: Exponential decay of risk scores.

To reiterate, a factor message $\lambda_{ij}^{(n)}$ is the maximum relevant risk score in the variable message $\mu_{ij}^{(n)}$ (or 0) that is scaled by the transmission rate α.[1]

Ayday, Yoo, and Halimi [3] assume that the contact set C may contain (1) multiple contacts between the same two individuals and (2) invalid contacts, or those lasting less than δ time. However, these assumptions introduce unnecessary complexity. Regarding assumption 1, suppose the i-th and j-th individual come into contact m times such that $t_k < t_\ell$ for $1 \leq k < \ell \leq m$. Let Λ_k be the set of relevant risk scores, according to the contact time t_k, where

$$\Lambda_k = \{\, \alpha s_t \mid s_t \in \mu_{ij}^{(n)}, t < t_k + \beta \,\}\,.$$

[1]This model of risk transmission and exposure risk updating is arguably unrealistic. However, it is beyond the scope of this work to propose alternative approaches.

Then $\Lambda_k \subseteq \Lambda_\ell$ if and only if $\max \Lambda_k \leq \max \Lambda_\ell$. Therefore, only the most recent contact time t_m is required to compute the factor message $\lambda_{ij}^{(n)}$. With respect to assumption 2, there are two possibilities.

1. If an individual has at least one valid contact, then their exposure score is computed over the subgraph induced in G by their contacts that define the neighborhood N_i of the variable vertex x_i.

2. If an individual has no valid contacts, then their exposure score is $\max S_i$ or 0, if all of their previously computed risk scores have expired.

In either case, a set C containing only valid contacts implies fewer factor vertices and edges in the factor graph G. Consequently, the complexity of Risk-Propagation is reduced by a constant factor since fewer messages must be computed.

2.1.3 Termination

To detect convergence, the normed difference between the current and previous exposure scores is compared to the threshold $\epsilon \in \mathbb{R}$. Note that $\mathbf{r}^{(n)}$ is the vector of exposure scores in the the n-th iteration such that $r_i^{(n)}$ is the i-th component of $\mathbf{r}^{(n)}$. The L_1 and L_∞ norms are sensible choices for detecting convergence. Ayday, Yoo, and Halimi [3] use the L_1 norm, which ensures that an individual's exposure score changed by at most ϵ after the penultimate iteration.

```
Risk-Propagation($S, C$)
 1: $(X, F, E) \leftarrow$ Factor-Graph($C$)
 2: $n \leftarrow 1$
 3: for each $x_i \in X$
 4:     $R_i^{(n-1)} \leftarrow$ top $k$ of $S_i$
 5:     $r_i^{(n-1)} \leftarrow \max R_i^{(n-1)}$
 6:     $r_i^{(n)} \leftarrow \infty$
 7: while $\|\mathbf{r}^{(n)} - \mathbf{r}^{(n-1)}\| > \epsilon$
 8:     for each $\{x_i, f_{ij}\} \in E$
 9:         $\mu_{ij}^{(n)} \leftarrow R_i^{(n-1)} \setminus \{\, \lambda_{ji}^{(\ell)} \mid \ell \in [1\,..\,n-1] \,\}$
10:         $\lambda_{ij}^{(n)} \leftarrow \max \{\, \alpha s_t \mid s_t \in \mu_{ij}^{(n)}, t < t_{ij} + \beta \,\}$
11:     for each $x_i \in X$
12:         $R_i^{(n)} \leftarrow$ top $k$ of $\{\, \lambda_{ji}^{(n)} \mid f_{ij} \in N_i \,\}$
13:         $r_i^{(n)} \leftarrow \max R_i^{(n)}$
14:     $n \leftarrow n + 1$
15: return $\mathbf{r}^{(n)}$
```

2.2 Asynchronous Risk Propagation

While Risk-Propagation offers proof of concept, it is not viable for real-world application. Risk-Propagation is an *offline algorithm* [18], because it requires the contact and health information of all individuals as input. As Ayday, Yoo, and Halimi [3] note, this centralization of personal data is not privacy-preserving. Risk-Propagation also introduces communication overhead and computational redundancy since most exposure scores are unlikely to change across frequent invocations. To mitigate this inefficiency, Ayday et al. [4] suggest running Risk-Propagation once or twice per day. Unfortunately, this cadence incurs substantial delay in updating individuals' exposure scores. In the midst of a pandemic, timely information is essential for individual and collective health.

To address the limitations of Risk-Propagation, Ayday, Yoo, and Halimi [3] propose decentralizing the factor graph such that the processing entity associated with the i-th individual maintains the state of the i-th variable vertex and the neighboring factor vertices. Applying one-mode projection onto the variable vertices [103], Figure 2.3 illustrates how each entity corresponds to a portion of the factor graph. More generally, Ayday, Yoo, and Halimi [3] envision risk propagation as a decentralized communication protocol for informing individuals about their infection risk. Such a message-passing protocol naturally aligns with the *actor model*, a local model of concurrent computing that defines computation as patterns of message passing amongst computational entities called *actors* [2, 17, 22, 38, 39, 40]. Since communication is asynchronous in the actor model, risk propagation defined in this way is called *asynchronous risk propagation.*

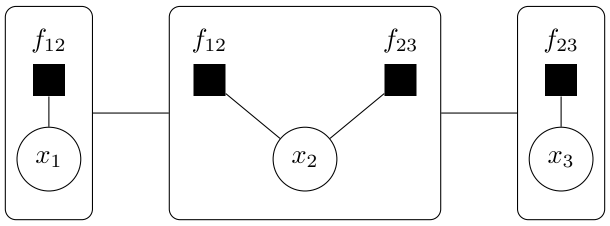

Figure 2.3: One-mode projection onto the variable vertices in Figure 2.1.

2.2.1 Actor Behavior

An actor a in the ShareTrace actor system represents an individual and is initialized by the Create-Actor operation [2] with the following attributes.

- $a.exposure$: the current exposure score of the individual. This attribute is either a symptom score, a risk score received from another actor, or the null risk score (see Null-Risk-Score).
- $a.scores$: a dynamic multiset of received risk scores. The key $s.key$ of a risk score s in $a.scores$ is the time it was computed $s.time$. An actor may receive multiple risk scores that were computed at the same time, so a multiset provides a complete history of relevant risk scores.
- $a.contacts$: a dynamic set where each element is a *proxy* [27] of another actor. That is, if the i-th individual was physically proximal with the j-th individual, then $a_i.contacts$ contains an element c such that $c.key$ is a name of the j-th actor and $c.time$ is their most recent time of contact.

```
Null-Risk-Score
1: $s.value \leftarrow 0$
2: $s.time \leftarrow 0$
3: **return** $s$
```

The interface of an actor is primarily defined by two types of messages: contacts and risk scores. Based on Section 2.1, let the *time to live* (TTL) of a message be the remaining time of its relevance. The reference time τ is assumed to be the current time.

```
CREATE-ACTOR
1: $a.exposure \leftarrow$ NULL-RISK-SCORE
2: $a.scores \leftarrow \emptyset$
3: $a.contacts \leftarrow \emptyset$
4: return $a$
```

```
RISK-SCORE-TTL($s$)
1: return $T_s - (\tau - s.time)$
```

```
CONTACT-TTL($c$)
1: return $T_c - (\tau - c.time)$
```

The HANDLE-RISK-SCORE operation defines how an actor behaves upon receiving a risk score. The UPDATE-EXPOSURE-SCORE operation is the online equivalent of updating an individual's exposure score in RISK-PROPAGATION. The MAX-RISK-SCORE operation returns the risk score with the maximum value. In the event of ties, the newer risk score is preferred. Expired risk scores are also deleted, which ensures the risk score returned is relevant and that *a.scores* does not grow without bound.

```
HANDLE-RISK-SCORE($a, s$)
1: if RISK-SCORE-TTL($s$) $> 0$
2:     $s.key \leftarrow s.time$
3:     INSERT($a.scores, s$)
4:     UPDATE-EXPOSURE-SCORE($a, s$)
5:     for each $c \in a.contacts$
6:         APPLY-RISK-SCORE($a, c, s$)
```

```
UPDATE-EXPOSURE-SCORE$(a, s)$
 1: if $a.exposure.value < s.value$
 2:     $a.exposure \leftarrow s$
 3: else if RISK-SCORE-TTL$(a.exposure) \leq 0$
 4:     $a.exposure \leftarrow$ MAX-RISK-SCORE$(a.scores, \infty)$
```

```
MAX-RISK-SCORE$(a, t)$
 1: $s^* \leftarrow$ NULL-RISK-SCORE
 2: for $s \in a.scores$
 3:     if RISK-SCORE-TTL$(s) \leq 0$
 4:         DELETE$(a.scores, s)$
 5:     else if $s.time < t$
 6:         if $s.value > s^*.value$
 7:             $s^* \leftarrow s$
 8:         else if $s.value = s^*.value$ and $s.time > s^*.time$
 9:             $s^* \leftarrow s$
10: return $s^*$
```

For the moment, assume that APPLY-RISK-SCORE is the online equivalent of computing a factor message (see Section 2.1). Line 2 indicates that a copy s' of the risk score s is updated using the transmission rate α. The SEND operation enables actor communication [2].

```
APPLY-RISK-SCORE$(a, c, s)$
 1: if $c.time + \beta > s.time$
 2:     $s'.value \leftarrow \alpha \cdot s.value$
 3:     SEND$(c.name, s')$
```

The problem with Apply-Risk-Score is that it causes risk scores to propagate *ad infinitum*. Unlike Risk-Propagation, a global convergence test is not available to terminate message passing, so it is necessary to define a local condition that determines if a risk score should be sent to another actor. Practically, “self-terminating” message-passing is necessary for asynchronous risk propagation to be scalable and cost-efficient. The intent of sending a risk score to an actor is to update its exposure score. According to Handle-Risk-Score, it is only necessary to send an actor risk scores with values that exceed its current exposure score. Thus, an actor can associate a *send threshold* with a contact such that the target actor only receives risk scores that exceed the threshold. To permit a trade-off between accuracy and efficiency, let the *send coefficient* $\gamma \in \mathbb{R}$ be a scaling factor that is applied to a risk score upon setting the send threshold.

```
Set-Send-Threshold($c, s$)
1: $s'.value \leftarrow \gamma \cdot s.value$
2: $c.threshold \leftarrow s'$
```

The Apply-Risk-Score operation that incorporates the concept of a send threshold is defined below. Assuming a finite number of actors, a positive send coefficient guarantees that a risk score has finite propagation.

```
Apply-Risk-Score($a, c, s$)
1: if $c.threshold.value < s.value$ and $c.time + \beta > s.time$
2:     $s'.value \leftarrow \alpha \cdot s.value$
3:     Set-Send-Threshold($c, s'$)
4:     Send($c.name, s'$)
```

The Set-Send-Threshold operation defines *how* the send threshold is updated, but not *when* it should be updated; the Update-Send-Threshold operation encapsulates the latter. The second predicate on Line 1 stems from the fact that the send threshold is a risk score and thus subject to expiry. The first predicate, however, is more subtle and will be revisited shortly. Consistent with Apply-Risk-Score, the risk score retrieved from *a.scores* is scaled by the transmission rate and set as the new send threshold.

```
Update-Send-Threshold($a, c$)
1: if $c.threshold.value > 0$ and Risk-Score-Ttl($c.threshold$) $\leq 0$
2:     $s \leftarrow$ Max-Risk-Score($a.scores, c.time + \beta$)
3:     $s'.value \leftarrow \alpha \cdot s.value$
4:     Set-Send-Threshold($c, s'$)
```

The send threshold should be current when evaluating Line 1 of Apply-Risk-Score above, so Update-Send-Threshold is invoked beforehand.

```
Apply-Risk-Score($a, c, s$)
1: Update-Send-Threshold($a, c$)
2: if $c.threshold.value < s.value$ and $c.time + \beta > s.time$
3:     $s'.value \leftarrow \alpha \cdot s.value$
4:     Set-Send-Threshold($c, s'$)
5:     Send($c.name, s'$)
```

Returning to the first predicate on Line 1 of Update-Send-Threshold, the send threshold has a value of 0 when it is first set or when there is no unexpired risk score in *a.scores* that was computed before the buffered contact time (Line 2). Suppose the first predicate is omitted from Line 1. Given that Update-Send-Threshold is the first statement in Apply-Risk-Score, it is possible that the send threshold is set prior to sending the target actor a relevant risk score. In the worst case, this prevents *all* risk scores from being sent to that actor. As a result, that individual would not receive risk scores that may impact their exposure score. Therefore, for correct message-passing behavior, the send threshold is updated only when its value is nonzero.

The aforementioned refinements to Apply-Risk-Score ensure message passing terminates and behaves correctly over time. To conclude the definition of Apply-Risk-Score, a message-passing optimization, called *sender-side aggregation* [61], is incorporated. Within a short period of time, an actor may receive several risk scores that are subsequently sent to multiple target actors. Rather than sending multiple risk scores, it would be more efficient to just send the final risk score. As a heuristic, Apply-Risk-Score can be modified so that a contact "buffers" a risk score that the target actor should receive.

```
Apply-Risk-Score($a, c, s$)
1: Update-Send-Threshold($a, c$)
2: if $c.threshold.value < s.value$ and $c.time + \beta > s.time$
3:     $s'.value \leftarrow \alpha \cdot s.value$
4:     Set-Send-Threshold($c, s'$)
5:     $c.buffered \leftarrow s'$
```

When the actor receives a periodic *flush timeout* message, all contacts are "flushed" by sending the buffered risk scores to the target actors. The Handle-Flush-Timeout operation also removes all expired contacts from *a.contacts*, which ensures *a.contacts* does not grow without bound.

```
HANDLE-FLUSH-TIMEOUT($a$)
1: for each $c \in a.contacts$
2:     if $c.buffered \neq$ NIL
3:         SEND($c.name, c.buffered$)
4:         $c.buffered \leftarrow$ NIL
5:     if CONTACT-TTL($c$) $\leq 0$
6:         DELETE($a.contacts, c$)
```

The HANDLE-CONTACT operation concludes the definition of actor behavior. Similar to HANDLE-RISK-SCORE, expired contacts are not processed. The MERGE-CONTACT operation adds the contact c to the set *a.contacts* if a contact with the same name does not already exist. Otherwise, the contact time of the existing contact is conditionally updated. For new contacts, a risk score from *a.scores* is applied to ensure that, if the actor receives no other risk score before the contact expires, the target actor is sent at least one relevant risk score.

```
HANDLE-CONTACT($a, c$)
1: if CONTACT-TTL($c$) $> 0$
2:     if MERGE-CONTACT($a.contacts, c$) $=$ NIL
3:         $s \leftarrow$ MAX-RISK-SCORE($a.scores, c.time + \beta$)
4:         APPLY-RISK-SCORE($a, c, s$)
```

```
MERGE-CONTACT($a.contacts, c$)
1: $c' \leftarrow$ SEARCH($a.contacts, c.name$)
2: if $c' =$ NIL
3:     $c.key \leftarrow c.name$
4:     $c.threshold \leftarrow$ NULL-RISK-SCORE
5:     INSERT($a.contacts, c$)
6: else if $c'.time < c.time$
7:     $c'.time \leftarrow c.time$
8: return $c'$
```

2.2.2 Message Reachability

The ShareTrace actor system is a *contact network*, a type of *temporal network* in which vertices represent individuals and edges indicate contact between them [42, 43]. The contact set defined in (2.2) is the *contact sequence* representation of a contact network [43]. Contact networks are typically used in epidemiological studies that aim to model and analyze the spreading dynamics of infection [19, 21, 51, 57, 70, 78, 105]. The ShareTrace actor network, however, models the spreading of infection *risk*. Holme and Saramäki [43] note that the transmission graph proposed by Riolo, Koopman, and Chick [78] "cannot handle edges where one node manages to not catch the disease." By framing the spreading phenomenon as continuous, rather than discrete, it is possible to model partial transmission.

A primitive of temporal reachability analysis 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 [64]. The following derivatives of a time-respecting path help quantify reachability in temporal networks [43].

- *influence set* $I(v)$: vertices that v can reach by a time-respecting path.
- *source set* $S(v)$: vertices that can reach v by a time-respecting path.
- *reachability ratio* $\rho(G)$: the average influence set cardinality in G.

Generally, a message-passing algorithm specifies constraints that determine when and what messages are sent between vertices. 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 broader definition of reachability that accounts for network topology *and* message-passing semantics [6].

Formally, let $\varphi(P, m)$ be the number of edges along the path P from vertex u to vertex v that satisfy the message-passing constraints for the message m,

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

where

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

Then vertex v is *message-reachable* from vertex u for the message m if there exists a path P such that $\varphi(P, m) = |P|$. The *message reachability* of a message m from vertex u is the length of the longest path from u to some message-reachable vertex v. Measures of temporal reachability can be extended to message reachability,

$$I(u,m) = \{\, v \in V \mid \exists P[\varphi(P,m) = |P|] \,\}$$

$$S(v,m) = \{\, u \in V \mid \exists P[\varphi(P,m) = |P|] \,\}$$

$$\rho(G,M) = \sum_{v \in V,\, m \in M} |I(v,m)| \cdot |V|^{-1}.$$

Asynchronous Risk Propagation

Let P be the set of contact edges along the path from actor u to actor v such that the actors are enumerated $1, \ldots, |P|$. Let $m = (s_u, t_u)$ be a symptom score of actor u. Let s_{ij} be the value of the risk score that is associated with the send threshold of the i-th actor for the j-th actor. Finally, let t_{ij} be the most recent contact time between the i-th and j-th individual. Then message reachability for asynchronous risk propagation is defined as

$$\varphi(P,m) = \sum_{(i,j) \in P} [\alpha^i s_u > \gamma \alpha s_{ij}] \cdot [t_u < t_{ij} + \beta], \tag{2.3}$$

where $[\cdot]$ is the Iverson bracket[2].

By relaxing the temporal constraint in (2.3), an upper bound on the reachability of a symptom score can be defined. Reversing the inequality of the first

[2]The *Iverson bracket* $[x]$ is the indicator function of the set values for which x is true.

term in (2.3) and solving for i,

$$\hat{\varphi}(P) \leq \begin{cases} 0 & \text{if } s_u = 0 \\ |P| & \text{if } s_v = 0 \\ \log_\alpha \dfrac{\gamma \alpha s_v}{s_u} & \text{otherwise,} \end{cases} \tag{2.4}$$

where $\gamma \alpha s_v$ is the send threshold of the actor that precedes actor v along the path P. Assuming a transmission rate of $\alpha = 0.8$ and a send coefficient of $\gamma = 1$, Figure 2.4 visualizes (2.4).

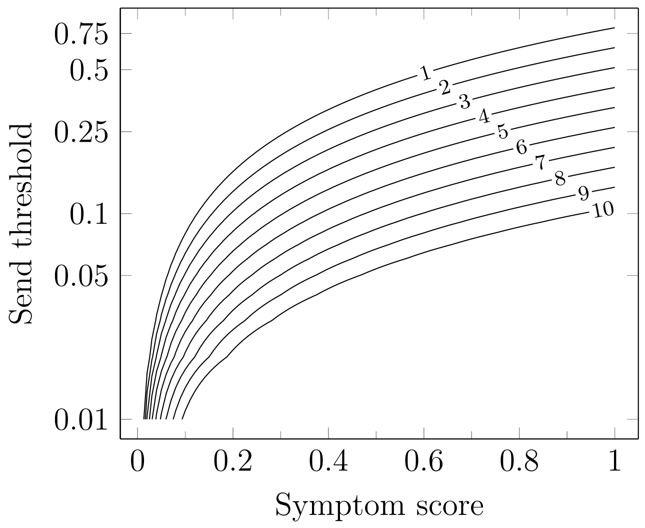

Figure 2.4: Message reachability for asynchronous risk propagation. Contour lines are shown for integral reachability values, indicating the maximum send threshold that permits sending the risk score.

2.3 System Design

With the algorithmic foundation of ShareTrace established, it is now possible discuss the aspects of system design. To a varying extent, prior work on ShareTrace [3, 4, Appendix A] has explored the practical considerations for deployment. This work, however, provides additional detail by contextualizing ShareTrace as an application of *mobile crowdsensing* (MCS), a "sensing paradigm that empowers ordinary citizens to contribute data sensed or generated from their mobile devices" that is aggregated "for crowd intelligence extraction and human-centric service delivery" [30]. To offer a clear comparison with other MCS applications, this section uses the classification criteria proposed by Capponi et al. [13]. Note, this section assumes the usage of asynchronous risk propagation that is described in Section 2.2.

2.3.1 Application Layer

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" [13]. ShareTrace follows proactive scheduling where the sensing task is to detect proximal interactions with other individuals. Naturally, the scheduling of this task is at the discretion of ShareTrace users.

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” [13]. The latter aligns with ShareTrace since each user is responsible for their own interactions.

Task Execution

With *single-task execution*, MCS applications assign one type of task to users, while *multi-task execution* assigns multiple types of tasks [13]. ShareTrace involves the single task of sensing proximal interactions to derive an individual’s infection risk.

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” [13]. ShareTrace follows volunteer-based recruitment, though incentive mechanisms would be an important consideration for widespread adoption [1, 69].

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” [13]. ShareTrace employs user-centric selection, because the purpose of the application is to passively sense in-person interactions and provide individuals with knowledge of their infection risk. However, ShareTrace does not require the presence of a central collector, such as a government health authority.

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” [13]. ShareTrace users could be contributors or consumers but would likelier be the latter.

2.3.2 Data Layer

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" [13].

ShareTrace relies on distributed—more specifically, decentralized—storage. Prior work [3, 4, Appendix A] suggests using Dataswyft Personal Data Accounts[3], which provide a data-oriented interface to self-sovereign identity (SSI) [74, 80]. However, personal data stores alone cannot provide individuals true controllership over their personal data and have historically not demonstrated successful adoption in the consumer market [66]. Regardless of whether message-passing between actors is decentralized, so long as actor computation is decentralized on mobile devices, using local storage is the simplistic approach.

To allow asynchronous message passing between actors in the ShareTrace actor system (i.e., to be delay-tolerant), the underlying messaging system must persist the messages sent between actors, at least until they expire. While further research on alternative decentralized technologies is needed [46, 95], the InterPlanetary FileSystem (IPFS)[4] [7, 82, 93], and the various higher-

[3]https://dataswyft.io
[4]https://ipfs.tech

level storage abstractions (e.g., OrbitDB[5] and web3.storage[6]) that have been developed with it, offers a promising solution. Additionally, investigating the feasibility of implementing secure publish-subscribe systems [20] with IPFS could provide robust security and privacy guarantees.

Data Format

Structured data is standardized and readily analyzable, while *unstructured data* requires significant processing before it can be used [13]. ShareTrace data, such as reported symptoms, biometric data, contact identifiers, and risk scores, is structured.

Data Dimensionality

Single-dimension data typically occurs when a single sensor is used, while *multi-dimensional data* arises with the use of multiple sensors. Prior work on ShareTrace assumes single-dimensional data: contact identifiers. Though, integrating biometric sensor data into the calculation of symptom scores would classify ShareTrace as using multi-dimensional data.

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" [13]. ShareTrace should aggregate contact

[5]`https://orbitdb.org`
[6]`https://web3.storage`

identifiers over time in order to determine which encounters were sustained (e.g., lasting at least 15 minutes). Moreover, if biometric sensors are also incorporated into the symptom score calculation, then signal processing would likely be required.

Data Analytics

Machine learning (ML) and data mining analytics "aim to infer information, identify patterns, or predict future trends" and are not real-time. On the contrary, *real-time analytics* consist of "examining collected data as soon as it is produced by the contributors" [13]. While the message passing that occurs during risk propagation may take place in near real-time, this is not a requirement. Thus, ShareTrace more closely aligns with the former category since it aims to predict an individual's infection risk.

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" [13]. ShareTrace applies predictive post-processing via risk propagation.

2.3.3 Communication Layer

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” [13]. Infrastructured technology is also referred to as the *infrastructured transmission paradigm* [59]. ShareTrace is delay-tolerant; thus, it can rely on WLAN infrastructure.

Infrastructureless Technology

Infrastructureless 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* [13]. Infrastructureless technology is also called the *opportunistic transmission paradigm* [59]. Because of its sufficient short-range accuracy and energy efficiency, ShareTrace uses Bluetooth Low Energy (BLE) to transmit and receive ephemeral identifiers.

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” [13]. ShareTrace is delay-tolerant, so it uses store-and-forward uploading.

Reporting Metholodgy

Individualized sensing is “when each user accomplishes the requested task individually and without interaction with other participants.” *Collaborative sens-*

ing is when “users communicate with each other, exchange data[,] and help themselves in accomplishing a task…Users are typically grouped and exchange data exploiting short-range communication technologies” [13]. The methodology of sensing is similar to the *sensing scale* which is typically dichotomized as *personal* [28, 54] (i.e., individualized) and *community* [28] or *group* [54] (i.e., collaborative). ShareTrace includes elements of both individualized and collaborative sensing. With respect to the former, each user may individually use biometric sensors to calculate their symptom score. At the same time, ShareTrace users collaboratively sense each other in order to estimate their exposure score. The latter aspect of ShareTrace is a requirement, so overall ShareTrace primarily applies collaborative sensing.

Reporting 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” [13]. ShareTrace requires synchronous timing, because contact sensing inherently requires synchronous communication between mobile devices.

2.3.4 Sensing Layer

Sensor Deployment

Dedicated deployment involves using “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” [13]. ShareTrace relies on non-dedicated deployment since it relies on BLE sensors that are ubiquitous 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 of energy” [13]. ShareTrace uses on-demand sensors (i.e., BLE), which run in the background and is not required for basic device functionality. ShareTrace may utilize always-on sensors, such as the accelerometer, to contextually activate the on-demand sensor and improve energy efficiency.

Data 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" [13]. In the event that ShareTrace only senses contact identifiers, it would be considered to use homogenous acquisition. If however, optional biometric sensors could be integrated into symptom score calculation, then ShareTrace would follow heterogenous acquisition.

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" [13]. ShareTrace continuously senses contact identifiers. However, in an effort to conserve energy, motion sensors could be used to selectively activate sensing when the user is carrying their mobile device.

Sensing Responsibility

When the *mobile device* is responsible, "devices or users take sampling decisions locally and independently from the central authority…It is often necessary to detect the context [of the] smartphones and wearable devices." When the *central collector* is responsible, they make "decisions about sensing and communicate them to the mobile devices" [13]. Given the human-centric and decentralized 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." With *opportunistic involvement*, "users…only declare their interest in joining a campaign and providing their sensors as a service…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" [13]. Earlier works on MCS refer to user involvement as the *sensing paradigm* [28, 54, 59]. ShareTrace applies opportunistic involvement since users do not need to take explicit action beyond installing the application on their mobile device. As stated previously, ShareTrace may also integrate on-demand sensors in order to contextually decide when to sense nearby users.

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	●
		Infrastructureless	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 2.1: Classifying ShareTrace as mobile crowdsensing (MCS) application. This table follows the four-layered architecture of MCS applications proposed by Capponi et al. [13]. Proposed design (●); plausible alternative (○).

Chapter 3

Evaluation

3.1 Reference Implementation

A reference implementation of Section 2.2 is available on GitHub[1]. Actors are implemented using the Akka toolkit[2], which offers high performance for large-scale actor systems. Experimental results indicate that the reference implementation can process contact networks with at least 100 thousand vertices and 10 million edges on a single machine, so it is suitable for small-scale and medium-scale experiments. In addition to using the Akka toolkit, several other optimizations are implemented.

- To reduce the size of event logs and result files, individual actor identifiers follow zero-based numbering and event records are serialized using the Ion format[3] with shortened field names.

[1]`https://github.com/cwru-xlab/sharetrace-akka`
[2]`https://doc.akka.io/docs/akka/2.8.5/typed/index.html`
[3]`https://amazon-ion.github.io/ion-docs`

- To reduce memory usage, FastUtil[4] data structures are used, including a specialized integer-based JGraphT[5] graph implementation [63]. Also, singletons [27], primitive data types, and reference equality are preferred where feasible and do not impact readability.
- To reduce runtime and increase throughput, logging is performed asynchronously with Logback[6] and the LMAX Disruptor[7].

Figure 3.1 shows the dependencies among the application components. Contextualizing this implementation with prior implementations of the driver-monitor-worker (DMW) framework (see Appendix A.3), `RiskPropagation` is the driver, `Monitor` is the monitor, and `User` is the worker. The key difference between this implementation and previous implementations of the DMW framework is that workers are stateful, which is necessary for decentralization.

$$\begin{tikzcd}
\texttt{Main} \arrow[r] & \texttt{Runner} \arrow[r] & \texttt{RiskPropagation} \arrow[r] & \texttt{Monitor} \arrow[r] & \texttt{User} \arrow[r] & \texttt{Contact}
\end{tikzcd}$$

Figure 3.1: Arrow diagram of the reference implementation.

Section 2.2 describes the behavior of a `User` and `Contact`. In order to evaluate `RiskPropagation`, each `User` logs the following types of `UserEvent`:

- `ContactEvent`: logged when the `User` receives an unexpired `ContactMessage`; contains the `User` identifier, the `Contact` identifier, and the contact time.

[4]`https://fastutil.di.unimi.it`
[5]`https://jgrapht.org`
[6]`https://logback.qos.ch/index.html`
[7]`https://lmax-exchange.github.io/disruptor`

- `ReceiveEvent`: logged when the `User` receives an unexpired `RiskScoreMessage`; contains the `User` identifier, the `Contact` identifier, and the `RiskScoreMessage`.
- `UpdateEvent`: logged when the `User` updates its exposure score; contains the `User` identifier, the previous `RiskScoreMessage`, and the current `RiskScoreMessage`.
- `LastEvent`: logged when the `User` receives a `PostStop` Akka signal[8] after the `Monitor` has stopped; contains the `User` identifier and the time of logging the final event, besides `LastEvent`; used to detect the end time of message passing.

For reachability analysis, a `RiskScoreMessage` contains the identifier of the `User` that propagated it and the identifier of the `User` that first sent it.

The `Monitor` is an actor that is responsible for transforming the `ContactNetwork` into a collection of `User`s and terminating when no `UpdateEvent` has occurred for a period of time. The `Monitor` logs several types of `LifecycleEvent`, ordered according to when they are logged during execution:

1. `RiskPropagationStart`
2. `CreateUsersStart`
3. `CreateUsersEnd`
4. `SendContactsStart`
5. `SendContactsEnd`
6. `SendRiskScoresStart`
7. `SendRiskScoresEnd`
8. `RiskPropagationEnd`

[8]`https://doc.akka.io/docs/akka/current/typed/actor-lifecycle.html#stopping-actors`

`RiskPropagation` logs execution properties, creates an Akka `ActorSystem` that creates a `Monitor` actor and sends it a `RunMessage`, and then waits until the `ActorSystem` terminates. Each execution of `RiskPropagation` is associated with a unique key that is included in each event record as mapped diagnostic context (MDC)[9].

The `Runner` specifies how `RiskPropagation` is created and invoked, usually through some combination of statically defined behavior and runtime configuration.

Finally, `Main` is the entry point into the application. It is responsible for parsing `Context`, `Parameters`, and `Runner` from configuration and invoking `Runner` with `Context` and `Parameters` as inputs. `Context` makes application-wide information accessible, such as the system time and user time[10], a pseudorandom number generator, `Runner` configuration, and loggers. `Parameters`, as the name suggests, is a collection of parameters that modify the behavior of the `Monitor`, `User`s, and `Contact`s.

To analyze the executions of `RiskPropagation` that are associated with the same configuration, `EventHandler`s process the execution properties and event logs. For each execution of `RiskPropagation`, one or more `EventHandler`s process the stream of event records and aggregate state about a particular aspect of the execution. Once the event stream has been processed, an `EventHandler` may perform a final aggregation or transformation of its state

[9]`https://logback.qos.ch/manual/mdc.html`

[10]System time is always the wall-clock time and is included in each logged event record. User time is configurable to either be the wall-clock time or fixed at the reference time. The latter ensures that no `RiskScoreMessage` or `ContactMessage` expires across executions of `RiskPropagation`.

and persist the result in a `Results` instance. The contents of the `Results` instance is then stored for further analysis. For example, each experiment in Section 3.2 was associated with multiple configurations. The results of those configurations were aggregated and flattened into a tabular dataset, which was subsequently analyzed and presented in Section 3.3.

The following `EventHandler`s were implemented for this work:

- `Reachability`: aggregates `ReceiveEvent`s that involve a distinct sender and receiver to determine the influence set cardinality, source set cardinality, and message reachability of each `User`.
- `Runtimes`: aggregates `LifecycleEvents` and `LastEvent`s to determine the runtime of creating `User`s, sending `ContactMessage`s, sending `Risk-ScoreMessage`s, message passing, and the overall execution of `Risk-Propagation`. Message-passing runtime is the duration between `Send-RiskScoresStart` and the latest `LastEvent`.
- `UserEventCounts`: aggregates `UserEvent`s to determine the frequency of each type for each `User`.
- `UserUpdates`: aggregates `UpdateEvent`s to determine the new exposure score of each `User` and the change in value.

3.2 Experimental Design

The following experiments were used to evaluate `RiskPropagation`.

1. How does the send coefficient affect accuracy and efficiency?

2. Do the distributions of risk scores and contact times affect runtime?

3. How does the size of the contact network affect runtime?

Experiment 1 was completed first to determine suitable parameter values for the remaining experiments. Experiment 2 was conducted before Experiment 3 to determine the necessity of evaluating all combinations of data distributions. That is, if the type of data distribution was found to have a statistically nonsignificant effect on the runtime of `RiskPropagation`, then the complexity of Experiment 3 could be meaningfully reduced. Experiment 3 was conducted to determine if a simple model could be used in practice to estimate the message-passing runtime of `RiskPropagation`. While Experiment 2 and Experiment 3 focus on benchmarking the reference implementation, more advanced simulation-based analysis of ShareTrace with COVI-AgentSim [31] is the subject of future work.

All experiments utilized the same random graphs: Barabasi-Albert graphs [5], Erdős-Rényi $G_{n,m}$ graphs [23], Watts-Strogatz graphs [101], and random regular graphs [48]. These graphs were selected because they are available in the JGraphT library and are parametric, either directly or indirectly, in the size and order of the network. The latter property allowed the effects of the topology to be isolated. While Barabasi-Albert graphs, Erdős-Rényi $G_{n,m}$ graphs, Watts-Strogatz graphs exhibit aspects of real-world contact networks [67], random regular graphs do not.

The following describes the parametrization of each type of contact network. Barabasi-Albert graphs are parametrized by the order n, the initial order n_0, and the increase in size m_0 upon each incremental increase in order.

The latter two parameters are determined by solving (3.1), where frac(x) is the fractional part of a real number x.

$$\begin{aligned} \underset{n_0,\, m_0}{\arg\min} \quad & \operatorname{frac}(m_0) \\ \text{subject to} \quad & n_0 \in [1\,..\,n-1], \\ & m_0 \in [1\,..\,n_0], \\ & m_0 = \frac{2m - n_0(n_0 - 1)}{2(n - n_0)} \end{aligned} \tag{3.1}$$

Erdős-Rényi graphs are parametrized by the order n and the size m. Random regular graphs are parametrized by the order n and, using the degree sum formula, the degree $d = \lfloor 2m/n \rfloor$. Table 3.1 specifies the default parameter values and seed for pseudorandom number generation. Table 3.2 specifies the experiment configurations. All experiments used fixed user time. The following sampling process was used to generate risk scores and contact times. Given the probability density function f_X and the cumulative distribution function F_X of a random variable X, sample a value $x \sim f_X$ and evaluate $c \cdot F_X(x)$ for some scalar $c \in \mathbb{R}$. Risk scores are composite data types, so risk score values and risk score times were sampled independently. Because risk scores are probabilities, $c = 1$ was used to scale the values. When sampling the times of risk scores and contacts, $c = T_s$ and $c = T_c$ were used, respectively.

Parameter	Default value
Transmission rate, α	0.8
Send coefficient, γ	1
Time buffer, β	2 days
Risk score expiry, T_s	14 days
Contact expiry, T_c	14 days
Flush timeout	3 seconds
Idle timeout	1 minute
Seed	12 345

Table 3.1: Default parameter values for experiments.

Aspect	Experiment 1	Experiment 2	Experiment 3
Network parameters	$n = 5 \cdot 10^3$ $m = 5 \cdot 10^4$ $n_0 = 21$ $m_0 = 10$ $d = 20$ $k = 20$	$n = 5 \cdot 10^3$ $m = 5 \cdot 10^4$ $n_0 = 21$ $m_0 = 10$ $d = 20$ $k = 20$	$n \in \{ 10^4 x \mid x \in [1\,..\,10] \}$ $\times$ $m \in \{ 10^6 x \mid x \in [1\,..\,10] \}$ Tables 3.3 to 3.6 for n_0, m_0, d, k
Parameters	$\gamma \in \{ 10^{-1} x \mid x \in [8\,..\,20] \}$ Table 3.1 for remaining	Table 3.1	Table 3.1
Distributions	$\{\text{Uniform}, \text{Standard normal}\}^3$	$\{\text{Uniform}, \text{Standard normal}\}^3$	Uniform
Repetitions	5	1 burn-in + 5	1 burn-in + 5
Networks	160 (40 per type) per parameter	160 (40 per type)	2000 (500 per type)

Table 3.2: Experiment configurations. The notation X^k is used to denote the k-ary Cartesian power of the set X. A "burn-in" repetition was used for Experiment 2 and Experiment 3 to avoid measuring the impact of Java class loading.

	Size m									
Order n	$1 \cdot 10^6$	$2 \cdot 10^6$	$3 \cdot 10^6$	$4 \cdot 10^6$	$5 \cdot 10^6$	$6 \cdot 10^6$	$7 \cdot 10^6$	$8 \cdot 10^6$	$9 \cdot 10^6$	$10 \cdot 10^6$
1	201	401	601	801	1001	1201	1401	1601	1801	2001
2	101	201	301	401	501	601	701	801	901	1001
3	882	1161	201	1023	597	401	1042	2036	601	4000
4	51	101	151	201	251	301	351	401	451	501
5	41	81	121	161	201	241	281	321	361	401
6	836	829	101	357	2270	201	3443	2103	301	3125
7	215	733	1471	606	2439	3225	201	2615	3116	1875
8	643	51	1196	101	1623	151	2853	201	1206	251
9	626	491	1880	1406	1913	617	2747	1416	201	2411
10	21	41	61	81	101	121	141	161	181	201

Table 3.3: Barabasi-Albert n_0 values for Experiment 3.

	Size m									
Order n	$1 \cdot 10^6$	$2 \cdot 10^6$	$3 \cdot 10^6$	$4 \cdot 10^6$	$5 \cdot 10^6$	$6 \cdot 10^6$	$7 \cdot 10^6$	$8 \cdot 10^6$	$9 \cdot 10^6$	$10 \cdot 10^6$
1	100	200	300	400	500	600	700	800	900	1000
2	50	100	150	200	250	300	350	400	450	500
3	21	46	100	120	164	200	223	212	300	77
4	25	50	75	100	125	150	175	200	225	250
5	20	40	60	80	100	120	140	160	180	200
6	11	28	50	66	42	100	19	100	150	90
7	14	25	28	55	30	12	100	68	62	121
8	10	25	29	50	47	75	38	100	105	125
9	9	21	14	34	36	65	37	79	100	81
10	10	20	30	40	50	60	70	80	90	100

Table 3.4: Barabasi-Albert m_0 values for Experiment 3.

	Size m									
Order n	$1 \cdot 10^6$	$2 \cdot 10^6$	$3 \cdot 10^6$	$4 \cdot 10^6$	$5 \cdot 10^6$	$6 \cdot 10^6$	$7 \cdot 10^6$	$8 \cdot 10^6$	$9 \cdot 10^6$	$10 \cdot 10^6$
1	200	400	600	800	1000	1200	1400	1600	1800	2000
2	100	200	300	400	500	600	700	800	900	1000
3	66	133	200	266	333	400	466	533	600	666
4	50	100	150	200	250	300	350	400	450	500
5	40	80	120	160	200	240	280	320	360	400
6	33	66	100	133	166	200	233	266	300	333
7	28	57	85	114	142	171	200	228	257	285
8	25	50	75	100	125	150	175	200	225	250
9	22	44	66	88	111	133	155	177	200	222
10	20	40	60	80	100	120	140	160	180	200

Table 3.5: Random regular d values for Experiment 3.

	Size m									
Order n	$1 \cdot 10^6$	$2 \cdot 10^6$	$3 \cdot 10^6$	$4 \cdot 10^6$	$5 \cdot 10^6$	$6 \cdot 10^6$	$7 \cdot 10^6$	$8 \cdot 10^6$	$9 \cdot 10^6$	$10 \cdot 10^6$
$1 \cdot 10^4$	200	400	600	800	1000	1200	1400	1600	1800	2000
$2 \cdot 10^4$	100	200	300	400	500	600	700	800	900	1000
$3 \cdot 10^4$	66	134	200	266	334	400	466	534	600	666
$4 \cdot 10^4$	50	100	150	200	250	300	350	400	450	500
$5 \cdot 10^4$	40	80	120	160	200	240	280	320	360	400
$6 \cdot 10^4$	34	66	100	134	166	200	234	266	300	334
$7 \cdot 10^4$	28	58	86	114	142	172	200	228	258	286
$8 \cdot 10^4$	26	50	76	100	126	150	176	200	226	250
$9 \cdot 10^4$	22	44	66	88	112	134	156	178	200	222
$10 \cdot 10^4$	20	40	60	80	100	120	140	160	180	200

Table 3.6: Watts-Strogatz k values for Experiment 3.

3.3 Results

Several Python libraries were utilized in the writing of this section. Data visualizations were created with seaborn [100] and Matplotlib [44]. Tabular datasets were created and analyzed with Polars [98]. Pingouin [97] was utilized in Section 3.3.1 to compute correlation coefficients. SciPy [99] was used in Section 3.3.2 to perform hypothesis testing. scikit-learn [71] was used to train and evaluate the quantile regression models in Section 3.3.3. Lastly, NumPy [34] was used to support statistical calculation and data visualization.

3.3.1 Experiment 1

As indicated in Table 3.2, each repetition was associated with a set of send coefficients, a contact network, and a set of symptom scores. For each repetition, the accuracy of a send coefficient γ was defined over D_i, the set of *nonzero*[11] differences between the i-th individual's exposure score and symptom score,

$$f_i(\gamma) = 1 - \max D_i + D_i(\gamma),$$

where $D_i(\gamma)$ is the difference associated with γ. Figure 3.2 and Table 3.7 confirm that a send coefficient of $\gamma = 1$ produces perfect accuracy. For $\gamma > 1$, accuracy degrades for a relatively small percentage of the population. Intuitively, as the send coefficient increases, it becomes likelier for actors to not

[11]It is unclear whether an individual's exposure score was never updated due to the send coefficient or because the actor did not receive any risk scores that were higher than the individual's symptom score. If an individual's exposure score was never updated for any send coefficient, their differences were not considered for evaluation.

propagate risk scores that would otherwise update the exposure score of another actor in the network.

As noted in Table 3.7, it is expected that $\gamma < 1$ also provides perfect accuracy, given sufficient time for message passing to complete. Because the executions associated with $\gamma < 1$ were suspected to be incomplete, the associated data was omitted from the remainder of this analysis.

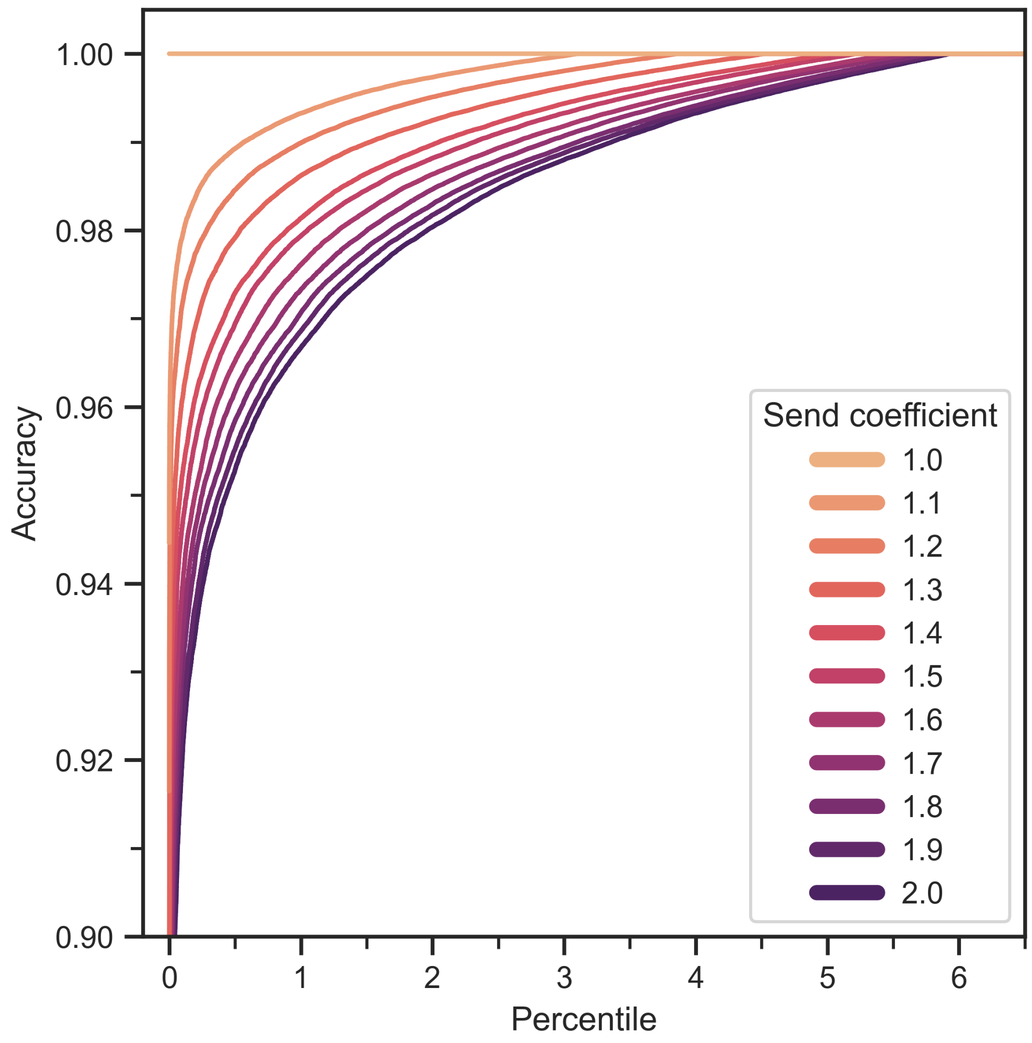

Figure 3.2: Cumulative accuracy distributions.

Figure 3.2 and Table 3.7 effectively describe the cumulative accuracy distributions of a population consisting of many disconnected contact networks. While that view can determine the accuracy `RiskPropagation` provides to a given percentile of the population, it does not quantify the extent to which a given send coefficient achieves perfect accuracy within each contact network.

Let γ_i^* be the maximum send coefficient for the i-th individual of a repetition that provides maximal accuracy,

$$\gamma_i^* = \max\left(\arg\max_{\gamma} f_i(\gamma)\right).$$

Figure 3.3 and Table 3.8 describe the proportion of individuals of each repetition that had a nonzero difference between their exposure score and symptom score, for which a given send coefficient was optimal. In other words, Figure 3.3 and Table 3.8 describe the empirical upper bound distribution of network-wide accuracy for each send coefficient.

With respect to efficiency, Figure 3.4 and Table 3.9 indicate that even modest increases above $\gamma = 1$ can reduce the total number of messages received by 10–20 % in a contact network. Larger increases above $\gamma = 1$ can reduce the total number of messages received by up to 30 %; however, accuracy should also be considered. Figure 3.5 indicates that message reachability is distributed consistently across network types. Though the proportion of individuals with a message reachability of 0 is inversely correlated with the send coefficient, the remainder of the cumulative distribution is nearly identical across send coefficients. Table 3.10 describes several percentiles of the message reachability

Send coefficient	P_0	$P_{0.01}$	$P_{0.1}$	P_1	P_2	P_3	P_4	P_5	P_6	P_{21}
0.8	0.724	0.802	0.844	0.871	0.889	0.903	0.914	0.924	0.933	1.000
0.9	0.903	0.926	0.937	0.968	0.986	0.993	0.997	0.999	1.000	1.000
1.0	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1.1	0.945	0.965	0.979	0.993	0.997	1.000	1.000	1.000	1.000	1.000
1.2	0.916	0.952	0.972	0.990	0.995	0.998	1.000	1.000	1.000	1.000
1.3	0.883	0.937	0.961	0.986	0.992	0.996	0.999	1.000	1.000	1.000
1.4	0.882	0.922	0.952	0.981	0.990	0.994	0.997	1.000	1.000	1.000
1.5	0.860	0.911	0.947	0.979	0.988	0.993	0.997	0.999	1.000	1.000
1.6	0.855	0.900	0.940	0.976	0.986	0.992	0.996	0.999	1.000	1.000
1.7	0.848	0.891	0.934	0.973	0.985	0.991	0.995	0.998	1.000	1.000
1.8	0.840	0.884	0.930	0.971	0.983	0.989	0.994	0.998	1.000	1.000
1.9	0.826	0.879	0.924	0.969	0.982	0.989	0.994	0.997	1.000	1.000
2.0	0.817	0.871	0.920	0.967	0.980	0.988	0.993	0.997	1.000	1.000

Table 3.7: Accuracy percentiles. The idle timeout (see Table 3.1) was suspected to be insufficient for the executions associated with a send coefficient $\gamma < 1$ to complete, but it is expected that $\gamma \leq 1$ permit perfect accuracy.

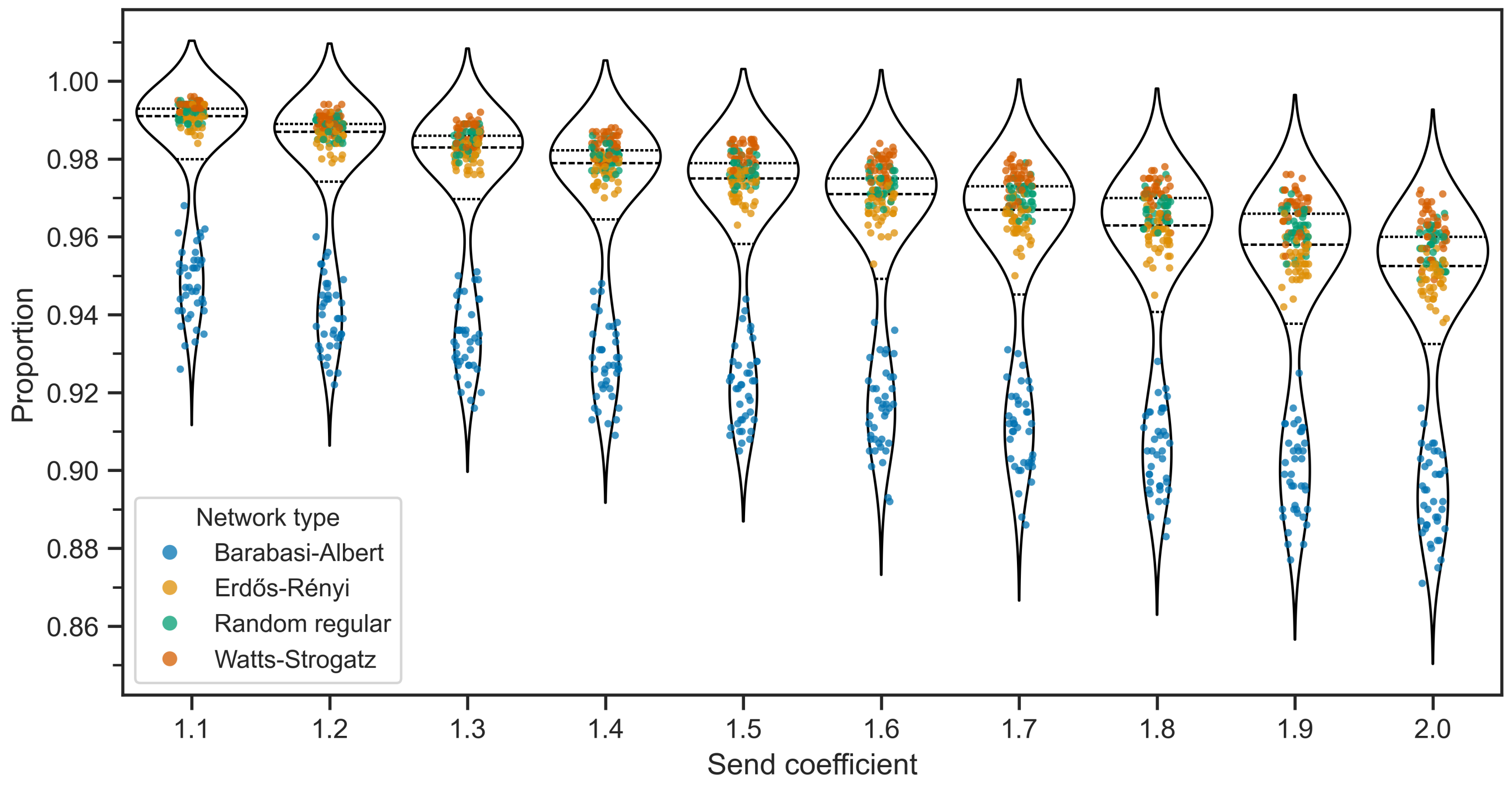

Figure 3.3: Send coefficient optimality distributions. The dashed line inside each violin marks the median. The upper and lower dotted lines inside each violin mark the upper and lower quartiles, respectively.

Send coefficient	P_0	P_{10}	P_{25}	P_{50}	P_{75}	P_{90}	P_{100}
1.0	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1.1	0.926	0.946	0.976	0.991	0.993	0.994	0.996
1.2	0.922	0.936	0.970	0.987	0.989	0.991	0.994
1.3	0.916	0.931	0.964	0.983	0.986	0.988	0.992
1.4	0.909	0.926	0.959	0.979	0.982	0.986	0.988
1.5	0.905	0.919	0.954	0.975	0.979	0.983	0.985
1.6	0.892	0.913	0.946	0.971	0.975	0.980	0.984
1.7	0.886	0.909	0.941	0.967	0.973	0.976	0.981
1.8	0.883	0.900	0.937	0.963	0.970	0.973	0.978
1.9	0.877	0.896	0.934	0.958	0.966	0.969	0.976
2.0	0.871	0.889	0.927	0.952	0.960	0.964	0.972

Table 3.8: Send coefficient optimality percentiles.

cumulative distribution, aggregating across send coefficients.

Send coefficient	P_0	P_{10}	P_{25}	P_{50}	P_{75}	P_{90}	P_{100}
1.0	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1.1	0.825	0.856	0.872	0.883	0.893	0.902	0.933
1.2	0.790	0.815	0.826	0.836	0.846	0.851	0.875
1.3	0.768	0.784	0.795	0.803	0.810	0.814	0.835
1.4	0.734	0.761	0.771	0.779	0.789	0.797	0.808
1.5	0.721	0.748	0.757	0.765	0.773	0.781	0.793
1.6	0.713	0.735	0.745	0.755	0.763	0.771	0.788
1.7	0.711	0.728	0.737	0.746	0.755	0.763	0.777
1.8	0.703	0.718	0.727	0.740	0.749	0.757	0.767
1.9	0.695	0.715	0.722	0.736	0.745	0.752	0.762
2.0	0.693	0.707	0.716	0.732	0.740	0.749	0.762

Table 3.9: Relative messages received percentiles.

A brief exploratory analysis is provided to conclude this section. Table 3.11 and Figure 3.6 and describe the degree distributions of the contact networks that were used for this experiment. While other degree distributions are centered around 20, the Barabasi-Albert distribution is characterized by a power

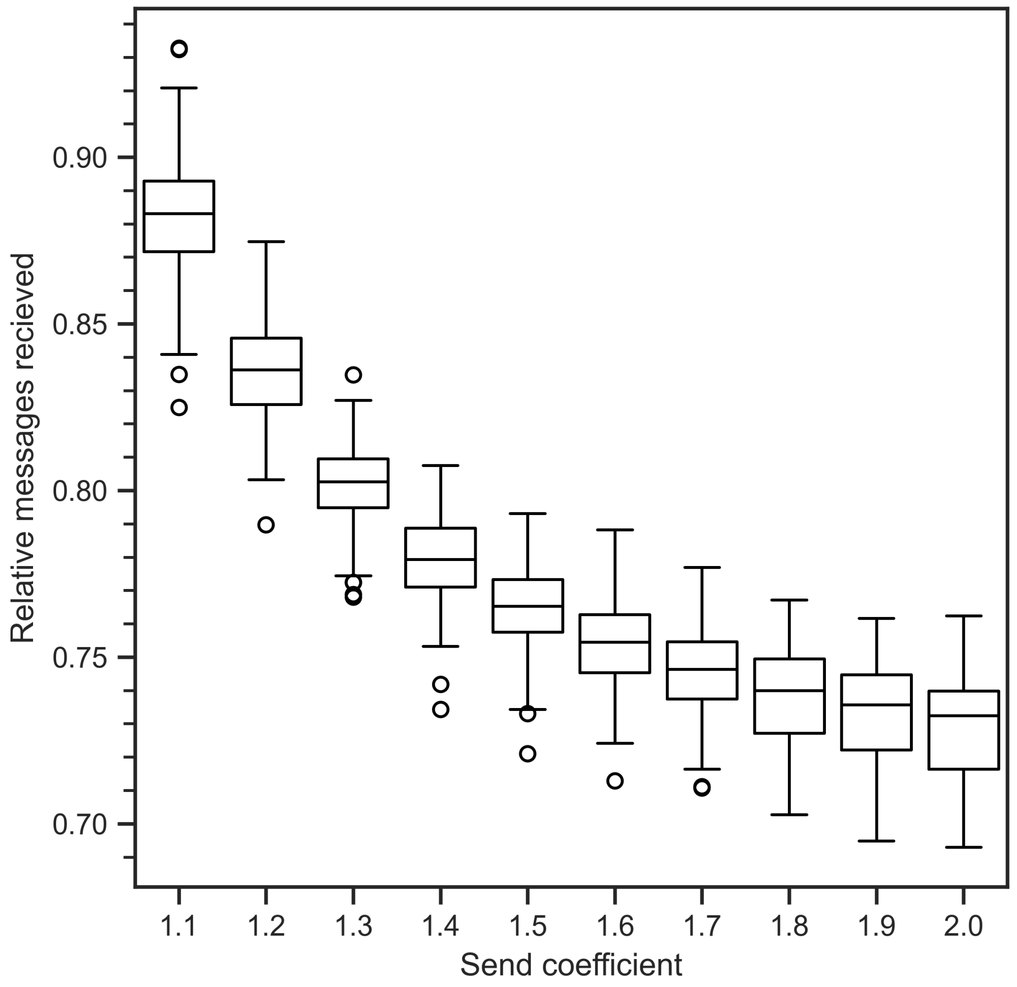

Figure 3.4: Message-passing efficiency. The send coefficient $\gamma = 1$ was used as a baseline for message-passing efficiency since it was found to be the maximum send coefficient that achieves perfect accuracy.

P_{25}	P_{50}	P_{75}	P_{90}	P_{95}	P_{99}	$P_{99.9}$	$P_{99.99}$	P_{100}
0	1	2	2	3	4	5	7	9

Table 3.10: Message reachability percentiles.

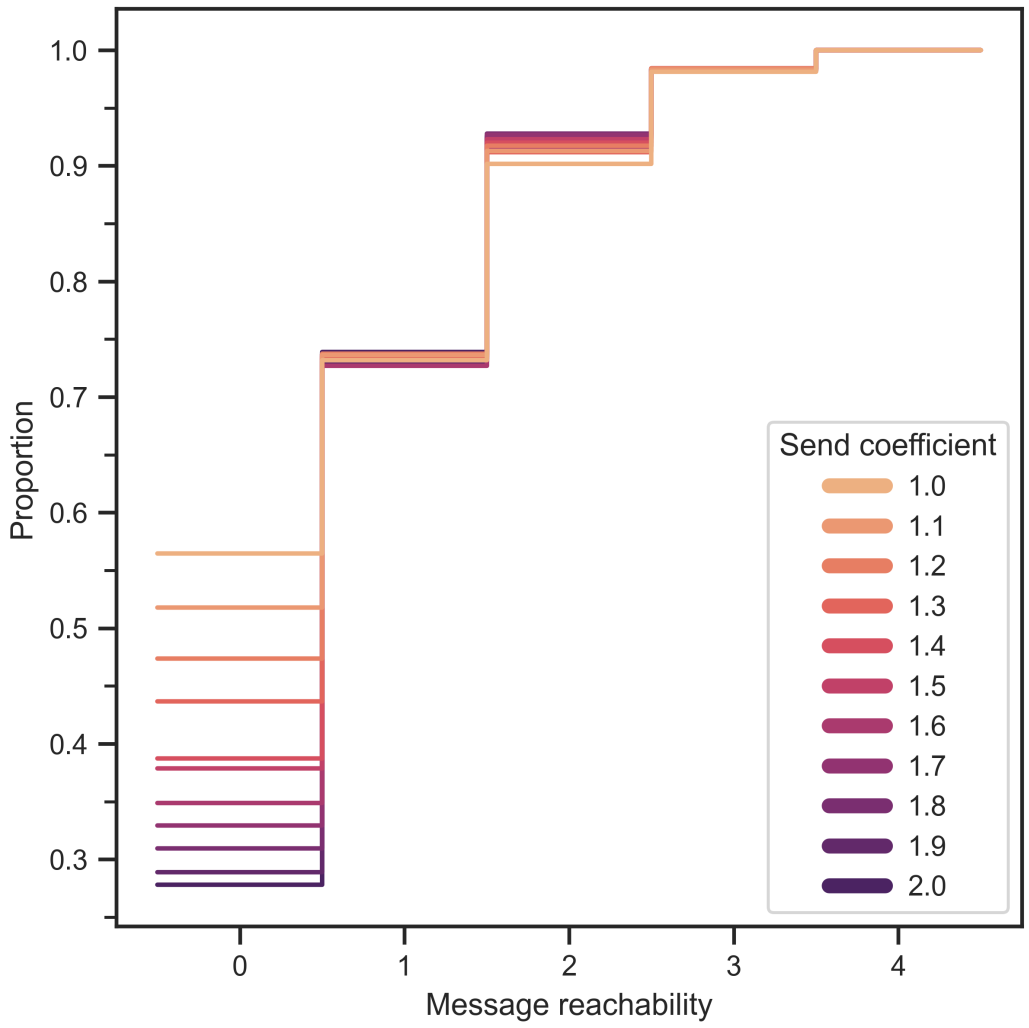

Figure 3.5: Message reachability cumulative distributions.

law that arises from the mechanism of preferential attachment. Figure 3.7 is the correlation matrix of the dataset attributes.

Network type	P_0	P_{10}	P_{25}	P_{50}	P_{75}	P_{90}	P_{100}
Barabasi-Albert	10	10	11	14	20	32	412
Erdős-Rényi	5	14	17	20	23	26	38
Random regular	20	20	20	20	20	20	20
Watts-Strogatz	14	18	19	20	21	22	29

Table 3.11: Contact network degree percentiles.

3.3.2 Experiment 2

A one-way analysis of variance (ANOVA) was considered for determining if the mean message-passing runtimes associated with different data distributions are statistically different. ANOVA assumes the observations are independently sampled, normally distributed, and homoscedastic within groups. The message-passing runtimes were indeed independently sampled. To test the latter two assumptions, the Shapiro-Wilk test [81] and the Fligner-Killeen test [26] were applied, respectively. While the runtimes were found to be homoscedastic ($p > 0.01$), they were not normally distributed ($p < 0.01$). As such, the Kruskal-Wallis test was applied instead of a one-way ANOVA. The test indicated that there is insufficient evidence to reject the null hypothesis that the median message-passing runtimes are the same across data distributions. See Table 3.12 for the test statistics and associated p-values. Note, this experiment assumes this statistical nonsignificance holds for other data distributions and network topologies.

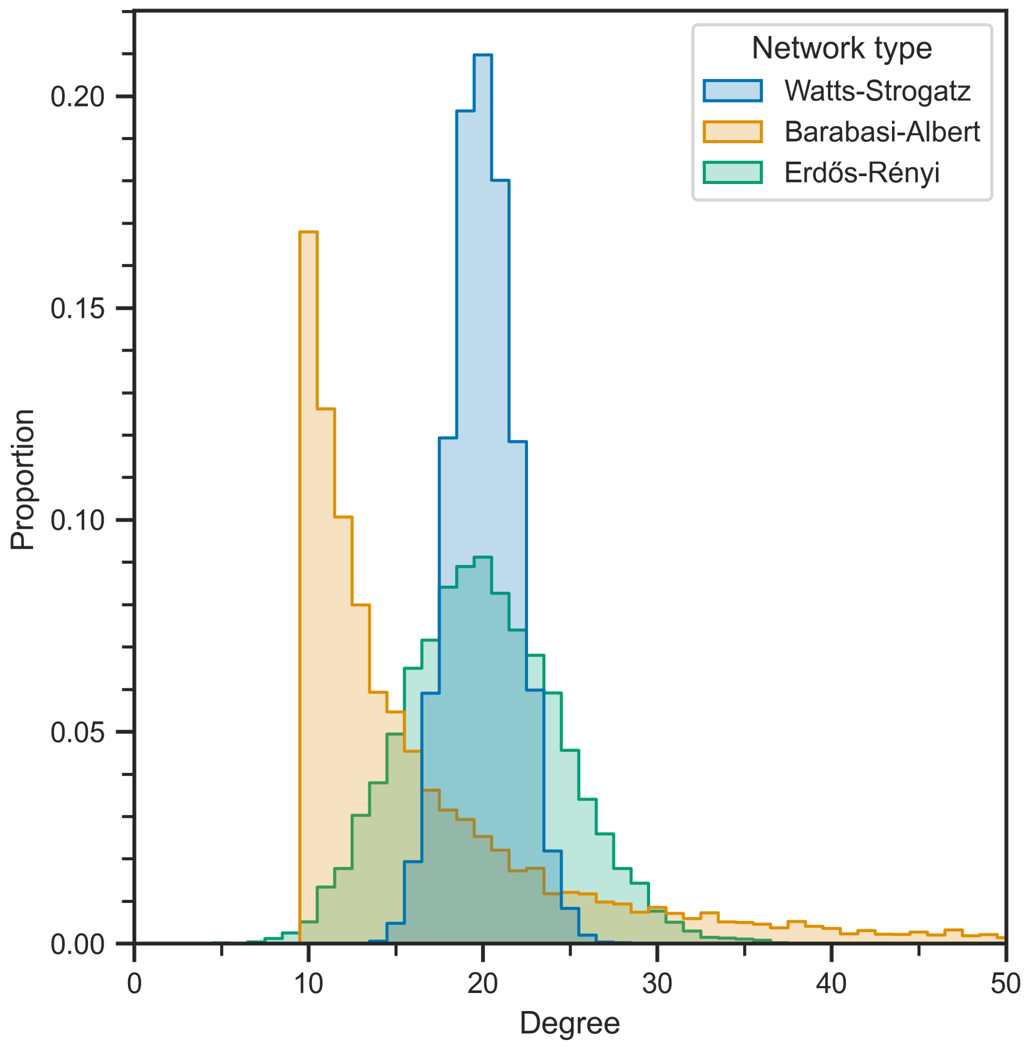

Figure 3.6: Contact network degree distributions. All vertices in random regular contact networks had a degree of 20, so the distribution was omitted to provide more visual space for the distributions of other contact networks.

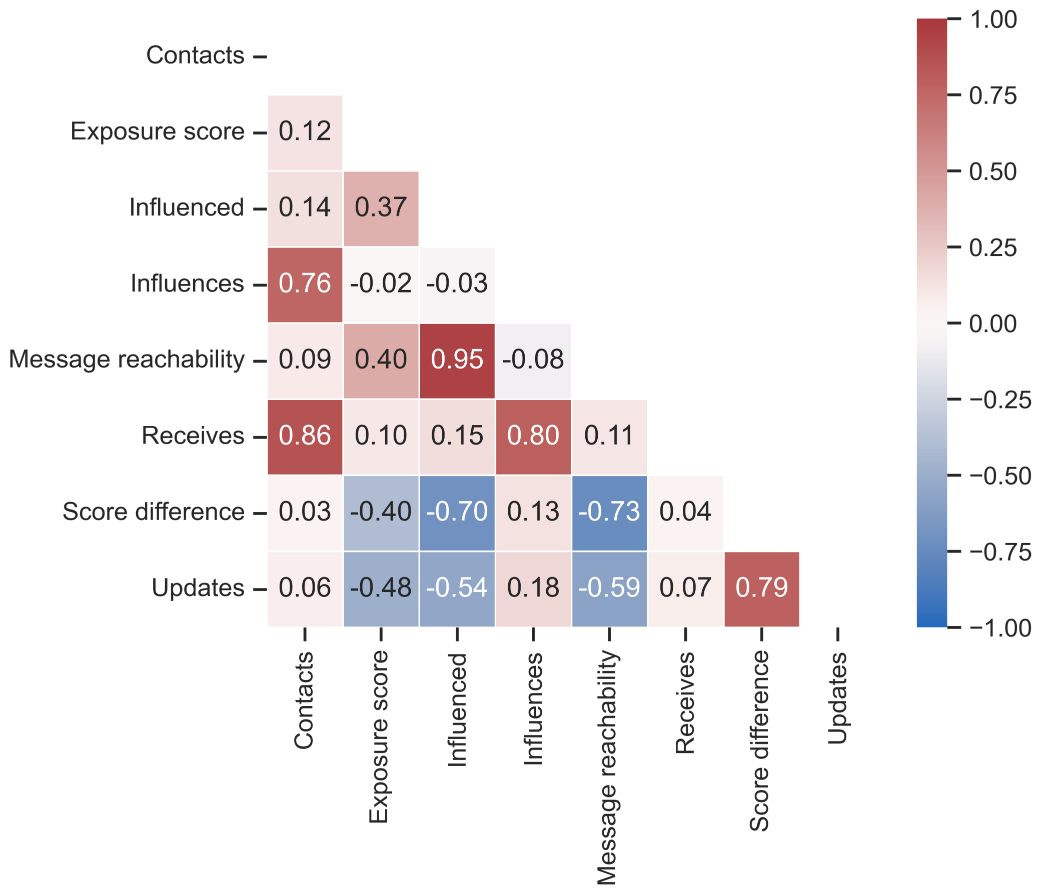

Figure 3.7: Correlation matrix of Experiment 1 dataset attributes. Each cell is the Spearman rank partial correlation coefficient [88], controlling for the effect of the send coefficient. All coefficients are significant ($p < 0.01$), adjusting for multiple comparisons via the Holm–Bonferroni method [41].

	Shapiro-Wilk test		Fligner-Killeen test		Kruskal-Wallis test	
Network type	Statistic	p-value	Statistic	p-value	Statistic	p-value
All	0.672	< 0.01	2.26	0.944	7.03	0.426
Barabasi-Albert	0.646	< 0.01	4.54	0.716	5.24	0.631
Erdős-Rényi	0.472	< 0.01	4.87	0.676	3.23	0.863
Random regular	0.583	< 0.01	3.90	0.791	3.02	0.883
Watts-Strogatz	0.437	< 0.01	2.47	0.929	5.97	0.543

Table 3.12: Hypothesis tests for message-passing runtime.

3.3.3 Experiment 3

Figure 3.8 indicates a linear relationship exists between the message-passing runtime and the size of the contact network. Given the runtime variance is heteroscedastic, quantile regression [49] was applied to determine the linear models that describe the $q = 0.05, 0.50, 0.95$ conditional quantiles of the message-passing runtime,

$$f_q(x) = \beta_0 + \beta_1 x. \tag{3.2}$$

The pinball loss function [90] with L_1 regularization,

$$L(\mathbf{y}, \hat{\mathbf{y}}) = \rho(\mathbf{y}, \hat{\mathbf{y}}) + \lambda \|\hat{\boldsymbol{\beta}}\|_1,$$

was utilized for parameter estimation, where

$$\rho(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n} \sum_{i=1}^{n} \begin{cases} q(y_i - \hat{y}_i) & \text{if } y_i \geq \hat{y}_i \\ (1-q)(\hat{y}_i - y_i) & \text{otherwise,} \end{cases}$$

and $y_i, \hat{y}_i$ are the true and predicted message-passing runtimes associated with the network size x_i. Nested 5-fold cross validation [14] was applied for model selection (inner loop) and model assessment (outer loop) [35, p. 222]. As part of model selection, grid search was applied over the set $\{\, 10^{-x} \mid x \in [1\,..\,5] \,\}$ to find the optimal regularization parameter λ. In addition to the mean pinball loss, the fraction of explained pinball deviance [36, 50] was calculated to

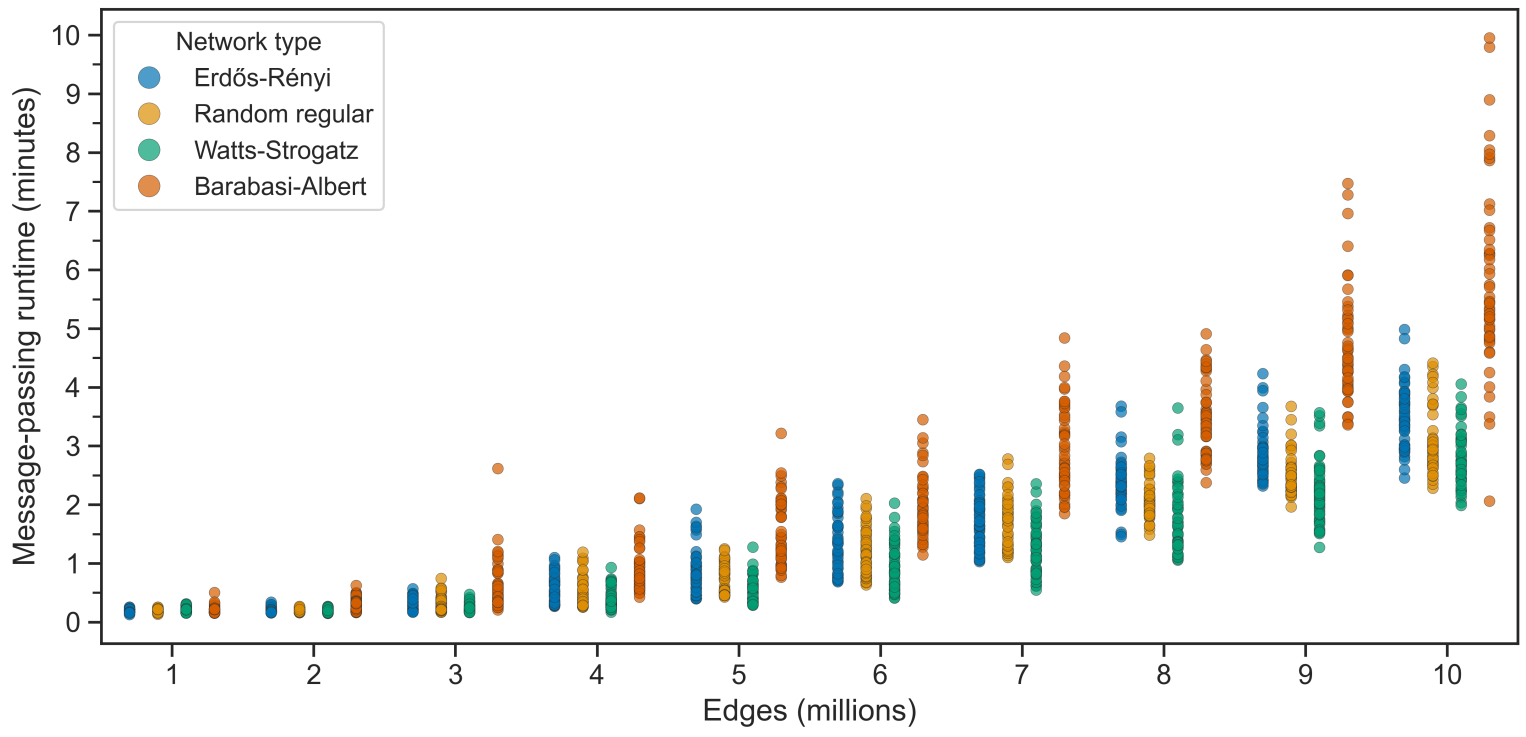

Figure 3.8: Message-passing runtimes.

provide a normalized measure for goodness of fit,

$$D^2(\mathbf{y}, \hat{\mathbf{y}}) = 1 - \frac{\rho(\mathbf{y}, \hat{\mathbf{y}})}{\rho(\mathbf{y}, \hat{\mathbf{y}}_0)},$$

where $\hat{\mathbf{y}}_0 = Q(\mathbf{y}, q) \cdot \mathbf{1}$ is the null model for the the q-th quantile.

Table 3.13 specifies the optimal models and evaluation metrics. Figure 3.9 overlays the observed message-passing runtimes with the predicted runtime quantiles.

				ρ		D^2	
q	β_0 (s)	β_1 (edges / s)	λ	Mean	SE	Mean	SE
0.05	−34.1	$1.25 \cdot 10^{-5}$	0.1	−2.94	0.05	0.255	0.011
0.50	−29.7	$1.99 \cdot 10^{-5}$	0.1	−15.6	0.4	0.520	0.009
0.95	−21.7	$3.48 \cdot 10^{-5}$	0.1	−5.91	0.30	0.496	0.018

Table 3.13: Models of message-passing runtime. For the q-th quantile, each row specifies the model parameters β_0, β_1 of (3.2) and the L_1 regularization parameter λ that maximized the fraction of explained pinball deviance D^2 on the test set during model assessment. The mean and standard error (SE) of the pinball loss ρ and D^2 are also provided from model assessment.

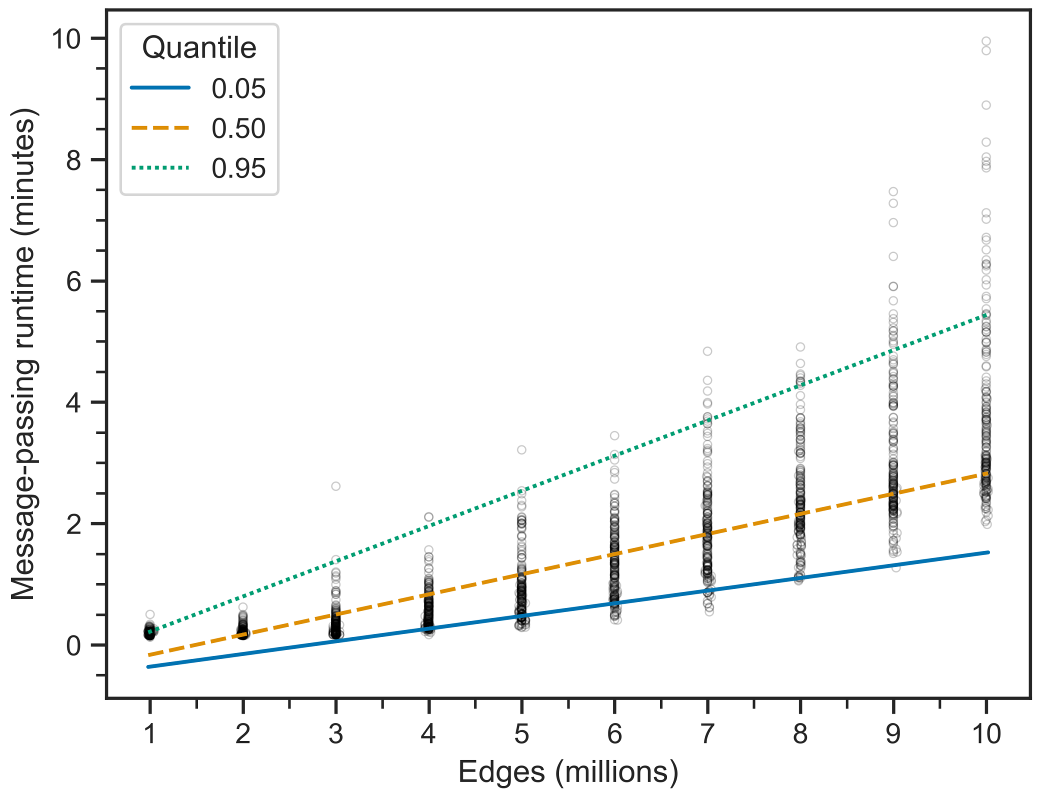

Figure 3.9: Message-passing runtimes with regression lines. The equation and parameters of each quantile regression line are specified in (3.2) and Table 3.13, respectively.

Chapter 4

Conclusion

This work provided a decentralized design of risk propagation, the message-passing algorithm that powers the ShareTrace contact tracing application. Message reachability was introduced as a means of measuring the dynamics of message passing on temporal networks, such as contact networks. On a practical note, ShareTrace was contextualized as a mobile crowdsensing (MCS) application using the four-layered architecture developed by Capponi et al. [13]. A reference implementation of asynchronous risk propagation was implemented and used to find the send coefficient that optimizes for accuracy and communication efficiency. Additionally, the scalability of the reference implementation was assessed. To ensure a fair evaluation, several types of contact network topologies and data distributions were utilized.

The following are subject to future work:

- Incorporate differential privacy techniques that are specifically designed for DCT applications, like ShareTrace, which utilize risk scores [79].

- Extend the calculation of risk scores to account for the transmission dynamics of the disease [24, 25].
- Formally define the security and privacy characteristics of ShareTrace, using the framework proposed by Kuhn, Beck, and Strufe [53] to characterize the latter.
- Integrate concepts from related approaches to DCT [15, 16, 32, 77].
- Explore the utility and feasibility of integrating decentralized technologies [7, 46, 82, 93, 95] and SSI [74, 80] into the design of ShareTrace.
- Conduct a simulation-based analysis of asynchronous risk propagation with COVI-AgentSim [31].

In May 2023, the World Health Organization (WHO) declared that COVID-19 is no longer a "global health emergency" [102]. However, as is evident throughout history, the risk posed by emerging pathogens persists [72, 89]. Thus, research on effective approaches, such as contact tracing, to preventing and mitigating future outbreaks remains critically important.

Appendix A

Prior Designs and Implementations

Before working on ShareTrace, I did not have experience developing distributed algorithms. The approach proposed in Chapter 2 is my *fifth* attempt at defining a performant implementation of risk propagation that is also decentralized and online. The prior four attempts offered valuable learnings that guided me toward the proposed approach; however, only the latter supports truly decentralized, privacy-preserving contact tracing. To document my efforts in developing this thesis, prior designs and implementations are provided in this appendix.

A.1 Thinking Like a Vertex

The first iteration of risk propagation[1] utilized Apache Giraph[2], an open-source version of the iterative graph-processing library, Pregel [60], which is based on the bulk synchronous parallel model of distributed computing [96]. Giraph follows the *"think like a vertex" paradigm* in which the algorithm is specified in terms of the local information available to a graph vertex [61].

Risk propagation was implemented as defined by Ayday, Yoo, and Halimi [3] and Ayday et al. [4], using the factor graph representation of the contact network. Moreover, the implementation assumed the use of Dataswyft Personal Data Accounts (PDAs)[3]. However, because the Exposure Notification API developed by Apple[4] and Google[5] does not permit remotely persisting ephemeral identifiers, the implementation assumed that an individual's geolocation data would be analyzed to generate the factor vertices in the factor graph (see Appendix A.5). Figure A.1 describes the high-level architecture. Callouts 1, 2, and 4 were implemented using a fan-out design in which a *ventilator* Lambda function divides the work amongst *worker* Lambda functions.

[1]`https://github.com/cwru-xlab/sharetrace-giraph`
[2]`https://giraph.apache.org`
[3]`https://www.dataswyft.io`
[4]`https://covid19.apple.com/contacttracing`
[5]`https://www.google.com/covid19/exposurenotifications`
[6]`https://aws.amazon.com/lambda`
[7]`https://aws.amazon.com/s3`
[8]`https://aws.amazon.com/emr`

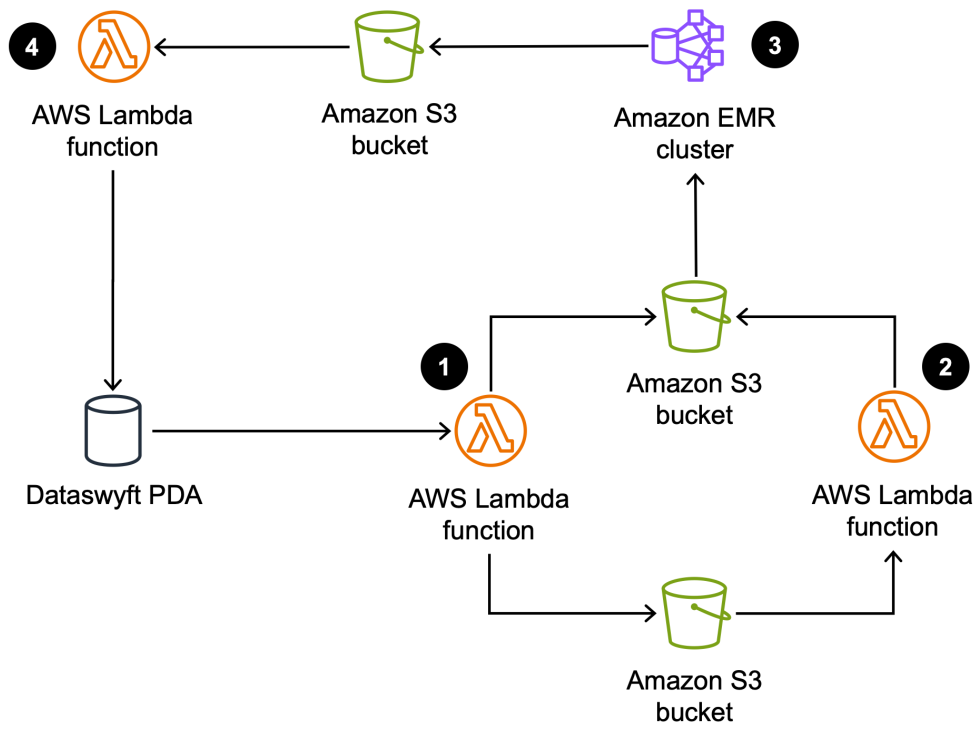

Figure A.1: ShareTrace batch-processing architecture. ❶ An AWS Lambda function[6] retrieves the recent risk scores and location data from the Dataswyft Personal Data Accounts (PDAs) of ShareTrace users. Risk scores are formatted as Giraph vertices and stored in an Amazon Simple Storage Service[7] (S3) bucket. Location data is stored in a separate S3 bucket. ❷ A Lambda function performs a contact search over the location data and stores the contacts as Giraph edges in the same bucket that stores the Giraph vertices. ❸ Amazon Elastic MapReduce[8] (EMR) runs risk propagation as a Giraph job and stores the exposure scores in an S3 bucket. ❹ A Lambda function stores the exposure score of each user in their respective PDA.

A couple factors prompted me to search for an alternative implementation.

1. *Dependency management incompatibility.* The primary impetus for reimplementation was the dependency conflicts between Giraph and other libraries. Despite several attempts (e.g., using different library versions, using different versions of Giraph, and forcing specific versions of transitive dependencies) to resolve the conflicts, a lack of personal development experience and stalled progress prompted me pursue alternative implementations.

2. *Implementation complexity.* For a relatively straightforward data flow, the architecture in Figure A.1 corresponded to over 4000 lines of source code. In retrospect, AWS Step Functions[9] could have been used to orchestrate the workflow, including the fan-out design pattern, which would have simplified the Lambda function implementations. Regarding the implementation of risk propagation, one-mode projection (first used in Appendix A.4) would have simplified the implementation since it avoids multiple types of vertices and messages.

A.2 Factor Subgraph Actors

In an attempt to simplify the design in Appendix A.1, I rewrote risk propagation using the Ray Python library[10]. While it claims to support actor-based programming, Ray only offers coarse-grained concurrency, with each actor being mapped to a physical core. To achieve parallelism, the factor graph was

[9] `https://aws.amazon.com/step-functions`
[10] `https://www.ray.io`

partitioned amongst the actors such that each actor maintained a subset of variable vertices *or* factor vertices. The graph topology was stored in shared memory since it was immutable. The lifetime of this design was brief for the following reasons.

1. *Poor performance.* Communication between Ray actors requires message serialization. Moreover, partitioning the factor graph into subsets of factor vertices and variable vertices results in maximal interprocess communication. Unsurprisingly, this choice of partitioning manifested in poor runtime performance.

2. *Design complexity.* Not using a framework, like Giraph, meant that this implementation required more low-level code to implement actor functionality and message passing. Regardless of the performance, the overall design of this implementation was poorly organized and overthought.

A.3 Driver-Monitor-Worker Framework

Based on the poor runtime performance and complexity of the previous approach, I speculated that centralizing the mutable aspects of risk propagation (i.e., the iterative exposure scores of each variable vertex) would improve both metrics. With this is mind, I designed the *monitor-worker-driver* (DMW) *framework*, which draws inspiration from the *tree of actors* design pattern[11]. Figure A.2 describes the framework.

For risk propagation, the driver creates the factor graph from the set of risk

[11] `https://docs.ray.io/en/latest/ray-core/patterns/tree-of-actors.html`

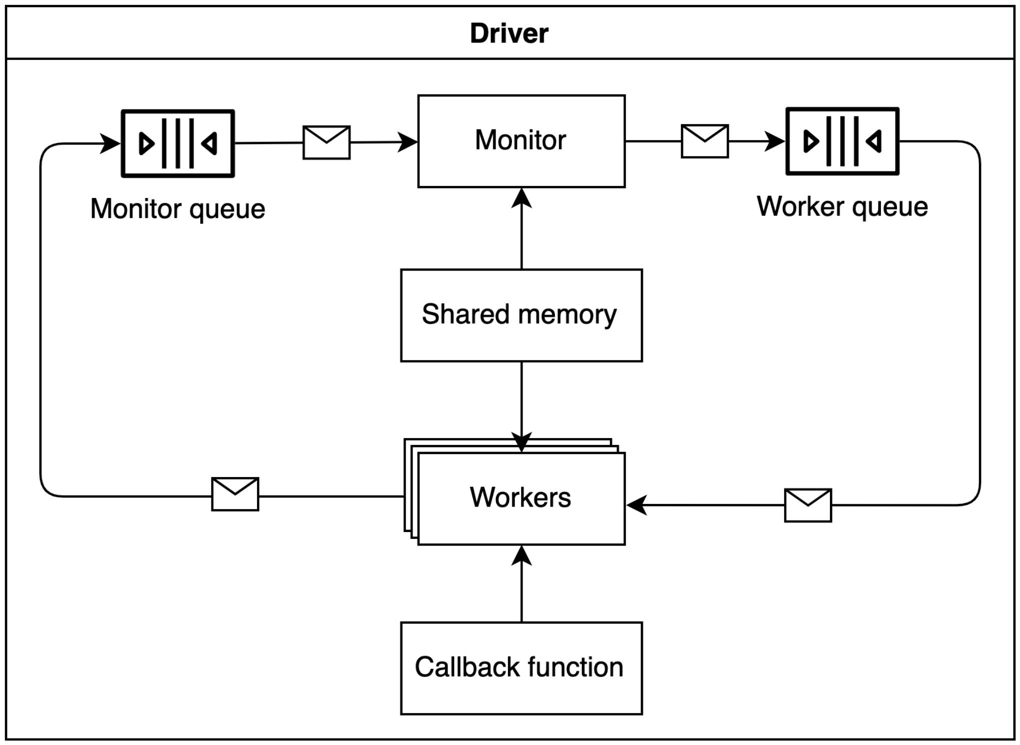

Figure A.2: Driver-monitor-worker framework. The *driver* is the entry point into the program. It initializes the monitor and workers, and then waits for termination. The *monitor* encapsulates the mutable state of the program and decides which messages are processed by workers. A *worker* is a stateless entity that processes the messages that the monitor puts in the *worker queue*. Worker behavior is defined by a *callback function*, which typically depends on the message contents. The side effects of processing a message are recorded as new messages and put in the *monitor queue*. This cycle repeats until some termination condition is satisfied. Any immutable state of the program can be stored in *shared memory* for efficient access.

scores S and contacts C, stores the factor graph in shared memory, sets the initial state of the monitor to be the maximum risk score of each individual, and puts all risk scores in the monitor queue. During message passing, the monitor maintains the exposure score for each variable vertex.

The DMW framework was the first approach that utilized the send coefficient to ensure the convergence and termination of message passing. However, because the DMW-based implementation assumed the factor graph representation of the contact network, the send coefficient was applied to both variable and factor messages.

Compared to Appendix A.2, this implementation provided a cleaner design and less communication overhead. However, what prompted yet another alternative implementation was its scalability. Because the monitor processes messages serially, it is a bottleneck for algorithms in which the workers perform fine-grained tasks. Indeed, the Ray documentation[12] notes that the parallelization of small tasks is an anti-pattern because the interprocess communication cost exceeds the benefit of multiprocessing. Unfortunately, the computation performed by factor vertices and variable vertices was fine-grained, so the scalability of the DMW framework was demonstrably poor.

A.4 Projected Subgraph Actors

The last alternative design of risk propagation is the subject of published work [91] and preprint [92] which were written during the coarse of my grad-

[12]`https://docs.ray.io/en/latest/ray-core/patterns/too-fine-grained-tasks.html`

uate studies. This approach built upon the insights from Appendix A.3 and Appendix A.2. Notably, that achieving parallelization required partitioning the contact network, but doing so in a way that minimized the communication cost. Rather than computing over the factor graph directly, Tatton et al. [91] projected the factor graph onto the variable vertices to produce a network of stateful workers. While Tatton et al. [91] claims that this approach is a viable distributed extension of the risk propagation algorithm first proposed by Ayday, Yoo, and Halimi [3], it still assumes an initial condition.

As with this work, Tatton et al. [91] found that increasing the send coefficient can provide improved message-passing efficiency. However, Tatton et al. [91] enforced a common send threshold across the contacts of an actor, so a lower send coefficient was found to be optimal. The performance benchmark that was conducted in Tatton et al. [91] encountered limitations with the Ray library, which was addressed in this work with the usage of Akka.

A.5 Contact Search

Though initial works on ShareTrace [3, 4] claim to use an individual's ephemeral identifiers to determine their contacts, this was practically infeasible at the time of development. As mentioned in Appendix A.1, the Exposure Notification API requires ephemeral identifiers to remain on an individual's mobile device. As such, I was tasked with devising an algorithm to find an individual's contacts based on their geolocation history; such an algorithm will be referred to as *contact search.* I ultimately discontinued this line of research in favor of the more scalable and privacy-preserving approach of ephemeral identifiers. Prior to this work, I was not familiar with computational geometry, nor the literature on the privacy-preserving extensions thereof. As such, the algorithms presented below are rudimentary and are not viable for practical usage.

Assume that the i-th individual's mobile device maintains a history L_i of their geolocation, such that $\ell_t \in L_i$ represents the geolocation of the i-th location at time t. The i-th and j-th individuals are *contacts* if the intersection of L_i and L_j contains a sequence of ϵ-proximal geolocations over a δ-contiguous time period (see Section 2.1).

Assume an individual's geolocation history represents a piecewise constant function. That is, if $\ell_{t_i}, \ell_{t_k} \in L$, $\ell_{t_j} \notin L$, and $t_j \in [t_i, t_k)$, then $\ell_{t_i} = \ell_{t_j}$ and $\ell_{t_j} \neq \ell_{t_k}$. If $\epsilon = 0$, then the problem of deciding if the i-th and j-th individuals are contacts can be reduced to the longest common substring problem. Using piecewise constant interpolation such that L_i and L_j start and end at the same time, then each geolocation history represents a string, where the position of a character is the time at which the individual was at that location. If there exists a string of at least length δ, then the i-th and j-th individuals are contacts. To find the most recent contact time, the two strings can be traversed in reverse order. Therefore, it takes time $O(n^2)$ to find all contacts in a population of n individuals. Figure A.3 provides a visual example of contact search.

The same string algorithm can be applied when $\epsilon > 0$ if geolocations are encoded to represent regions of space. One such encoding is *geohashing*, which

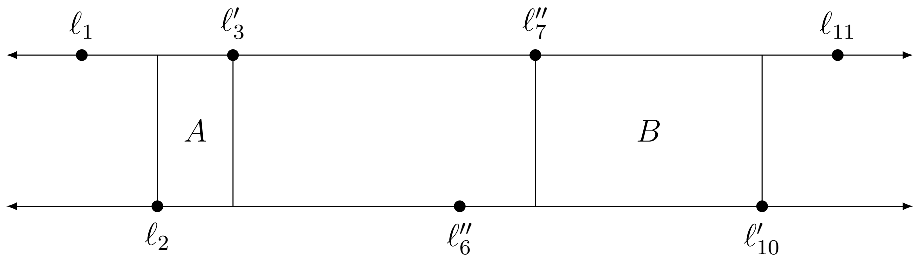

Figure A.3: Example of contact search. Each line denotes time, increasing from left to right. A point ℓ_i is a geolocation that is recorded at time i. Regions A and B indicate periods of time in which the two individuals were colocated, but, for the sake of example, only region B represents contact (e.g., $\delta > 1$).

transforms *geographic coordinates* (i.e., latitude-longitude ordered pairs [84, p. 5]) into alphanumeric strings, called *geohashes*, where the length of a geohash correlates with its geospatial precision [65]. This encoding obfuscates an individual's precise location, which provides some basic privacy. However, obfuscation can quickly lead to dense contact networks, which adversely impacts the accuracy and performance of an offline implementation of risk propagation.

Another approach to solving contact search when $\epsilon > 0$ is to construct a spatial data structure, such as a ball tree [47, 68], from the geolocations of all individuals. To find an individual's contacts, a fixed-radius near neighbors search [8, 11] can performed for each geolocation in the individual's history. Additional bookkeeping is needed to determine which individuals correspond to the neighbors of a geolocation.

Any metric can be paired with a ball tree. However, because geolocations are represented as geographic coordinates, metrics that assume a Cartesian coordinate system may be unsuitable. The simplest geometric models of the Earth is that of a sphere. Given two geographic coordinates, the problem of finding the length of the geodesic[13] between them is known as the *inverse geodetic problem* [87]. Assuming a spherical Earth, the solution to the inverse problem is to find the length of the segment that joins the two points on a great circle[14].

Let $\theta = \frac{d}{r}$ be the *central angle*, where d is the distance between the two points along the great circle of a sphere with radius r (see Figure A.4).

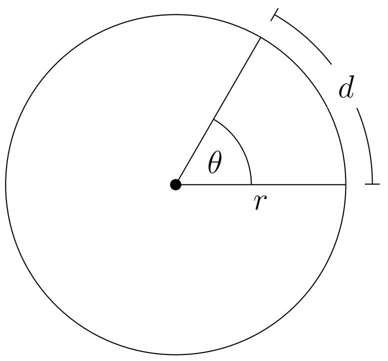

Figure A.4: Central angle of a great circle.

The *haversine* (i.e., the half "versed" sine) of a central angle θ is defined as

$$\operatorname{hav}\theta = \frac{\operatorname{vers}\theta}{2} = \frac{1-\cos\theta}{2} = \sin^2\frac{\theta}{2}. \tag{A.1}$$

The great-circle distance d between two points can be found by inversing (A.1) and solving,

$$d(\ell,\ell') = 2\cdot\arcsin\sqrt{\operatorname{hav}\frac{\phi-\phi'}{2} + \cos\phi\cdot\cos\phi'\cdot\operatorname{hav}\frac{\lambda-\lambda'}{2}},$$

where $\ell = (\phi,\lambda)$ is a latitude-longitude coordinate in radians [12, pp. 157–162].

[13]The *geodesic* is the shortest segment between two points on an ellipsoid [58].

[14]The *great circle* is the cross-section of a sphere that contains its center [58]

More advanced *geodetic datum* [58, pp. 71–130] could be used to provide better geospatial accuracy. Additionally, by projecting geodetic coordinates onto the plane, metrics that assume a Cartesian coordinate system could be used instead [58, pp. 265–326].

Appendix B

Data Structures

A *dynamic set* X is a mutable collection of distinct elements such that x is a pointer to an element in X. The value of the key attribute $x.key$ uniquely identifies x within X. A dynamic set X supports the following operations [18].

- $\textsc{Insert}(X, x)$ adds the element pointed to by x to X.
- $\textsc{Delete}(X, x)$ removes the element pointed to by x from X.
- $\textsc{Search}(X, k)$ returns a pointer x to an element in X such that $x.key = k$; or NIL, if no such element exists in X.

A *dynamic multiset* X is a dynamic set that allows multiple elements to have the same key [18]. The definitions of DELETE and SEARCH differ as follows.

- $\textsc{Delete}(X, x)$ removes the elements pointed to by x from X. Elements in X with the key value $x.key$ that are *not* pointed to by x are retained.
- $\textsc{Search}(X, k)$ returns a collection of pointers to the elements in X such that $x.key = k$, or NIL, if no such elements exists in X.

Appendix C

Pseudocode Conventions

Pseudocode conventions are mostly consistent with Cormen et al. [18].

- 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*: the "procedure receives its own copy of the parameters, and if it assigns a value to a parameter, 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 attributes of the object are not."

- A **return** statement “immediately transfers control back to the point of call in the calling procedure.”
- 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] Saleh Afroogh et al. “Tracing app technology: An ethical review in the COVID-19 era and directions for post-COVID-19”. In: *Ethics and Information Technology* 24.3 (2022). DOI: `10.1007/s10676-022-09659-6` (cit. on pp. 10, 32).

[2] Gul Agha. “ACTORS: A model of concurrent computation in distributed systems”. PhD thesis. MIT, 1985. URL: `https://hdl.handle.net/1721.1/6952` (cit. on pp. 19, 20, 22).

[3] 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: *Proceedings of the 12th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*. 2021. DOI: `10.1145/3459930.3469553` (cit. on pp. 10, 11, 14, 16–19, 31, 34, 75, 81).

[4] Erman Ayday et al. *ShareTrace: A smart privacy-preserving contact tracing solution by architectural design during an epidemic*. White paper. Case Western Reserve University, 2020 (cit. on pp. 19, 31, 34, 75, 81).

[5] Albert-Làszlò Barabàsi and Rèka Albert. “Emergence of scaling in random networks”. In: *Science* 286.5439 (1999), pp. 509–512. DOI: `10.1126/science.286.5439.509` (cit. on p. 48).

[6] Alain Barrat and Ciro Cattuto. “Temporal networks of face-to-face human interactions”. In: *Temporal Networks*. Springer, 2013. DOI: `10.1007/978-3-642-36461-7_10` (cit. on p. 28).

[7] Juan Benet. *IPFS - content addressed, versioned, P2P file system*. 2014. arXiv: `1407.3561 [cs.NI]` (cit. on pp. 34, 73).

[8] Jon Louis Bentley. *A survey of techniques for fixed radius near neighbor searching*. Tech. rep. Stanford University, 1975 (cit. on p. 83).

[9] Christopher M. Bishop. “Pattern recognition and machine learning”. In: *Information Science and Statistics*. Springer, 2006 (cit. on p. 15).

[10] Mark Briers, Marcos Charalambides, and Chris Holmes. *Risk scoring calculation for the current NHSx contact tracing app*. 2020. arXiv: `2005.11057 [cs.CY]` (cit. on p. 12).

[11] Sergey Brin. “Near neighbor search in large metric spaces”. In: *Proceedings of the 21th International Conference on Very Large Data Bases*. 1995, pp. 574–584 (cit. on p. 83).

[12] Glen Van Brummelen. *Heavenly mathematics: The forgotten art of spherical trigonometry*. Princeton University Press, 2013 (cit. on p. 84).

[13] Andrea Capponi et al. “A survey on mobile crowdsensing systems: Challenges, solutions, and opportunities”. In: *IEEE Communaction Surveys*

& Tutorials 21.3 (2019), pp. 2419–2465. DOI: `10.1109/comst.2019.2914030` (cit. on pp. 31–42, 72).

[14] Gavin C. Cawley and Nicola L.C. Talbot. “On over-fitting in model selection and subsequent selection bias in performance evaluation”. In: *Journal of Machine Learning Research* 11 (2010), pp. 2079–2107 (cit. on p. 68).

[15] Renato Cherini et al. “Toward deep digital contact tracing: Opportunities and challenges”. In: *IEEE Pervasive Computing* 22.4 (2023), pp. 15–25. DOI: `10.1109/mprv.2023.3320987` (cit. on pp. 9–11, 73).

[16] Hyunghoon Cho, Daphne Ippolito, and Yun William Yu. *Contact tracing mobile apps for COVID-19: Privacy considerations and related trade-offs*. 2020. arXiv: `2003.11511 [cs.CR]` (cit. on pp. 10, 11, 73).

[17] William Clinger. “Foundations of actor semantics”. PhD thesis. MIT, 1981. URL: `https://hdl.handle.net/1721.1/6935` (cit. on p. 19).

[18] Thomas H. Cormen et al. *Introduction to algorithms*. The MIT Press, 2022 (cit. on pp. 18, 86, 87).

[19] Meggan E. Craft. “Infectious disease transmission and contact networks in wildlife and livestock”. In: *Philosophical Transactions of the Royal Society B* 370.1669 (2015). DOI: `10.1098/rstb.2014.0107` (cit. on p. 27).

[20] Shujie Cui et al. “Collusion defender: Preserving subscribers’ privacy in publish and subscribe systems”. In: *IEEE Transactions on Dependable*

and Secure Computing 18.3 (2021), pp. 1051–1064. DOI: 10.1109/tdsc.2019.2898827 (cit. on p. 35).

[21] Leon Danon et al. “Networks and the epidemiology of infectious disease”. In: *Interdiscipinary Perspectives on Infectious Diseases* 2011 (2011). DOI: 10.1155/2011/284909 (cit. on p. 27).

[22] 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*. 2016, pp. 31–40. DOI: 10.1145/3001886.3001890 (cit. on p. 19).

[23] Paul Erdős and Alféd Rényi. “On random graphs I.” In: *Publicationes Mathematicae* 6.3–4 (1959), pp. 290–297. DOI: 10.5486/pmd.1959.6.3-4.12 (cit. on p. 48).

[24] Luca Ferretti et al. “Digital measurement of SARS-CoV-2 transmission risk from 7 million contacts”. In: *Nature* 626.7997 (2024), pp. 145–150. DOI: 10.1038/s41586-023-06952-2 (cit. on p. 73).

[25] Luca Ferretti et al. “Quantifying SARS-CoV-2 transmission suggests epidemic control with digital contact tracing”. In: *Science* 368.6491 (2020), eabb6936. DOI: 10.1126/science.abb6936 (cit. on p. 73).

[26] Michael A. Fligner and Timothy J. Killeen. “Distribution-free two-sample tests for scale”. In: *Journal of the American Statistical Association* 71.353 (1976), pp. 210–213. DOI: 10.1080/01621459.1976.10481517 (cit. on p. 64).

[27] Erich Gamma et al. *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley, 1995 (cit. on pp. 20, 44).

[28] Raghu K. Ganti, Fan Ye, and Hui Lei. "Mobile crowdsensing: Current state and future challenges". In: *IEEE Communications Magazine* 49.11 (2011), pp. 32–39. DOI: `10.1109/mcom.2011.6069707` (cit. on pp. 38, 41).

[29] Alexander E. Gorbalenya et al. "The species severe acute respiratory syndrome-related coronavirus: Classifying 2019-nCoV and naming it SARS-CoV-2". In: *Nature Microbiology* 5.4 (2020), pp. 536–544. DOI: `10.1038/s41564-020-0695-z` (cit. on p. 9).

[30] Bin Guo et al. "Mobile crowd sensing and computing: The review of an emerging human-powered sensing paradigm". In: *ACM Computing Surveys* 48.1 (2015), pp. 1–31. DOI: `10.1145/2794400` (cit. on p. 31).

[31] Prateek Gupta et al. *COVI-AgentSim: An agent-based model for evaluating methods of digital contact tracing*. 2020. arXiv: `2010.16004 [cs.CY]` (cit. on pp. 48, 73).

[32] Prateek Gupta et al. "Proactive contact tracing". In: *PLOS Digital Health* 2.3 (2023), pp. 1–19. DOI: `10.1371/journal.pdig.0000199` (cit. on pp. 9, 11, 73).

[33] Lea Hamner et al. "High SARS-CoV-2 attack rate following exposure at a choir practice – Skagit County, Washington, March 2020". In: *Morbidity and Mortality Weekly Report* 69.19 (2020). DOI: `10.15585/mmwr.mm6919e6` (cit. on p. 16).

[34] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (2020), pp. 357–362. DOI: `10.1038/s41586-020-2649-2` (cit. on p. 56).

[35] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: Data mining, inference, and prediction.* Springer, 2009 (cit. on p. 68).

[36] Trevor Hastie, Robert Tibshirani, and Martin Wainwright. *Statistical learning with sparsity: The lasso and generalizations.* CRC Press, 2015 (cit. on p. 68).

[37] Michael Haus et al. “Security and privacy in device-to-device (D2D) communication: A review”. In: *IEEE Communications Surveys & Tutorials* 19.2 (2017), pp. 1054–1079. DOI: `10.1109/comst.2017.2649687` (cit. on p. 9).

[38] Carl Hewitt. “Viewing control structures as patterns of passing messages”. In: *Artificial Intelligence* 8.3 (1977), pp. 323–364. DOI: `10.1016/0004-3702(77)90033-9` (cit. on p. 19).

[39] Carl Hewitt and Henry Baker. *Laws for communicating parallel processes.* Working paper. MIT Artificial Intelligence Laboratory, 1977 (cit. on p. 19).

[40] Carl Hewitt, Peter Bishop, and Richard Steiger. “A universal modular ACTOR formalism for artificial intelligence”. In: *Proceedings of the 3rd International Joint Conference on Artificial Intelligence.* 1973, pp. 235–

245. URL: https://dl.acm.org/doi/10.5555/1624775.1624804 (cit. on p. 19).

[41] Sture Holm. “A simple sequentially rejective multiple test procedure”. In: *Scandinavian Journal of Statistics* 6.2 (1979), pp. 65–70. URL: https://www.jstor.org/stable/4615733 (cit. on p. 66).

[42] Petter Holme. “Modern temporal network theory: A colloquium”. In: *The European Physical Journal B* 88.9 (2015). DOI: 10.1140/epjb/e2015-60657-4 (cit. on p. 27).

[43] Petter Holme and Jari Saramäki. “Temporal networks”. In: *Physics Reports* 519.3 (2012). DOI: 10.1016/j.physrep.2012.03.001 (cit. on pp. 27, 28).

[44] J. D. Hunter. “Matplotlib: A 2d graphics environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/mcse.2007.55 (cit. on p. 56).

[45] Brian Karrer and M. E. J. Newman. “Message passing approach for general epidemic models”. In: *Physical Review E* 82.1 (2010). DOI: 10.1103/physreve.82.016101 (cit. on p. 11).

[46] Navin Keizer et al. “A survey on content retrieval on the decentralised web”. In: *ACM Computing Surveys* 56.8 (2024). DOI: 10.1145/3649132 (cit. on pp. 34, 73).

[47] Ashraf M. Kibriya and Eibe Frank. “An empirical comparison of exact nearest neighbour algorithms”. In: *Knowledge Discovery in Databases:*

PKDD 2007. 2007, pp. 140–151. DOI: 10.1007/978-3-540-74976-9_16 (cit. on p. 83).

[48] Jeong Han Kim and Van H. Vu. “Generating random regular graphs”. In: *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*. 2003, pp. 213–222. DOI: 10.1145/780542.780576 (cit. on p. 48).

[49] Roger Koenker and Gilbert Bassett. “Regression quantiles”. In: *Econometrica* 46.1 (1978), pp. 33–50. DOI: 10.2307/1913643 (cit. on p. 68).

[50] Roger Koenker and José A. F. Machado. “Goodness of fit and related inference processes for quantile regression”. In: *Journal of the American Statistical Association* 94.448 (1999), pp. 1296–1310. DOI: 10.2307/2669943 (cit. on p. 68).

[51] Andreas Koher et al. “Contact-based model for epidemic spreading on temporal networks”. In: *Physical Review X* 9.3 (2019). DOI: 10.1103/physrevx.9.031017 (cit. on p. 27).

[52] Frank R. Kschischang, Brendan J. Frey, and Hans A. Loeliger. “Factor graphs and the sum-product algorithm”. In: *IEEE Transactions on Information Theory* 47.2 (2001). DOI: 10.1109/18.910572 (cit. on p. 13).

[53] Christiane Kuhn, Martin Beck, and Thorsten Strufe. “Covid notions: Towards formal definitions—and documented understanding—of privacy goals and claimed protection in proximity-tracing services”. In:

Online Social Networks and Media 22 (2021). DOI: 10.1016/j.osnem.2021.100125 (cit. on p. 73).

[54] Nicholas D. Lane et al. “A survey of mobile phone sensing”. In: *IEEE Communications Magazine* 48.9 (2010), pp. 140–150. DOI: 10.1109/mcom.2010.5560598 (cit. on pp. 38, 41).

[55] Trystan Leng et al. “The effect of notification window length on the epidemiological impact of COVID-19 contact tracing mobile applications”. In: *Communications Medicine* 2.1 (2022). DOI: 10.1038/s43856-022-00143-2 (cit. on p. 15).

[56] Bo Li and David Saad. “Impact of presymptomatic transmission on epidemic spreading in contact networks: A dynamic message-passing analysis”. In: *Physical Review E* 103.5 (2021). DOI: 10.1103/physreve.103.052303 (cit. on p. 11).

[57] Andrey Y. Lokhov et al. “Inferring the origin of an epidemic with a dynamic message-passing algorithm”. In: *Physical Review E* 90.1 (2014). DOI: 10.1103/physreve.90.012801 (cit. on p. 27).

[58] Zhiping Lu, Yunying Qu, and Shubo Qiao. *Geodesy: Introduction to geodetic datum and geodetic systems*. Springer, 2014 (cit. on pp. 84, 85).

[59] Huadong Ma, Dong Zhao, and Peiyan Yuan. “Opportunities in mobile crowd sensing”. In: *IEEE Communications Magazine* 52.8 (2014), pp. 29–35. DOI: 10.1109/mcom.2014.6871666 (cit. on pp. 37, 41).

[60] Grzegorz Malewicz et al. "Pregel: A system for large-scale graph processing". In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data.* 2010, pp. 135–146. DOI: 10.1145/1807167.1807184 (cit. on p. 75).

[61] Robert McCune, Tim Weninger, and Greg Madey. "Thinking like a vertex: A survey of vertex-centric frameworks for large-scale distributed graph processing". In: *ACM Computing Surveys* 48.2 (2015). DOI: 10.1145/2818185 (cit. on pp. 25, 75).

[62] Cristina Menni et al. "Real-time tracking of self-reported symptoms to predict potential COVID-19". In: *Nature Medicine* 26.7 (2020). DOI: 10.1038/s41591-020-0916-2 (cit. on p. 12).

[63] Dimitrios Michail et al. "JGraphT—A Java library for graph data structures and algorithms". In: *ACM Transactions on Mathematical Software* 46.2 (2020), pp. 1–29. DOI: 10.1145/3381449 (cit. on p. 44).

[64] James Moody. "The importance of relationship timing for diffusion". In: *Social Forces* 81.1 (2002). URL: https://www.jstor.org/stable/3086526 (cit. on p. 28).

[65] G.M. Morton. *A computer oriented geodetic data base and a new technique in file sequencing.* Tech. rep. International Business Machines, 1966 (cit. on p. 83).

[66] Arvind Narayanan et al. *A critical look at decentralized personal data architectures.* 2012. arXiv: 1202.4503 [cs.CY] (cit. on p. 34).

[67] M. E. J. Newman. “The structure and function of complex networks”. In: *SIAM Review* 45.2 (2003), pp. 167–256. DOI: 10.1137/S003614450342480 (cit. on p. 48).

[68] Stephen M. Omohundro. *Five balltree construction algorithms*. Tech. rep. International Computer Science Institute, 1989 (cit. on p. 83).

[69] Kiemute Oyibo et al. “Factors influencing the adoption of contact tracing applications: Systematic review and recommendations”. In: *Frontiers in Digital Health* 4 (2022). DOI: 10.3389/fdgth.2022.862466 (cit. on pp. 10, 32).

[70] Romualdo Pastor-Satorras et al. “Epidemic processes in complex networks”. In: *Reviews of Modern Physics* 87.3 (2015). DOI: 10.1103/revmodphys.87.925 (cit. on p. 27).

[71] Fabian Pedregosa et al. “Scikit-learn: Machine learning in Python”. In: *Journal of Machine Learning Research* 12.85 (2011), pp. 2825–2830. URL: https://jmlr.org/papers/v12/pedregosa11a.html (cit. on p. 56).

[72] Jocelyne Piret and Guy Boivin. “Pandemics throughout history”. In: *Frontiers in Microbiology* 11 (2021). DOI: 10.3389/fmicb.2020.631736 (cit. on p. 73).

[73] Francisco Pozo-Martin et al. “Comparative effectiveness of contact tracing interventions in the context of the COVID-19 pandemic: A systematic review”. In: *European Journal of Epidemiology* 38.3 (2023), pp. 243–266. DOI: 10.1007/s10654-023-00963-z (cit. on pp. 9, 10).

[74] Alex Preukschat and Drummond Reed. *Self-sovereign identity: Decentralized digital identity and verifiable credentials*. Manning, 2021 (cit. on pp. 34, 73).

[75] Joren Raymenants et al. “Empirical evidence on the efficiency of backward contact tracing in COVID-19”. In: *Nature Communications* 13.1 (2022). DOI: `10.1038/s41467-022-32531-6` (cit. on p. 15).

[76] Leonie Reichert, Samuel Brack, and Björn Scheuermann. “A survey of automatic contact tracing approaches using Bluetooth Low Energy”. In: *ACM Transactions on Computing for Healthcare* 2.2 (2021). DOI: `10.1145/3444847` (cit. on pp. 9, 10).

[77] Leonie Reichert, Samuel Brack, and Björn Scheuermann. *Ovid: Message-based automatic contact tracing*. Cryptology ePrint Archive, Paper 2020/1462. 2020. URL: `https://eprint.iacr.org/2020/1462` (cit. on pp. 10, 11, 73).

[78] Christopher S. Riolo, James S. Koopman, and Stephen E. Chick. “Methods and measures for the description of epidemiologic contact networks”. In: *Journal of Urban Health* 78.3 (2001). DOI: `10.1093/jurban/78.3.446` (cit. on p. 27).

[79] Rob Romijnders et al. “Protect your score: Contact-tracing with differential privacy guarantees”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 38.13 (2024), pp. 14829–14837. DOI: `10.1609/aaai.v38i13.29402` (cit. on p. 72).

[80] Frederico Schardong and Ricardo Custódio. "Self-sovereign identity: A systematic review, mapping and taxonomy". In: *Sensors* 22.15 (2022). DOI: `10.3390/s22155641` (cit. on pp. 34, 73).

[81] S. S. Shapiro and M. B. Wilk. "An analysis of variance test for normality (complete samples)". In: *Biometrika* 52.3/4 (1965), pp. 591–611. DOI: `10.2307/2333709` (cit. on p. 64).

[82] Ruizhe Shi et al. "A closer look into IPFS: Accessibility, content, and performance". In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 8.2 (2024). DOI: `10.1145/3656015` (cit. on pp. 34, 73).

[83] Viktoriia Shubina et al. "Survey of decentralized solutions with mobile devices for user location tracking, proximity detection, and contact tracing in the COVID-19 era". In: *Data* 5.4 (2020). DOI: `10.3390/data5040087` (cit. on p. 9).

[84] Jan Van Sickle. *Basic GIS coordinates*. CRC Press, 2004 (cit. on p. 83).

[85] Lucy Simko et al. "COVID-19 contact tracing and privacy: A longitudinal study of public opinion". In: *Digital Threats* 3.3 (2022). DOI: `10.1145/3480464` (cit. on p. 10).

[86] Sudhvir Singh et al. "How an outbreak became a pandemic: A chronological analysis of crucial junctures and international obligations in the early months of the COVID-19 pandemic". In: *Lancet* 398.10316 (2021), pp. 2109–2124 (cit. on p. 9).

[87] Lars E. Sjöberg and Masoud Shirazian. “Solving the direct and inverse geodetic problems on the ellipsoid by numerical integration”. In: *Journel of Surveying Engineering* 138.1 (2012), pp. 1–36 (cit. on p. 84).

[88] Charles Spearman. “The proof and measurement of association between two things”. In: *The American Journal of Psychology* 15.1 (1904), pp. 72–101. DOI: `10.2307/1412159` (cit. on p. 66).

[89] Syed Amin Tabish and Syed Nabil. “An age of emerging and reemerging pandemic”. In: *Health* 14 (2022), pp. 1021–1037. DOI: `10.4236/health.2022.1410073` (cit. on p. 73).

[90] Ichiro Takeuchi et al. “Nonparametric quantile estimation”. In: *Journal of Machine Learning Research* 7 (2006), pp. 1231–1264 (cit. on p. 68).

[91] Ryan Tatton et al. “ShareTrace: Contact tracing with the actor model”. In: *2022 IEEE International Conference on E-health Networking, Application & Services (HealthCom)*. 2022, pp. 13–18. DOI: `10.1109/healthcom54947.2022.9982762` (cit. on pp. 80, 81).

[92] Ryan Tatton et al. *ShareTrace: Contact tracing with the actor model*. 2022. arXiv: `2203.12445v3 [cs.DC]` (cit. on p. 80).

[93] Dennis Trautwein et al. “Design and evaluation of IPFS: A storage layer for the decentralized web”. In: *Proceedings of the ACM SIGCOMM 2022 Conference*. 2022, pp. 739–752. DOI: `10.1145/3544216.3544232` (cit. on pp. 34, 73).

[94] Carmela Troncoso et al. “Deploying decentralized, privacy-preserving proximity tracing”. In: *Communications of the ACM* 65.9 (2022), pp. 48–57. DOI: `10.1145/3524107` (cit. on p. 9).

[95] Carmela Troncoso et al. “Systematizing decentralization and privacy: Lessons from 15 years of research and deployments”. In: *Proceedings on Privacy Enhancing Technologies* 2017.4 (2017), pp. 307–329. DOI: `10.1515/popets-2017-0056` (cit. on pp. 34, 73).

[96] Leslie G. Valiant. “A bridging model for parallel computation”. In: *Communications of the ACM* 33.8 (1990). DOI: `10.1145/79173.79181` (cit. on p. 75).

[97] Raphael Vallat. “Pingouin: Statistics in Python”. In: *Journal of Open Source Software* 3.31 (2018), p. 1026. DOI: `10.21105/joss.01026` (cit. on p. 56).

[98] Ritchie Vink et al. *Pola-rs/polars: Python Polars 1.13.1*. Version py-1.13.1. 2024. DOI: `10.5281/zenodo.14153549` (cit. on p. 56).

[99] Pauli Virtanen et al. “SciPy 1.0: Fundamental algorithms for scientific computing in python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: `10.1038/s41592-019-0686-2` (cit. on p. 56).

[100] Michael L. Waskom. “Seaborn: Statistical data visualization”. In: *Journal of Open Source Software* 6.60 (2021). DOI: `10.21105/joss.03021` (cit. on p. 56).

[101] Duncan J. Watts and Steven H. Strogatz. "Collective dynamics of 'small-world' networks". In: *Nature* 393.6684 (1998), pp. 440–442 (cit. on p. 48).

[102] Jacqui Wise. "Covid-19: WHO declares end of global health emergency". In: *BMJ* 381 (2023). DOI: `10.1136/bmj.p1041` (cit. on p. 73).

[103] Tao Zhou et al. "Bipartite network projection and personal recommendation". In: *Physical Review E* 76.4 (2007) (cit. on p. 19).

[104] Na Zhu et al. "A novel coronavirus from patients with pneumonia in china, 2019". In: *New England Journal of Medicine* 382.8 (2020), pp. 727–733. DOI: `10.1056/nejmoa2001017` (cit. on p. 9).

[105] Lorenzo Zino and Ming Cao. "Analysis, prediction, and control of epidemics: A survey from scalar to dynamic network models". In: *IEEE Circuits and Systems Magazine* 21.4 (2021). DOI: `10.1109/mcas.2021.3118100` (cit. on p. 27).