

Chapter 1

Risk Propagation

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 is a timestamped infection probability where $s \in [0, 1]$ and $t \in \mathbb{N}$ 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 (Menni et al., 2020); 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 dis-

tribution 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 nodes (i.e., contacts).

Let $G = (X, F, E)$ be a *factor graph* where X is the set of variable nodes, F is the set of factor nodes, and E is the set of edges incident between them (Kschischang et al., 2001).

A *variable node* $x : \Omega \rightarrow \{0, 1\}$ is a random variable that represents the infection status of an individual, where the sample space is $\Omega = \{healthy, infected\}$ and

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

Thus, $p_t(x_i) = s_t$ is a risk score of the i -th individual.

A *factor node* $f : X \times X \rightarrow [0, 1]$ defines the transmission of infection risk between two contacts. Specifically, contact between the i -th and j -th individual is represented by the factor node $f(x_i, x_j) = f_{ij}$, which is adjacent to the variable nodes x_i, x_j . This work and Ayday et al. (2021) assume risk transmission is a symmetric function, $f_{ij} = f_{ji}$. However, it may be extended to account for an individual's susceptibility and transmissibility such that $f_{ij} \neq f_{ji}$. Figure 1.1 depicts a factor graph that reflects the domain constraints.

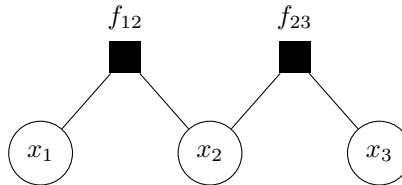


Figure 1.1: A factor graph of 3 variable nodes and 2 factor nodes.

1.1 Synchronous Risk Propagation

Ayday et al. (2021) 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

$$S_i = \{ s_t \mid \tau - t < T_s \} \in S \quad (1.1)$$

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

$$C = \{ (i, j, t) \mid i \neq j, \tau - t < T_c \} \quad (1.2)$$

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*¹. Naturally, risk scores and contacts have finite relevance, so (1.1) and (1.2) 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 $X = \emptyset$.

¹While Ayday et al. (2021) require contact over a δ -contiguous period of time, the Centers for Disease Control and Prevention (2021) account for contact over a 24-hour period.

1.1.1 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 node x_i to the factor node 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 (Bishop, 2006, pp. 411–415).

1.1.2 Factor Messages

A *factor message* $\lambda_{ij}^{(n)}$ from the factor node f_{ij} to the variable node 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 distance between the nodes 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 expression

$t_{ij} + \beta$ is called the *buffered contact time*.

3. Risk transmission between contacts is incomplete. Thus, a risk score decays exponentially along its transmission path in G at a rate of $\log \alpha$, where $\alpha \in (0, 1)$ is the *transmission rate*. Figure 1.2 visualizes this decay.

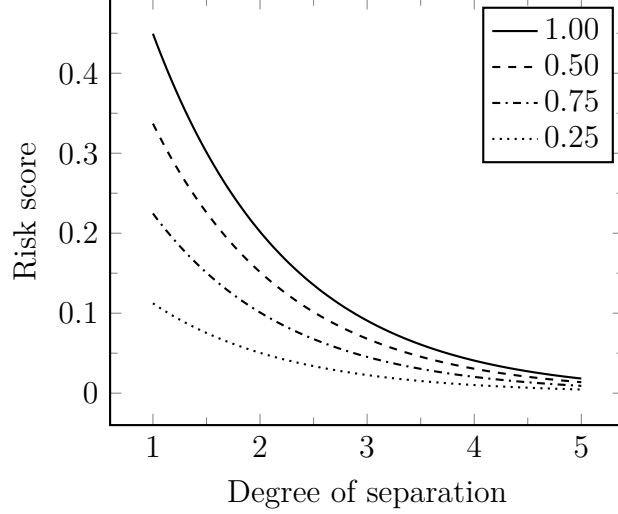


Figure 1.2: Exponential decay of risk scores.

To summarize, 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 .

Ayday et al. (2021) 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

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

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 node 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 nodes 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.

1.1.3 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 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 et al. (2021) 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:  $(X, F, E) \leftarrow \text{CREATE-FACTOR-GRAPH}(C)$ 
2:  $n \leftarrow 1$ 
3: for each  $x_i \in X$ 
4:    $R_i^{(n-1)} \leftarrow \text{top } K \text{ of } S_i$ 
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:      $\mu_{ij}^{(n)} \leftarrow R_i^{(n-1)} \setminus \{\lambda_{ji}^{(k)} \mid k \in [1 \dots n-1]\}$ 
10:    for each  $\{x_i, f_{ij}\} \in E$ 
11:       $\lambda_{ij}^{(n)} \leftarrow \max \{\alpha s_t \mid s_t \in \mu_{ij}^{(n)}, t < t_{ij} + \beta\}$ 
12:    for each  $x_i \in X$ 
13:       $R_i^{(n)} \leftarrow \text{top } K \text{ of } \{\lambda_{ji}^{(n)} \mid f_{ij} \in N_i\}$ 
14:    for each  $x_i \in X$ 
15:       $r_i^{(n-1)} \leftarrow r_i^{(n)}$ 
16:       $r_i^{(n)} \leftarrow \max R_i^{(n)}$ 
17:     $n \leftarrow n + 1$ 
18: return  $\mathbf{r}^{(n)}$ 

```

1.2 Asynchronous Risk Propagation

While RISK-PROPAGATION offers proof of concept, it is not viable for real-world application. RISK-PROPAGATION is an *offline algorithm* (Cormen et al., 2022), because it requires the contact and health information of all individuals

as input. As Ayday et al. (2021) note, this centralization of personal data is not privacy-preserving. RISK-PROPAGATION is also inefficient. Most exposure scores are not likely to change across frequent invocations, which implies communication overhead and computational redundancy. To mitigate this inefficiency, Ayday et al. (2020) suggest running RISK-PROPAGATION once or twice per day. This cadence introduces substantial delay in updating individuals’ exposure scores. Timely information is essential for individual and collective health, especially during a pandemic.

To address the limitations of RISK-PROPAGATION, Ayday et al. (2021) propose decentralizing the factor graph such that the processing entity (e.g., mobile device or “personal cloud”) associated with the i -th individual maintains the state of the i -th variable node and the neighboring factor nodes. More generally, Ayday et al. (2021) envision risk propagation as a decentralized communication protocol for informing individuals of their infection risk. Such a message-passing protocol naturally aligns with the *actor model* which describes concurrent computation as the sending and processing of messages amongst *actors* (Appendix C).

1.2.1 Actor Behavior

An actor in the ShareTrace actor system corresponds to an individual. The CREATE-ACTOR operation (Agha, 1985) initializes an actor a with the following attributes.

- $a.exposure$: the current exposure score of the individual. This attribute is either a symptom score, a risk score sent by another actor, or the null

risk score (see NULL-RISK-SCORE).

- *a.contacts*: a *dictionary* (Appendix B) of contacts. In the context of an actor, a contact is a *proxy* (Gamma et al., 1995) of the actor that represents an individual with which the individual represented by this actor was physically proximal. That is, if the i -th individual interacted with the j -th individual, then $a_i.contacts$ contains a contact c such that $c.key = c.name$ is a name of the j -th actor and $c.time$ is the most recent time of contact. This attribute extends the concept of *actor acquaintances* (Agha, 1985; Hewitt, 1977; Hewitt and Baker, 1977) to be time-varying.
- *a.scores*: a dictionary of exposure scores such that $s.key$ for an exposure score s is the time interval during which $a.exposure = s$. The null risk score is returned for queries in which the dictionary does not contain a risk score with a key that intersects the given query interval. Figure 1.3 depicts a hypothetical step function that *a.scores* represents.

NULL-RISK-SCORE

```
1:  $s.value \leftarrow 0$   
2:  $s.time \leftarrow 0$   
3:  $s.sender \leftarrow \text{NIL}$   
4: return  $s$ 
```

CREATE-ACTOR

```

1:  $a.contacts \leftarrow \emptyset$ 
2:  $a.scores \leftarrow \emptyset$ 
3:  $a.exposure \leftarrow \text{NULL-RISK-SCORE}$ 
4: return  $a$ 

```

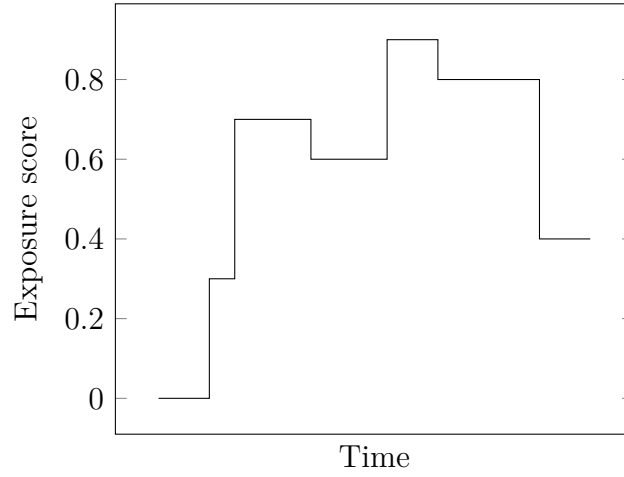


Figure 1.3: Historical exposure scores of an actor.

The interface of an actor is primarily defined by two types of messages: contacts and risk scores. Based on Section 1.1, 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.

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 the actions an actor performs upon receiving a risk score. The key initially associated with the risk score is the time interval for which it is relevant. For the dictionary $a.scores$, MERGE preserves the mapping invariant defined above such that risk scores are ordered first by value and then by time. Thus, $s.key \subseteq [s.time, s.time + T_s)$ for all exposure scores in $a.scores$. The UPDATE-EXPOSURE-SCORE operation describes how $a.exposure$ is updated. The dictionary $a.contacts$ is assumed to only contain unexpired contacts. Additional context is needed before specifying APPLY-RISK-SCORE in detail. For now, it is sufficient to know that the operation uses the risk score to update the state of a contact.

```

HANDLE-RISK-SCORE( $a, s$ )
1: if RISK-SCORE-TTL( $s$ ) > 0
2:    $s.key \leftarrow [s.time, s.time + T_s)$ 
3:   MERGE( $a.scores, s$ )
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( $a.exposure$ ) ≤ 0
4:    $a.exposure \leftarrow \text{MAXIMUM}(a.scores)$ 

```

For the moment, assume that APPLY-RISK-SCORE is equivalent to com-

putting a factor message (see Section 1.1). Line 2 indicates that the risk score s is copied and updated.

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*. For asynchronous risk propagation to be scalable and cost-efficient, actors should only send risk scores that offer new information to other actors. 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.

The objective of sending a risk score to another actor is to update its exposure score. Based on HANDLE-RISK-SCORE, it is only necessary to send an actor risk scores with values greater than its current exposure score. However, an actor is only privy to the risk scores that it sends. Thus, an actor can associate a *send threshold* for a contact that must be exceeded for a risk score to be sent to the target actor. To permit a trade-off between accuracy and efficiency of asynchronous risk propagation, 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: $c.threshold \leftarrow s'$

Below is the definition of APPLY-RISK-SCORE that incorporates this new aspect of message passing. 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 this update behavior. The latter predicate of Line 1 stems from the fact that the send threshold is a risk score and thus subject to expiry. The former predicate is more subtle and will be revisited shortly. The MAXIMUM-OLDER-THAN is the same as MAXIMUM with the additional restriction that the interval key intersects with the query interval $(-\infty, c.time + \beta)$. In this way, the returned risk score is always relevant to the contact. As with APPLY-RISK-SCORE, the risk score retrieved from $a.scores$ is also 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: $s \leftarrow \text{MAXIMUM-OLDER-THAN}(a.scores, c.time + \beta)$
- 3: $s'.value \leftarrow \alpha \cdot s.value$
- 4: SET-SEND-THRESHOLD(c, s')

The UPDATE-SEND-THRESHOLD is invoked during APPLY-RISK-SCORE, so another modification to the operation is defined below.

```

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, there are two cases in which the send threshold has a value of 0:

1. when initially assigning the send threshold to be the null risk score; and
2. when no key interval in $a.scores$ intersects the query interval, and thus the null risk score again is assigned the send threshold.

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 contact a relevant risk score. In the worst case, this prevents *any* risk score from being sent to the target actor, thus providing the individual associated with target actor a false sense of security that they are not at risk of being infected. From a broader message-passing perspective, updating the send threshold to a non-null risk score *before* applying the risk score received by the actor may introduce a non-trivial amount of inaccuracy. To summarize, updating the send threshold only

when its value is nonzero ensures correct message-passing behavior between actors.

Up until this point, the refinements to **APPLY-RISK-SCORE** have focused on ensuring that message passing terminates and correctly adjusts over time, as risk scores expire. Prior to concluding this section, a message-passing optimization is introduced. Over a given period of time, an actor may receive several risk scores that are then propagated to multiple contacts. Intuitively, rather than sending multiple risk scores, it would be more efficient to send just the final risk score. To increase the likelihood of that this occurs, **APPLY-RISK-SCORE** can be modified so that a contact “buffers” a single risk score that is intended to be sent to the target actor. Upon receiving a *flush timeout* message, the actor then “flushes” all contacts by sending the buffered message of each contact to the respective target actor. This is also known as *sender-side aggregation* in which the contact is an *aggregator* of risk scores.

The final iteration **APPLY-RISK-SCORE** integrates sender-side aggregation. Moreover, **HANDLE-FLUSH-TIMEOUT** clarifies the concept of “flushing” a contact. A flush timeout is assumed to be a periodically occurring “self” message.

```

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:   if  $c.name \neq s.sender$ 
6:      $c.buffered \leftarrow s'$ 

```

HANDLE-FLUSH-TIMEOUT(a)

- 1: **for each** $c \in a.contacts$
- 2: **if** $c.buffered \neq \text{NIL}$
- 3: SEND($c.name, c.buffered$)
- 4: $c.buffered \leftarrow \text{NIL}$

To conclude the specification of actor behavior, the HANDLE-CONTACT operation indicates how an actor responds when a new contact or contact with a newer contact time is received. Similar to HANDLE-RISK-SCORE, expired contacts are not processed. The MERGE operation for $a.contacts$ differs from how its used for $a.scores$. Specifically, if a contact with the same key already exists, its contact time is updated to that of the new contact; all other state of the previous contact is maintained.

HANDLE-CONTACT(a, c)

- 1: **if** CONTACT-TTL(c) > 0
- 2: $c.threshold \leftarrow \text{NULL-RISK-SCORE}$
- 3: $c.buffered \leftarrow \text{NIL}$
- 4: $c.key \leftarrow c.name$
- 5: MERGE($a.contacts, c$)
- 6: $s \leftarrow \text{MAXIMUM-OLDER-THAN}(a.scores, c.time + \beta)$
- 7: APPLY-RISK-SCORE(a, c, s)

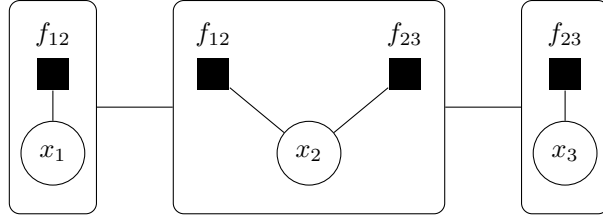


Figure 1.4: One-mode projection onto the variable nodes in Figure 1.1.

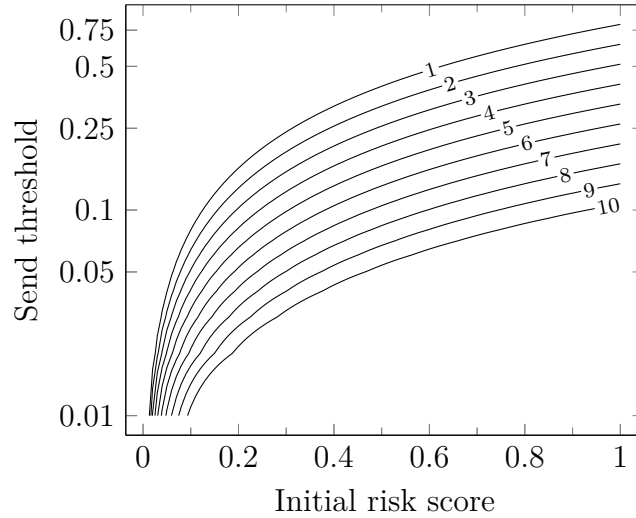


Figure 1.5: Estimated message reachability. Contour lines are shown for integral reachability values. Given an initial risk score and message reachability, a contour line indicates an upper bound on the permissible send threshold.

1.2.2 Message Reachability

1.3 Mobile Crowdsensing

Mobile crowdsensing (MCS) is a “sensing paradigm that empowers ordinary citizens to contribute data sensed or generated from their mobile devices” that is aggregated “in the cloud for crowd intelligence extraction and human-centric service delivery” (Guo et al., 2015). Over the past decade, substantial research has been conducted on defining and classifying MCS applications (Capponi

et al., 2019; Guo et al., 2015, and references therein). While not discussed in previous work Ayday et al. (2020, 2021), ShareTrace is a MCS application. The following characterization of ShareTrace assumes the four-layered architecture of a MCS application Capponi et al. (2019), which offers a comprehensive set of classification criteria that To offer a clear comparison, Table 1.1 follows the same structure as Capponi et al. (2019). Some aspects of the architecture, namely sampling frequency and sensor activity, are marked according to how ShareTrace is described in previous work, rather than how it would function to optimize for energy efficiency. More detail is provided below in this regard. When classifying ShareTrace as an MCS application, the following description is helpful:

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

1.3.1 Application Layer

1.3.1.1 Application Tasks

Task Scheduling *Proactive scheduling* allows users to decide when and where they contribute sensing data, while *reactive scheduling* requires that “a user receives a request, and upon acceptance, accomplishes a task” Capponi et al. (2019). ShareTrace follows proactive scheduling where the sensing task is to detect proximal interactions with other users. Naturally, the scheduling of this task is at the discretion of the ShareTrace user.

Task Assignment *Centralized assignment* assumes that “a central authority distributes tasks among users.” Conversely, with *decentralized assignment*, “each participant becomes an authority and can either perform the task or forward it to other users. This approach is very useful when users are very interested in specific events or activities” Capponi et al. (2019). The latter naturally aligns with ShareTrace in which each user is responsible for their own interactions that are both temporally and spatially specific.

Task Execution With *single-task execution*, MCS applications assign one type of task to users, while *multi-task execution* assigns multiple types of tasks Capponi et al. (2019). ShareTrace only involves the single task of sensing proximal interactions. Alternatively, ShareTrace could be defined more abstractly as sensing infection risk through interactions and user symptoms. In this case, ShareTrace would follow a multi-task execution model, where the task of sensing user symptoms is achieved through user reporting or bodily sensors.

1.3.1.2 Application Users

User Recruitment *Volunteer-based recruitment* is when citizens can “join a sensing campaign for personal interests...or willingness to help the community,” while *incentive-based recruitment* promotes participation and offers control over the recruitment rate...These strategies are not mutually exclusive and users can first volunteer and then be rewarded for quality execution of sensing tasks” Capponi et al. (2019). ShareTrace assumes volunteer-based recruitment. However, as a decentralized application (dApp) that aligns with the

principles of self-sovereignty, an incentive structure that rewards users with verifiable, high-quality data is plausible.

User Selection *User-centric selection* is when “contributions depend only on participants['] willingness to sense and report data to the central collector, which is not responsible for choosing them.” *Platform-centric selection* is “when the central authority directly decides data contributors...Platform centric decisions are taken according to the utility of data to accomplish the targets of the campaign” Capponi et al. (2019). ShareTrace employs user-centric selection, because the purpose of the application is to passively sense the user’s interactions and provide them with the knowledge of their infection risk.

User Type A *contributor* “reports data to the MCS platform with no interest in the results of the sensing campaign” and “are typically driven by incentives or by the desire to help the scientific or civil communities.” A *consumer* joins “a service to obtain information about certain application scenario[s] and have a direct interest in the results of the sensing campaign” Capponi et al. (2019). For ShareTrace, users can be either a consumer or a contributor but are likely biased toward the former.

1.3.2 Data Layer

1.3.2.1 Data Management

Data Storage *Centralized storage* involves data being “stored and maintained in a single location, which is usually a database made available in the cloud. This approach is typically employed when significant processing or data aggregation is required.” *Distributed storage* “is typically employed for delay-tolerant applications, i.e., when users are allowed to deliver data with a delayed reporting” Capponi et al. (2019). For sensing human interaction, ShareTrace relies on distributed storage in the form of Dataswift Personal Data Accounts. Moreover, ShareTrace is delay-tolerant, so distributed storage is most appropriate. However, for reporting the population-level risk distribution, it is likely that centralized storage would be used.

Data Format *Structured data* is standardized and readily analyzable. *Unstructured data*, however, requires significant processing before it can be used Capponi et al. (2019). ShareTrace deals with structured data (e.g., user symptoms, actor URIs).

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

1.3.2.2 Data Processing

Data Pre-processing *Raw data output* implies that no modification is made to the sensed data. *Filtering and denoising* entail “removing irrelevant and redundant data. In addition, they help to aggregate and make sense of data while reducing at the same time the volume to be stored” Capponi et al. (2019). ShareTrace only retains the actor URIs that correspond to valid contacts (i.e., lasting at least 15 minutes).

Data Analytics *Machine learning (ML) and data mining analytics* are not real-time. They “aim to infer information, identify patterns, or predict future trends.” On the contrary, *real-time analytics* consist of “examining collected data as soon as it is produced by the contributors” Capponi et al. (2019). ShareTrace aligns with the former category since it aims to infer the infection risk of users.

Data Post-processing *Statistical post-processing* “aims at inferring proportions given quantitative examples of the input data. *Prediction post-processing* aims to determine “future outcomes from a set of possibilities when given new input in the system” Capponi et al. (2019). ShareTrace applies predictive post-processing via risk propagation.

1.3.3 Communication Layer

1.3.3.1 Communication Technology

Infrastructured Technology *Cellular* connectivity “is typically required from sensing campaign[s] that perform real-time monitoring and require data reception as soon as it is sensed.” *WLAN* “is used mainly when sensing organizers do not specify any preferred reporting technologies or when the application domain permits to send data” at “a certain amount of time after the sensing process” Capponi et al. (2019). Infrastructured technology is also referred to as the *infrastructured transmission paradigm* Ma et al. (2014). ShareTrace does not require cellular infrastructure, because it is delay-tolerant and thus only requires WLAN.

Infrastructure-less Technology *Infrastructure-less technologies* “consists of device-to-device (D2D) communications that do not require any infrastructure...but rather allow devices in the vicinity to communicate directly.” Technologies include *WiFi-Direct*, *LTE-Direct*, and *Bluetooth* (Capponi et al., 2019). Infrastructure-less technology is also called the *opportunistic transmission paradigm* (Ma et al., 2014). ShareTrace uses Bluetooth because of its energy efficiency and short range.

1.3.3.2 Data Reporting

Upload mode With *relay uploading*, “data is delivered as soon as collected.” *Store-and-forward* “is typically used in delay-tolerant applications when campaigns do not need to receive data in real-time” (Capponi et al., 2019). Because

ShareTrace is delay-tolerant, it uses store-and-forward uploading.

Methodology *Individualized sensing* is “when each user accomplishes the requested task individually and without interaction with other participants.” *Collaborative sensing* is when “users communicate with each other, exchange data[,] and help themselves in accomplishing a task or delivering information to the central collector. Users are typically grouped and exchange data exploiting short-range communication technologies, such as WiFi-[D]irect or Bluetooth...Note that systems that create maps merging data from different users are considered individual because users do not interact between each other to contribute” (Capponi et al., 2019). The methodology of sensing is similar to the *sensing scale* which is typically dichotomized as *personal* (Ganti et al., 2011; Lane et al., 2010) (i.e., individualized) and *community* (Ganti et al., 2011) or *group* (Lane et al., 2010) (i.e., collaborative). ShareTrace is inherently collaborative, relying on mobile devices to exchange actor URIs to estimate infection risk. Thus, collaborative sensing is used.

Timing *Timing* is based on whether devices “should sense in the same period or not.” *Synchronous timing* “includes cases in which users start and accomplish at the same time the sensing task. For synchronization purposes, participants communicate with each other.” *Asynchronous timing* occurs “when users perform sensing activity not in time synchronization with other users” (Capponi et al., 2019). ShareTrace requires synchronous timing, because contact sensing inherently requires synchronous communication between the involved devices.

1.3.4 Sensing Layer

1.3.4.1 Sensing Elements

Sensor Deployment *Dedicated deployment* involves the use of “non-embedded sensing elements,” typically for a specific task. *Non-dedicated deployment* utilizes sensors that “do not require to be paired with other devices for data delivery but exploit the communication capabilities of mobile devices” (Capponi et al., 2019). ShareTrace relies on non-dedicated deployment since it relies on Bluetooth that is 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” (Capponi et al., 2019). ShareTrace uses Bluetooth, which may be considered on-demand. While energy efficient, users do control when it is enabled. Ideally, ShareTrace would also use always-on sensors to enable Bluetooth with context awareness (i.e., that the user is carrying or nearby the device).

Acquisition *Homogeneous acquisition* “involves only one type of data and it does not change from one user to another one,” while *heterogeneous acquisition* “involves different data types usually sampled from several sensors” (Capponi

et al., 2019). ShareTrace is homogeneous, because all users sense the same data from one type of sensor.

1.3.4.2 Data Sampling

Sampling Frequency *Continuous sensing* “indicates tasks that are accomplished regularly and independently [of] the context of the smartphone or the user[’s] activities.” *Event-based sensing* is “data collection [that] starts after a certain event has occurred. In this context, an event can be seen as an active action from a user or the central collector, but also a given context awareness” (Capponi et al., 2019). ShareTrace sensing is continuous but would ideally be event-based to conserve device energy.

Sensing Responsibility When the *mobile device* is responsible, “devices or users take sampling decisions locally and independently from the central authority...When devices take sampling decisions, it is often necessary to detect the context [of the] smartphones and wearable devices...The objective is to maximize the utility of data collection and minimize the cost of performing unnecessary operations.” When the *central collector* is responsible, they make “decisions about sensing and communicate them to the mobile devices” (Capponi et al., 2019). Given the human-centric nature of the ShareTrace sensing task, mobile devices are responsible.

User involvement *Participatory involvement* “requires active actions from users, who are explicitly asked to perform specific tasks. They are responsible to consciously meet the application requests by deciding when, what,

where, and how to perform sensing tasks.” *Opportunistic involvement* means that “users do not have direct involvement, but only declare their interest in joining a campaign and providing their sensors as a service. Upon a simple handshake mechanism between the user and the MCS platform, a MCS thread is generated on the mobile device (e.g., in the form of a mobile app), and the decisions of what, where, when, and how to perform the sensing are delegated to the corresponding thread. After having accepted the sensing process, the user is totally unconscious with no tasks to perform and data collection is fully automated...The smartphone itself is context-aware and makes decisions to sense and store data, automatically determining when its context matches the requirements of an application. Therefore, coupling opportunistic MCS systems with context-awareness is a crucial requirement” (Capponi et al., 2019). Earlier works on MCS refer to user involvement as the *sensing paradigm* (Ganti et al., 2011; Lane et al., 2010; Ma et al., 2014). ShareTrace is opportunistic, ideally with context-awareness.

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

Table 1.1: ShareTrace classification using the four-layered architecture of a mobile crowdsensing application Capponi et al. (2019). Always (•); with context-awareness (○).

Appendix A

Pseudocode Conventions

Pseudocode conventions are mostly consistent with Cormen et al. (2022).

- 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 *attribute* 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.” Thus, attribute assignment “is visible

if the calling procedure has a pointer to the same object.”

- 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 $\{ x \in X \mid \text{PREDICATE}(x) \}$ is used to create a subset of a collection X in place of constructing an explicit data structure.

Appendix B

Data Structures

Let a *dynamic set* X be a mutable collection of distinct elements. Let x be a pointer to an element in X such that $x.key$ uniquely identifies the element in X . Let a *dictionary* be a dynamic set that supports insertion, deletion, and membership querying (Cormen et al., 2022).

- $\text{INSERT}(X, x)$ adds the element pointed to by x to X .
- $\text{DELETE}(X, x)$ removes the element pointed to by x from X .
- $\text{SEARCH}(X, k)$ returns a pointer x to an element in the set X such that $x.key = k$; or NIL , if no such element exists in X .
- $\text{MERGE}(X, x)$ adds the element pointed to by x , if no such element exists in X ; otherwise, the result of applying a function to x and the existing element is added to X .
- $\text{MAXIMUM}(X)$ returns a pointer x to the maximum element of the totally ordered set X ; or NIL if X is empty.

Appendix C

Actor Model

The *actor model* is a local model of concurrent computing that defines computation as a strict partial ordering of events (Hewitt, 1977; Hewitt and Baker, 1977). An *event* is defined as “a transition from one local state to another” (Hewitt and Baker, 1977); or, more concretely, “a *message* arriving at a computational agent called an *actor*.” The actor that receives the message of an event is called the *target* of the event.

Upon receipt, an actor may send messages to itself or other actors, update its local state, or create other actors.

May send messages to other actors (Clinger) Update its local state (Clinger)
Create other actors

Events caused

An event may *activate* subsequent events

Ordering laws:

Passing messages between actors is the only means of communication. Thus, control and data flow are inseparable in the actor model (Hewitt et al., 1973).

A minimal specification of an actor includes its *name* and *behavior*, or how it acts upon receiving a message. Clinger (1981) explains,

An actor's name is a necessary part of its description because two different actors may have the same behavior. An actor's behavior is a necessary part of its description because the same actor may have different behaviors at different times.

acquaintances, or a finite collection of names of the actors that it knows about (Hewitt et al., 1973).

Thus, the specification of an actor system consists of the following (Hewitt, 1977).

- Deciding on the natural kinds of actors (objects) to have in the system to be constructed.
- Deciding for each kind of actor what kind of messages it should receive.
- Deciding for each kind of actor what it should do when it receives each kind of message.

Bibliography

Gul Agha. *Actors: A model of concurrent computation in distributed systems*.

PhD thesis, MIT, 1985. URL <http://hdl.handle.net/1721.1/6952>.

Erman Ayday, Fred Collopy, Taehyun Hwang, Glenn Parry, Justo Karell, James Kingston, Irene Ng, Aiden Owens, Brian Ray, Shirley Reynolds, Jenny Tran, Shawn Yeager, Youngjin Yoo, and Gretchen Young. Sharetrace: A smart privacy-preserving contact tracing solution by architectural design during an epidemic. Technical report, Case Western Reserve University, 2020. URL <https://github.com/cwru-xlab/sharetrace-papers/blob/main/sharetace-whitepaper.pdf>.

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

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

- Andrea Capponi, Claudio Fiandrino, Burak Kantarci, Luca Foschini, Dzmitry Kliazovich, and Pascal Bouvry. A survey on mobile crowdsensing systems: Challenges, solutions, and opportunities. *IEEE Commun. Surv. Tut.*, 21(3): 2419–2465, 2019. doi:10.1109/COMST.2019.2914030.
- William Clinger. *Foundations of actor semantics*. PhD thesis, MIT, 1981. URL <http://hdl.handle.net/1721.1/6935>.
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, fourth edition, 2022.
- Centers for Disease Control and Prevention. Quarantine and isolation, 2021. <https://www.cdc.gov/coronavirus/2019-ncov/your-health/quarantine-isolation.html>.
- Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley, 1995.
- Raghu K. Ganti, Fan Ye, and Hui Lei. Mobile crowdsensing: current state and future challenges. *IEEE Commun. Mag.*, 49(11):32–39, 2011. doi:10.1109/MCOM.2011.6069707.
- Bin Guo, Zhu Wang, Zhiwen Yu, Yu Wang, Neil Y. Yen, Runhe Huang, and Xingshe Zhou. Mobile crowd sensing and computing: The review of an emerging human-powered sensing paradigm. *ACM Comput. Surv.*, 48(1): 1–31, 2015. doi:10.1145/2794400.

- Carl Hewitt. Viewing control structures as patterns of passing messages. *Artificial Intelligence*, 8(3):323–364, 1977. doi:10.1016/0004-3702(77)90033-9.
- Carl Hewitt and Henry Baker. Laws for communicating parallel processes. Working paper, MIT Artificial Intelligence Laboratory, 1977. URL <http://hdl.handle.net/1721.1/41962>.
- 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, IJCAI’73*, pages 235–245, 1973. URL <https://dl.acm.org/doi/10.5555/1624775.1624804>.
- Frank R. Kschischang, Brendan J. Frey, and Hans A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans. Inf. Theory*, 47, 2001. doi:10.1109/18.910572.
- Nicholas D. Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, and Andrew T. Campbell. A survey of mobile phone sensing. *IEEE Commun. Mag.*, 48(9):140–150, 2010. doi:10.1109/MCOM.2010.5560598.
- Huadong Ma, Dong Zhao, and Peiyan Yuan. Opportunities in mobile crowd sensing. *IEEE Commun. Mag.*, 52(8):29–35, 2014. doi:10.1109/MCOM.2014.6871666.
- Cristina Menni, Ana M Valdes, Maxim B Freidin, Carole H Sudre, Long H Nguyen, David A Drew, Sajaysurya Ganesh, Thomas Varsavsky, M Jorge Cardoso, Julia S El-Sayed Moustafa, Alessia Visconti, Pirro Hysi, Ruth

C E Bowyer, Massimo Mangino, Mario Falchi, Jonathan Wolf, Sebastien Ourselin, Andrew T Chan, Claire J Steves, and Tim D Spector. Real-time tracking of self-reported symptoms to predict potential COVID-19. *Nat. Med.*, 26, 2020. doi:10.1038/s41591-020-0916-2.