Contact tracing infection risk estimate

Distributed simulation approach

ViraTrace

April 2020, v1.0

This document outlines the distributed data infection spread model developed by ViraTrace, and is divided into two parts. First part outlines the current state of contact tracing applications and their main deficiencies. Second part covers in detail how the model is implemented and what benefits it brings to the standard contact tracing.

1 Contact tracing

In light of the COVID-19 situation, there have been a number of similar contact tracing mobile applications in development across the globe, such as TraceTogether in Singapore, PrivateKit at MIT, COVID Watch, NOVID20, etc. Main goal of these solutions is to record all close-proximity interactions between users and, when one of the users is identified as infected, notify all their recent contacts about possible infection risk. Such approach is believed to slow down the spread of infection, and reduce the resources needed for "classical" manual contact tracing done by epidemiologists.

Because such interaction records consist of sensitive personal data and could pose a privacy risk if leaked, most of the development has been focused on distributed storage systems, mainly by storing user interactions only on their own devices. With such distributed implementations, commonly based on bluetooth communication, workflow usually goes as follows:

1. After installing the app, the user agrees to allow bluetooth being used to transmit anonymized IDs.

2. When another device with the app installed is detected by the bluetooth in close proximity, a record of the interaction is stored locally, containing time of interaction and the detected ID of the other device.

3. If a user tests positive for an infection, through some authorization mechanism (depending on the local authority practices), this information is passed on to all the devices which were in contact with their device since the estimated time of infection (e.g. the incubation period).

For a more in-depth review of an example of such a solution, please refer to the COVID Watch Whitepaper which describes all the implementation details and levels of privacy that can be achieved.

1.1 Network analysis on distributed data

Privacy gained by implementing contact tracing in a distributed fashion is always paid for by the significant restrictions on the range of possible data explorations and modelings, especially with regards to social network analysis. Because such analysis is very important in epidemiological research, and because a lot of additional valuable information is encoded in the network topology, it remains an open question how best to approach such distributed analysis. This document outlines one possible approach to this problem.

1.2 Multiple-step risk propagation

Apart from the reduced network analysis capabilities which are present with all distributed implementations, there are additional deficiencies with current implementations. One such deficiency is the binary output they produce in terms of infection status: each user is either notified as a direct contact of the infected, or is not notified at all. One way to move away from this is to use additional interaction data as "weight" of an interaction, to be used at a later point when deciding which alert level to use with which contact of the infected. For example, duration of the interaction and distance can be used, with longer and closer interactions with the infected resulting in higher-priority alert.

Another, even more crucial deficiency, is the fact that only the first contacts of the infected are alerted, with their subsequent contacts being unaware of their risk increase. In other words, if Alice meets Bob, and Bob at a later point meets Carol, and if Alice tests positive for the virus, only Bob is notified, although Carol is at significant risk too. Even more so, if a family of five meets Alice, and at a later point meets Carol, Carol is at significantly higher risk of being infected than either Bob or any member of the family. However, she would still not be notified about that risk.

This problem is even more pronounced if a high percentage of the infected are asymptomatic. These individuals would, in most cases, not be registered as infected, and their contacts would not be notified about the risk, even though we might believe them (the asymptomatic infected individual) to be infected based on their contacts. Considering all these problems: lack of advanced network analysis, binary risk alerts and single-step risk propagation, we propose a solution which:

1. Proves that it is possible to build advanced models even on distributed data, based on emergent network properties.

2. Creates a continuous risk assessment which enables multiple-level prioritization of the possibly infected individuals, e.g. testing for the top 5%, isolation for the top 10%, and "soft alerts" for the top 20% of users.

3. Propagates the risk through the network, not limiting the risk assessment only to the direct contacts of the reported infected individuals.

2 Distributed infection risk estimation

2.1 Data

There are two components to the data on which our model operates:

1. Temporal network of interactions, stored in a distributed fashion, with each node holding its adjacent edges

2. Infection status of each node, as reported by the user and confirmed by the authorities, also stored on the device itself (each person's status on their device only), with the estimated time of infection (e.g. based on the symptoms or some other indicators)

As already noted, interactions are stored based on the IDs broadcasted by the bluetooth of each device. Infection status is reported differently based on the country of deployment, local laws and practices, and additional deployment details. For example, some countries have centralized infection databases based on anonymized IDs known only to the infected individual, which can be leveraged by the app to update the current status (more info is available here).

2.2 Communication infrastructure

Apart from the locally available communication between devices, which mainly consists of short messages sent over low-energy bluetooth, our model also requires a messaging service to be available between devices to enable communication based on the anonymized IDs after the bluetooth connection is lost.

In other words, when interaction happens, devices exchange IDs, and at a later point these devices should be able to send messages to one another based on those IDs. Although a range of peer-to-peer mobile solutions exist, we believe that a centralized communication channel should provide more stability, without sacrificing privacy. Additionally, centralized server must be able to broadcast messages to all devices, which most implementations already support.

2.3 General idea

General concept behind our method relies on the idea of Monte Carlo simulations, which in essence tries to estimate some probability P of event E by simulating lots of "outcome" and counting how many times E happened. For example, to estimate the probability of getting a sum of 5 when rolling three dice, one might roll three dice a thousand times, count how many times the sum was 5, and estimate the probability by dividing that number with a thousand.

Obviously, to be able to perform such counting, we need to run the simulation. This section explains, in general, how simulations can be executed over distributed datasets we previously described.

Presume that our goal is to run a single realistic simulation of the infection spreading during the last N days. We start with a broadcasted message to all the devices saying "Starting simulation S with payload P". S is a unique simulation ID, and P holds different simulation parameters, such as transmission probability p. For now, we presume that these parameters are already known for the virus in question.

When they receive such broadcasted message, noninfected nodes do nothing. Infected nodes take their estimated time of infection, and for each stored interaction since that time they:

1. "Flip a coin" (generate a random bit) with transmission probability p
2. If "coin lands tails" (generated bit is 0), they do nothing, otherwise they send a "You are infected in S" message to the node on the other end of the interaction, alongside a "rough time" of the interaction t (e.g. a date).

When a node receives such a message for the first time, it repeats the same procedure for all the interactions that occurred after t, and stores t. If at a later point it receives the same message with some t' where $t' < t$, it does the same for all the interactions between t' and t. This approach requires us to store the time t of the "oldest" interaction over which we received a message. Finally, if a message is received for an interaction t'' with $t'' > t$, it is ignored, as all the messages were already sent for interactions since t.

After the simulation has ended, which either occurs after a predefined timeout, or after no new messages have passed over a server w.r.t S and server broadcasts the "Simulation S ended" message, each node is able to determine whether they were infected in this simulation or not (depending on whether it received any messages or not).

It is important to note that there is an obvious discrepancy between these simulations and the real world in terms of "recovery" - our simulations do not model a standard SIR model, as nodes stay infectious forever, and spread the infection through all the interactions that occurred after they received a simulated infection. In practice, we don't try to solve this problem, as we only simulate a couple of days of interactions during which we can presume none of the currently infected nodes, as well as none of the "simulation-infected" nodes, were recovered. However, if it is a serious issue for any reason, it can be solved by defining the recovery period T_r and forwarding the information only through interactions which happened between t and $t + T_r$. In that case, however, a following scenario is possible, presuming $T_r = 5$:

1. Node B receives a message from node A_1 based on interaction at time $t = 10$.
2. Node B propagates the infection further over to C based on interaction at time $t = 12$.
3. Node B receives a message from node A_2 based on interaction at time $t = 5$.

At this point, node B would "learn" that at time $t = 12$ it was in fact recovered, and a message sent to C was false. For B to be able to inform C about this, B would need to log all the messages they sent. For C to be able to use this new piece of information (that B did not, in fact, infect them in this simulation), C would need a complete log of all the incoming messages too. This complicates the system significantly, and such case will not be further discussed in this document, but should be a topic of future work.

2.4 Parallel simulations

To successfully estimate the risk of infection, we have to run a large number of simulations. However, running them sequentially would be infeasible since each such run would, in practice, take hours, taking into account the network latency and the fact that nodes are not online consistently. Instead, we run the simulations in parallel.

We make an additional presumption that the interaction network can be viewed as discretely temporal, and we refer to the single period of each network "snapshot" as a "day", although it can be thought of as any predefined time period. In other words, we presume that during a single "day", infection can only pass through a single step of interactions, as newly infected individuals have to first pass through the latent phase during which they are not infectious themselves. This means that the state of each node does not change continuously, but in discrete time steps (e.g. days), and that the collective "inputs" of all time steps up to, and including, some t_i result in the "output" of that node at t_{i+1}. In other words, to be infectious at time t_j, node must have been infected at some time t_i with $i < j$, where i and j are discrete time periods. This further means that the "evolution" of the node's state during time is fully described by it's infection state on each "day" of the simulation.

To implemented parallel simulations, we make the following changes to our simple model:

- Instead of states being described by a single bit, they are described by a binary vector V. Each element of this vector V_i corresponds to the state of the node at i-th simulation.
- Infected nodes start with a one-vector [1...1], while noninfected nodes start with zero-vector [0...0] for each simulated day.
- Messages passed from one node to another contain such binary vectors.
- When new vector V_i is received for some day i, all the vectors for days $j, j > i$ are updated by setting them to $V_i \vee V_j$ where $\vee$ is the bitwise OR operator. All of the updated vectors are repropagated further through the network.
- When some vector is propagated through the interaction I_k, node first generates a random binary mask (filter) where each bit is 1 with probability p. This mask is applied to its current state vector (by AND operation) and the result is sent to the other node. This mask must be stored locally for each interaction during the simulations, as future "repropagations" of vectors over the same interaction

must use the same mask. Otherwise, it would be possible for some vector bit V_i to change from 1 in the first message to 0 in the second one, which would require nodes to store the whole "incoming" history to be able to apply such changes. In other words, we require all of the state-vectors to only be "additively" changed, i.e. their sum (total number of "ones") can only increase while the simulation is running, making it possible to simply combine the previous result with the new input using the OR operator.

When the simulation ends, each node is able to asses its own risk by summing up the final (last day) state vector. If this sum is above some threshold, user receives a notification of some alert level, e.g. test prioritization, self-isolation, etc.

2.4.1 Example

Let us describe a simple scenario of information propagation using the example shown in figure 1. Presume the interactions occurred from left to right, with each set of parallel interactions occurring over a single day. Simulation runs as follows:

1. Simulation start is broadcasted by the server, and the infected (red) node initializes a one-vector.

2. Infected node generates a mask for the interaction with A, applies it to the initial one-vector and sends the result to A. Generated mask is stored as was already discussed, but in this example is never reused.

3. Upon receiving the vector from the infected node, A looks at all the interactions which happened since the day after the interaction with the infected node, which includes all four of the $A-B_i$ interactions. It generates a mask for each interaction and passes on the vector to each of the B_i nodes.

4. Each of the B_i nodes proceeds in the same manner, sending their vectors b_i to node C.

5. C finally combines all the incoming vectors as $b_1 \vee b_2 \vee b_3 \vee b_4$.

For simplicity, below each node we only define the vector corresponding to the day after the incoming interaction occurred. For example, node A starts with [00000] on day 1 when it gets in contact with the infected node. Then on day 2 its vector is updated to [10110] and stays the same for the rest of the simulation. Note that "on day two" is not a time reference (all days are simultaneously simulated), but refers to one of the state-vectors which describe this node's evolution trough time. In other words, when running 5 parallel simulations for 6 days, each node's full state evolution is described with 6 vectors of 5 bits each. For each of the interactions, nodes send over the vector which corresponds to their state on that particular day when the interaction occurred, masked with the random binary mask which models the probabilistic nature of the virus transmission.

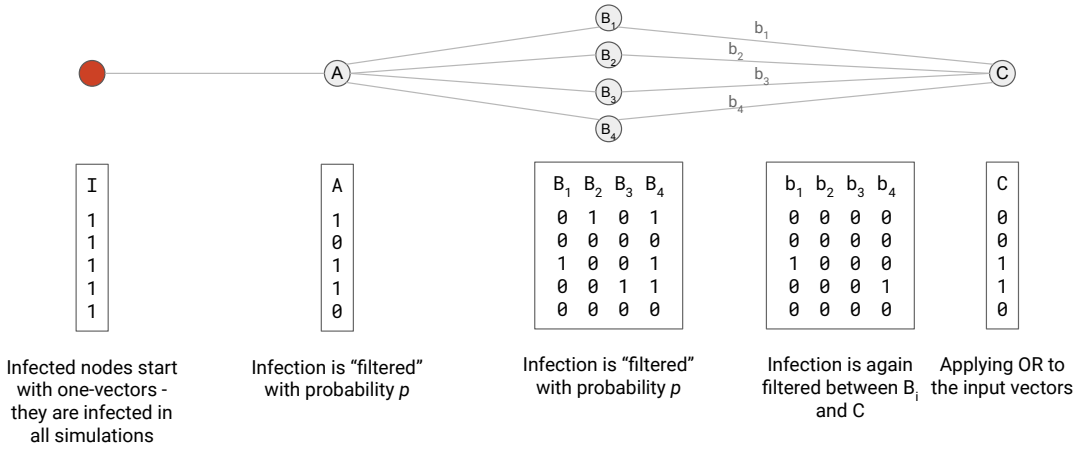

Figure 1: Simple example of state-vector propagation over proximity interactions

2.4.2 Why simulation instead of probability propagation?

Some readers might ask why we need to run simulations, when instead we could just propagate the infection probabilities. For example, infected node would have a probability of infection of 1, their nearest neighbors probability of p (the transmission probability), next step p^2, and so on. If some node is connected to two nodes with probabilities of infection of p_1 and p_2, its combined probability of infection would be $pp_1 + (1 - pp_1)pp_2$. This approach, however, would not work as we would need to calculate a joint probability for each subset of nodes being infected. Example in figure 2 shows how the calculated probability of node C being infected is greater than the calculated probability of node A being infected. However, infection of node A is a prerequisite to the infection of node C, meaning that the probability of the infection of A must be higher or equal to that of node C.

2.4.3 Parallel simulations – a synchronous approach

In our discussions so far, we have focused on the high-latency, low-node-availability communication scenario, in which nodes only sporadically come online and their communication is slow. This makes it very important for the whole simulation to be asynchronous, as more active network components shouldn't be impeded by the less active ones. However, there is a different, synchronous approach which removes the need for any local storage of simulation-specific data (such as mask vectors, output vectors, etc.).

Using again the presumption of discrete time, server might coordinate the simulation

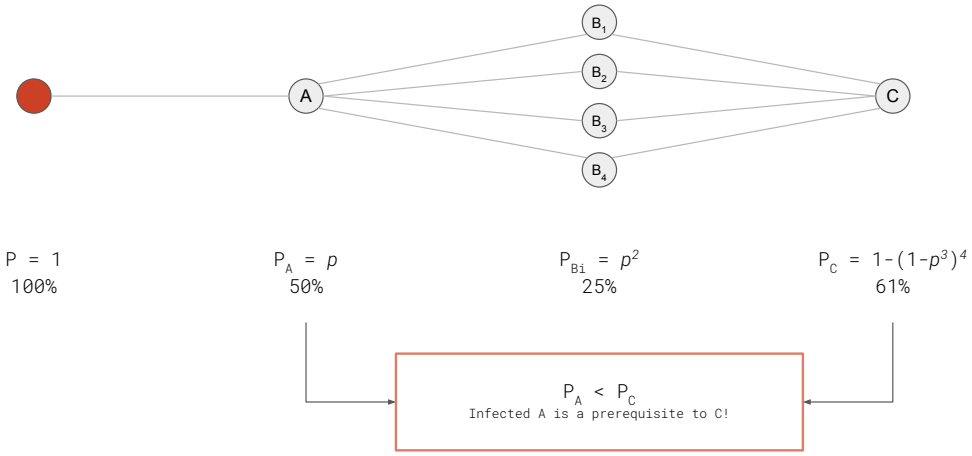

Figure 2: Example of probability propagation leading to a contradiction

to run on a "per-day" basis. When the initial "Simulation starting" broadcast is sent, it must contain a date with which the simulation starts. All of the nodes then share the vectors in the usual way only for that day. That means that there is a complete distinction between input and output information for each node: inputs are stored and finally combined with the OR operator, while outputs are previously stored state vectors, either the initial ones on day zero (one-vector for infected, and zero-vector for noninfected nodes) or the resulting input combination from the previous day. Instead of listening on the messages to determine when to stop the whole simulation, server now listens to determine when to move it to the next day.

This approach becomes even more important when privacy is of major concern, and should be the topic of future work.

2.5 Choosing infection parameters

We previously presumed that we already know the infection parameters for the simulation, mainly the transmission probability p. Note that, when talking about this probability, we might as well be talking about some function $p(I_k)$ which determines this transmission probability based on the interaction attributes: duration, closeness, temperature, etc. For simplicity, we presume these probabilities are constant, i.e. $p(I_k) = p$.

Results section of this paper shows that, because of the clustered and sparse nature of the proximity networks, risk-ranking of the nodes does not change significantly with different transmission probabilities. In other words, although we cannot estimate the

exact infection probability without knowing exact transmission probabilities, we can still successfully rank the nodes based on that risk. With such ranking, we can produce different action-prioritization levels, such as testing for the top 5%, isolation for the top 10%, and "soft alerts" for the top 20% of users.

With this in mind, the problem of choosing the infection parameters changes from trying to match them as close as possible to the real world, to trying to "tune" them so that the best gradation of risk throughout the network is achieved. In other words, if transmission probabilities are too high, most of the network will get infected in each simulation, making it hard to differentiate between high-risk nodes. On the other hand, if transmission probabilities are too low, too many simulations would need to be performed in order to estimate the risk of lower-risk nodes.

2.6 Results

Results are currently not included in this document, but are shared as a supplement:

- As a Jupyter notebook here,
- As a static HTML file here.

All of the presented results, unless otherwise stated, were generated in the following way:

1. Static network G is either randomly generated or a real static network is used.
2. For each edge e of G a frequency $0 < f_e < 1$ is drawn from some predefined distribution P_f.
3. For each "day" (an abstract unit of time, as discussed previously) a subset of edges E_i is drawn based on the edge frequencies. This is a "daily snapshot" of the network, as discussed in the section 2.4.3.
4. An initial set of infected nodes is determined.
5. For each "day", infection is transmitted over each edge in E_i with probability p, and for each newly infected node a recovery time is drawn from some distribution (usually normal).
6. Nodes due for recovery on days i are set to the recovered state.

These simulations are repeated N times for D days (in most cases $N = 1000$, $D = 5$) in a manner similar to the binary-vector approach already described, resulting in N different outcomes of infection transmission. Each one of the simulations is then, iteratively, used as a "ground truth" with other $N - 1$ simulations being used as the basis for risk estimates. Each of these N runs produces a ranking of nodes based on their risk estimates, and a threshold T can be chosen to determine what percent of the nodes to "notify" as infected. Varying this threshold produces a ROC curve which we report.

Additionally, because we believe it is unquestionable that all the direct contacts with the known infected individuals should be notified (a.k.a the standard contact tracing approach), we removed all such nodes from our results. This means that the reported results only count those nodes which were not in direct contact with known infected individuals, which further means that the performance of the model is purely an "additive" performance on top of the standard approach. Further work needs to be done to determine how well this approach models risk differences between the direct-infection-contact nodes.

3 Conclusion

We believe the results we have gathered so far set a firm foundation for future work on this model and present a good argument for inclusion of such a model in a wide range of contact tracing solutions. We also believe that these positive results prove that contact tracing with distributed data storage does not necessarily prohibit any kind of social network modelling, and definitely does not restrain neither users nor epidemiologists to only a basic set of functionalities.

With this whitepaper we hope to start a discussion in the contact tracing community, and stand open to any and all suggestions, discussions and error reports.