Complex Networks

Final Project

SI simulation in a temporal network

Denaldo Lapi and Samy Chouiti

May 29, 2022

Contents

List of Figures

List of Tables

1 Introduction

1.1 Goals

The purpose of this work is to perform a Susceptible-Infected (SI) simulation on top of a temporal mail exchange network, containing mail messages exchanged between users at specific *Unix* timestamps[1].

The goal of the project is to explore the dynamics and properties of the SI simulation by using a real temporal data by:

- analyzing the characteristics of the infection prevalence curves
- analyzing the role of the network nodes in the virus spreading
- analyze the possibility and the importance of immunizing specific nodes
- analyzing nodes infection times in relation to several parameters
- analyzing roles of the infection probability on the SI simulation

The implementation will be performed in the following order:

- at first, we'll perform a detailed exploratory analysis of the considered network, in order to fully understand its characteristics and main properties. We'll visualize plots of the networks and of its distribution, and also of the main networks centrality measures.
- then, we'll implement the SI simulation process and we'll run our experiments one by one.

We would like to remark that all the functions were implemented in *Python* [2], by building a notebook and by using two auxiliary *Python* files with definitions of helper functions for performing plots and some simulations.

1.2 Context

In order to draw contextual conclusions about our findings, we will consider the infection of a computer worm over the whole network and thus see what strategy a malicious hacker could use in order to ensure maximum infection.

[1]Number of seconds since 1/1/1970

[2]*Python documentation*

1.3 Contents of the delivery folder

In particular, the final project folder contains the following files:

- *notebook.ipynb*: a *Jupyter Notebook* with the descriptions of all the steps and results of our analysis
- *simulation_utils.py*: a *Python* file containing auxiliary functions for performing the SI simulation and also for reading the network according to specific criteria
- *network_utils.ipynb*: a *Python* file containing the functions that we used to plot the network's degree distribution
- *report.pdf*: a *.pdf* file containing this report

We would also like to notice that we used the *NetworkX* module[3]of *Python* for threating and analyzing our complex network. An important remark ia that we used also the *igraph* [4] *Python* module which is implemented in *C* and allows faster computations for some of the network properties such as centralities.

This report outlines the technical details of all experiments and simulations performed, as well as the rationale for choosing such tests. The results of the performed simulations and experiments are discussed and reflected upon.

Finally some conclusions are drawn from the performed analysis and tests.

1.4 Notations used

For the rest of the paper, we will the following notations:

- ρ: proportion of infected nodes in the network
- p: for an infected node, probability of infecting its adjacent nodes

[3]*NetworkX documentation*
[4]*igraph documentation*

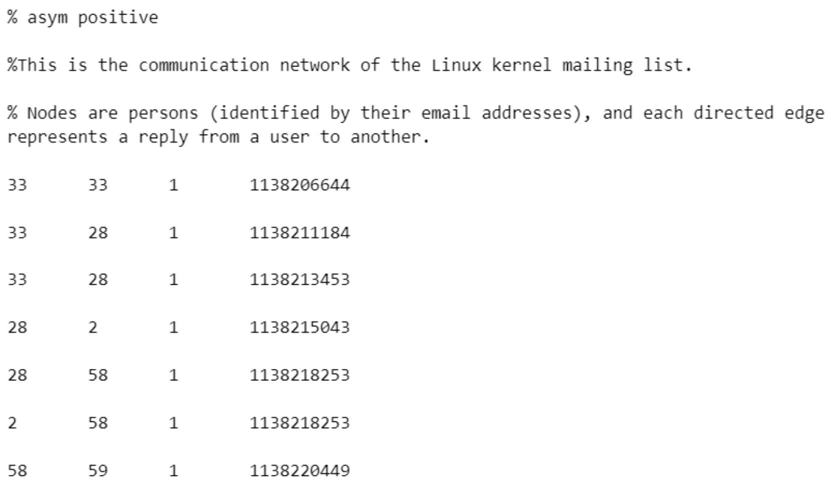
2 Network analysis

The first step of our work was to understand the temporal network we used for testing the SI simulation.

The analyzed network was taken from: `https://networkrepository.com/comm-linux-kernel-reply.php`[5] and, as it can be seen from the website, it represents a *Dynamic Network*.

2.1 Description of the network

The download network file from the above mentioned website [5] was a *.EDGES* file, containing the list of the edges between the nodes of the network. We therefore explored this file to better understand the structure of the network, and we report in Figure 1 a short part of it.



```
% asym positive

%This is the communication network of the Linux kernel mailing list.

% Nodes are persons (identified by their email addresses), and each directed edge
represents a reply from a user to another.

33      33      1       1138206644

33      28      1       1138211184

33      28      1       1138213453

28      2       1       1138215043

28      58      1       1138218253

2       58      1       1138218253

58      59      1       1138220449
```

Figure 1: First lines of the network file.

The above Figure shows that the file contains the edges of a communication network of the *Linux kernel mailing list*.

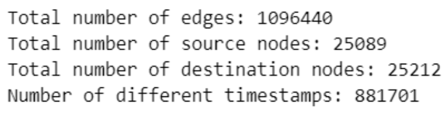
In particular, nodes are persons (identified by their email addresses), and each directed edge represents a reply from a user to another. The last column represents the timestamp of the mail exchange between two users.

Therefore, the network of interest is a *temporal* and *directed* network that represents the exchange of mails over time between nodes that represent persons.

Notice that the third column of the file represents the weight of each edge.

[5]Ryan A. Rossi and Nesreen K. Ahmed. *https://networkrepository.com/*, 2015.

As a next step, we analyzed separately source nodes and destination nodes, together with the timestamps, the main statistics for that are shown in Figure 2.



```
Total number of edges: 1096440
Total number of source nodes: 25089
Total number of destination nodes: 25212
Number of different timestamps: 881701
```

Figure 2: Nodes and timestamps

The range of node ids in the network goes from 1 to 63399, while the number of total nodes, as shown in the above figure is of 27927. We therefore concluded some identifiers are simply not present.

An interesting aspect we then analyzed was the range of timestamps and their distribution. Indeed timestamps are provided in a *Unix* format, which makes them less readable and interpretable. We understood that the network contains email exchanged during 8 full years, from 2006 to 2013, between 27927 users/mails.

While the distribution of the timestamps is shown in Figure 3

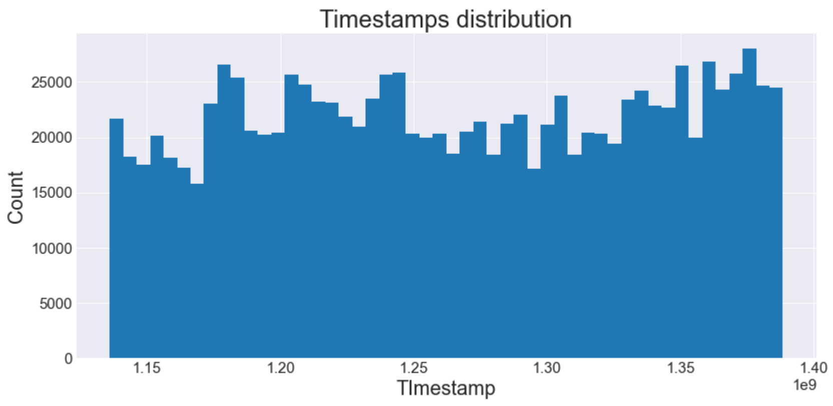


Figure 3: Timestamps distribution

2.2 Reading the network

We then proceeded to read the network as a *NetworkX* graph object.

We tried several possibilities:

- Reading the network by using the *NetworkX read_edgelist()* function, which allows to directly read a *.EDGES* file. We added each edge to the network with an attribute *weight* and an attribute *timestamp* encoding the timestamp of the link between 2 nodes. What we obtained at the end was a directed network with multiple edges.

- Read the network as static, without taking into account the timestamps. In that case, if a directed edge appeared into 2 different timestamps, we increased the weight by one, without considering the case of multiple edges. What we obtained in this case was a directed network, without multiple edges. The function for reading the network file in this way is the *read_network_static_singular()* defined in the *simulation_utils.py* file.

2.3 Network properties

As a next step of our analysis we checked the connectivity of the network by looking at 2 different types:

- *strong connectivity*: which states that a directed graph is strongly connected if and only if every vertex in the graph is reachable from every other vertex
- *weak connectivity*: which states that a directed graph is connected when the direction of the edge between nodes is ignored

We performed both the checking by using the static directed network defined before, without multiple edges (it reduces a lot the computational time) and by using the already defined *NetworkX* functions. We understood that the graph is not strongly connected nor weakly connected.

We therefore extracted the biggest weakly connected sub-graph and the biggest strongly connected sub-graphs by using the corresponding *NetworkX* functions:

- the biggest weakly connected sub-graph is composed by 24567 nodes and 239825 edges
- the biggest strongly connected sub-graph is composed by 18531 nodes and 230721 edges (obviously a lower number, being strong connectivity a stronger requirement)

By taking into account the **biggest weakly connected sub-component**, which is the network we will use for later computations and interpretations, we calculated some of the most important network descriptors, by using both *NetworkX* and *Igraph*. The results of the computation are shown in Table 1.

2.4 Degree distribution

Our analysis of the network proceeded by considering the degree distribution of the network nodes. We used some utility functions implemented in the *network_utils.py* for plotting the histograms of the probability density function (PDF) and of the complementary cumulative degree distribution (CCDF). We performed the plots

Nodes	24567
Edges	239825
Min. degree	1
Max. degree	4918
Avg. degree	19.52
Avg. clustering coeff.	0.1056
Assoritativity	-0.1827
Avg. path length	infinite
Diameter	112

Table 1: Original network properties. The infinite Avg. path length is due to small non-weakly connected components in the original network.

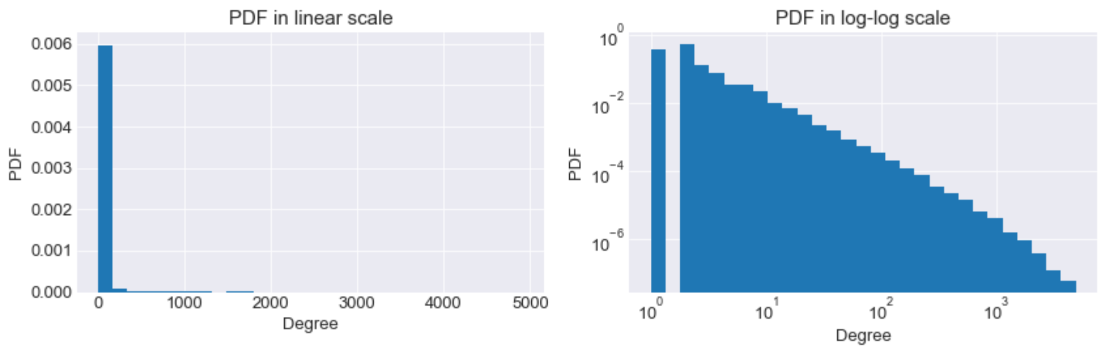


Figure 4: PDF plot

and all the rest of the analysis by considering the biggest weakly component subgraph, which allowed to simplify a lot the computational time, without loss of generality, considering the huge size of the network. The obtained histograms are reported in Figures 4 and 5.

From the histograms we can clearly visualize as the nodes degree distribution follows a power-law distribution pattern, with a very few nodes characterized by high degree values; while the majority of the network nodes has very low degree values. We can therefore conclude that the analyzed network is a *scale-free* network [6].

Basically the network has a nodes degree distribution that is described by the equation (1):

$$P(k) \sim k^{-\lambda} \tag{1}$$

We therefore decided to estimate the exponent of the empirical degree distribution by using the maximum likelihood estimation approach (MLE), described

[6] *scale-free network definition*

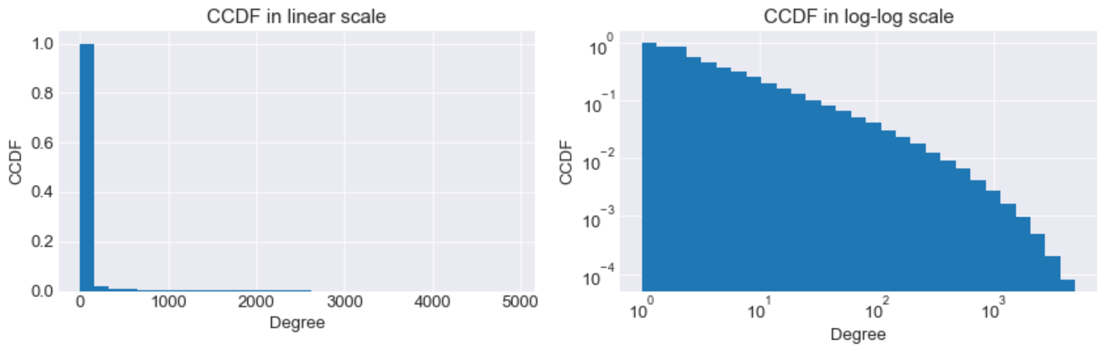


Figure 5: CCDF plot

in the paper “Power-law distributions in empirical data”[7].

This algorithm is already implemented in the *powerlaw Python* package, described in the paper “Powerlaw: a Python package for analysis of heavy-tailed distributions”[8]

In the provided Jupyter notebook *notebook.ipynb* are reported all the details of the used function, together with the calculation of the exponent.

After performing the exponent estimation, we decided to plot the nodes degree distribution in linear scale and in log-log scale, showing how the plot gives a straight line in log-log, since it follow the equation(1), where the exponent γ is the one calculated with the MLE approach. The function used to perform the plots is defined in the auxiliary file *network_utils.py*. The log-log plot shows also the theoretical line of the power-law distribution. The plots are reported in the Figure 6.

2.5 Centrality measures

As a following step of our network analysis, we analyzed the main centrality measures of the nodes, that will be used to interpret the results of some of the SI simulations we performed. We calculated the following centrality measures for each node of the network:

- *Degree centrality*
- *Betweenness centrality*

[7]Clauset, Aaron, Cosma Rohilla Shalizi, and Mark EJ Newman. “Power-law distributions in empirical data.” SIAM review 51.4 (2009): 661-703.

[8]Alstott, Jeff, Ed Bullmore, and Dietmar Plenz. “powerlaw: a Python package for analysis of heavy-tailed distributions.” PloS one 9.1 (2014): e85777.

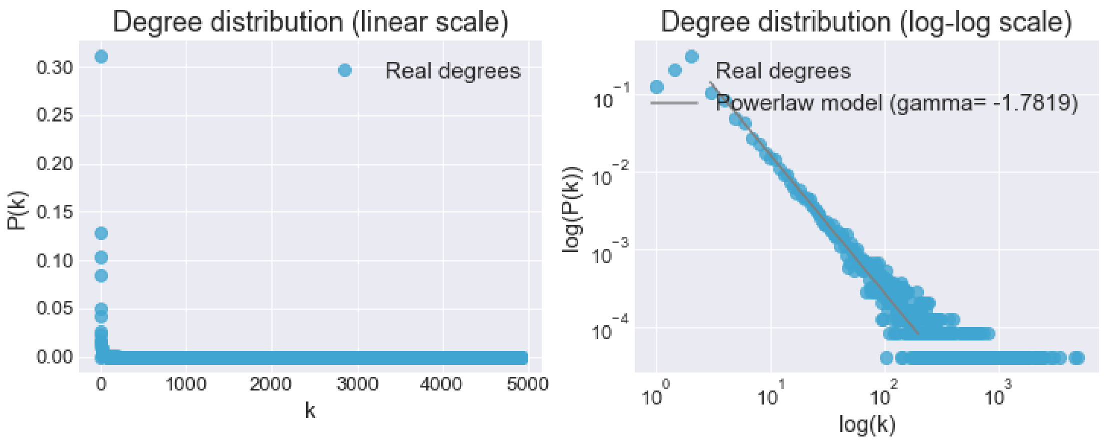


Figure 6: Distribution plot

- *Eigenvector centrality*

The calculations were performed using the *igraph* module, since they required too long with *NetworkX*. For each centrality we performed also a *scatter plot* reporting the centrality values for each node. The plots are reported in Figures 7, 8, 9 and 10.

2.6 Network plot

After performing all the analyses we decided to visualize the network by showing a full plot of it. However, *networkX* and *igraph* are not the best way to visualize networks of big size such as the one we anlyzed. The website from which we downloaded the network[5], allows also to visualize some interesting plots of the network, by encoding through colors and shapes some network properties such as the degree or centralities. We therefore used this tool to visualize the full network, which is reported in the Figure 11

The plot colors nodes and edges according to the betweenness centrality values, while the size of nodes and edges is proportional to the degree. In the plot we can also see many not connected sub-graphs, which were not considered in our SI simulation, since they are nodes that do not get in contact with the others.

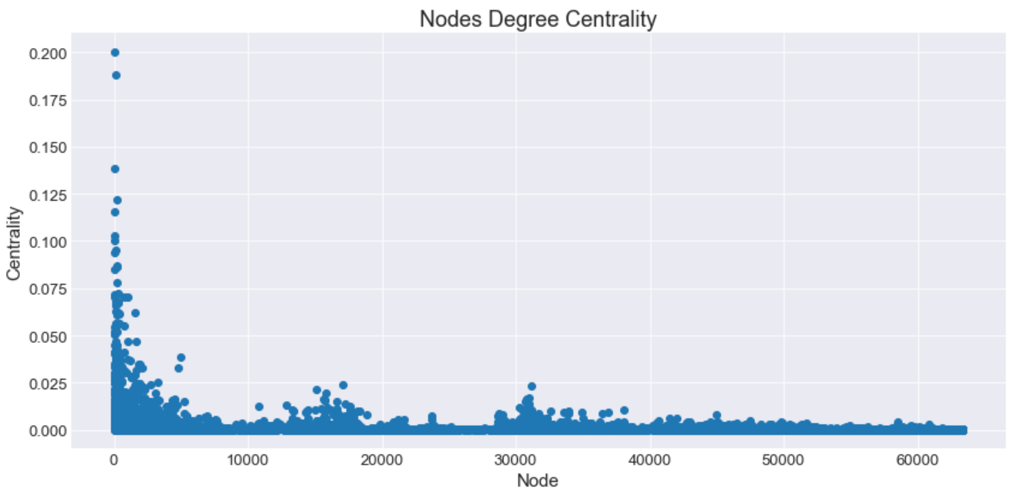


Figure 7: Degree centrality

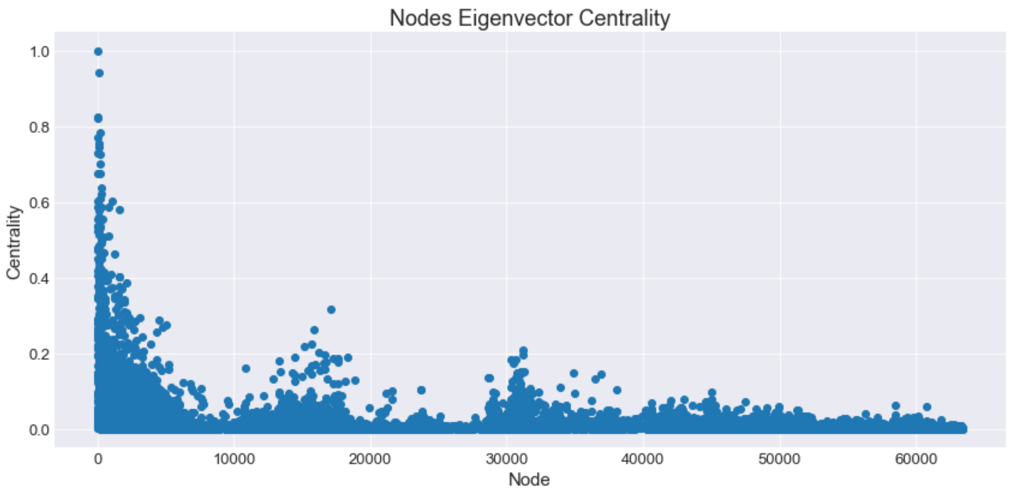


Figure 8: Eigenvector centrality

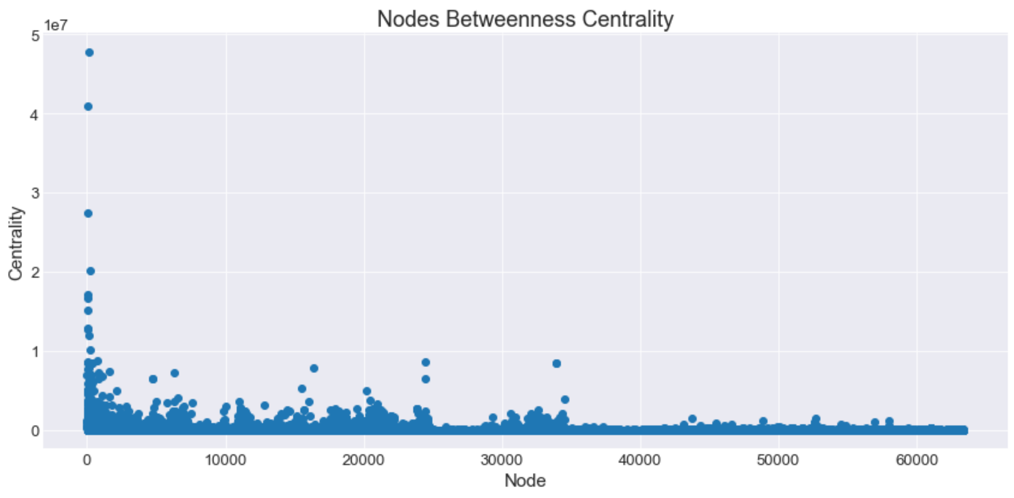


Figure 9: Betweeness centrality

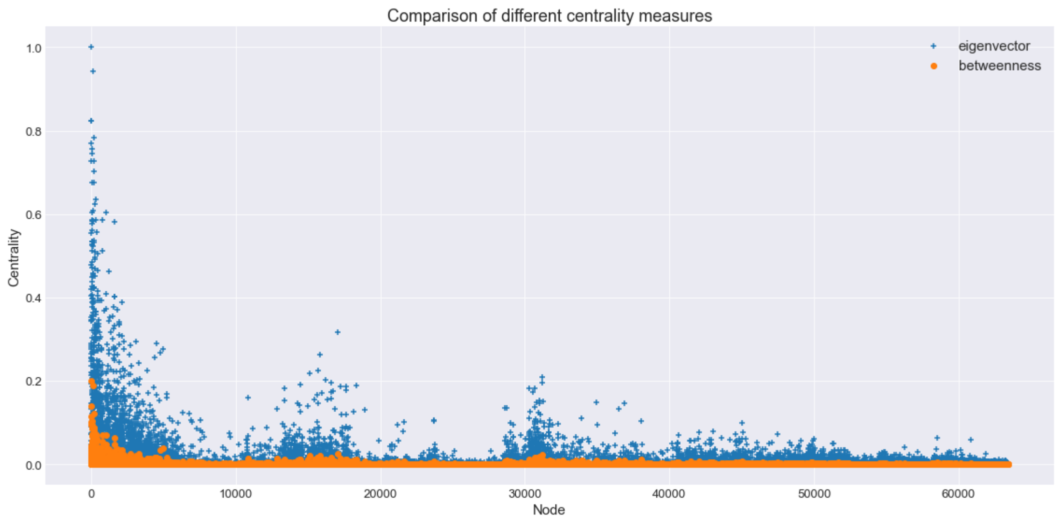


Figure 10: Comparison of centrality measures

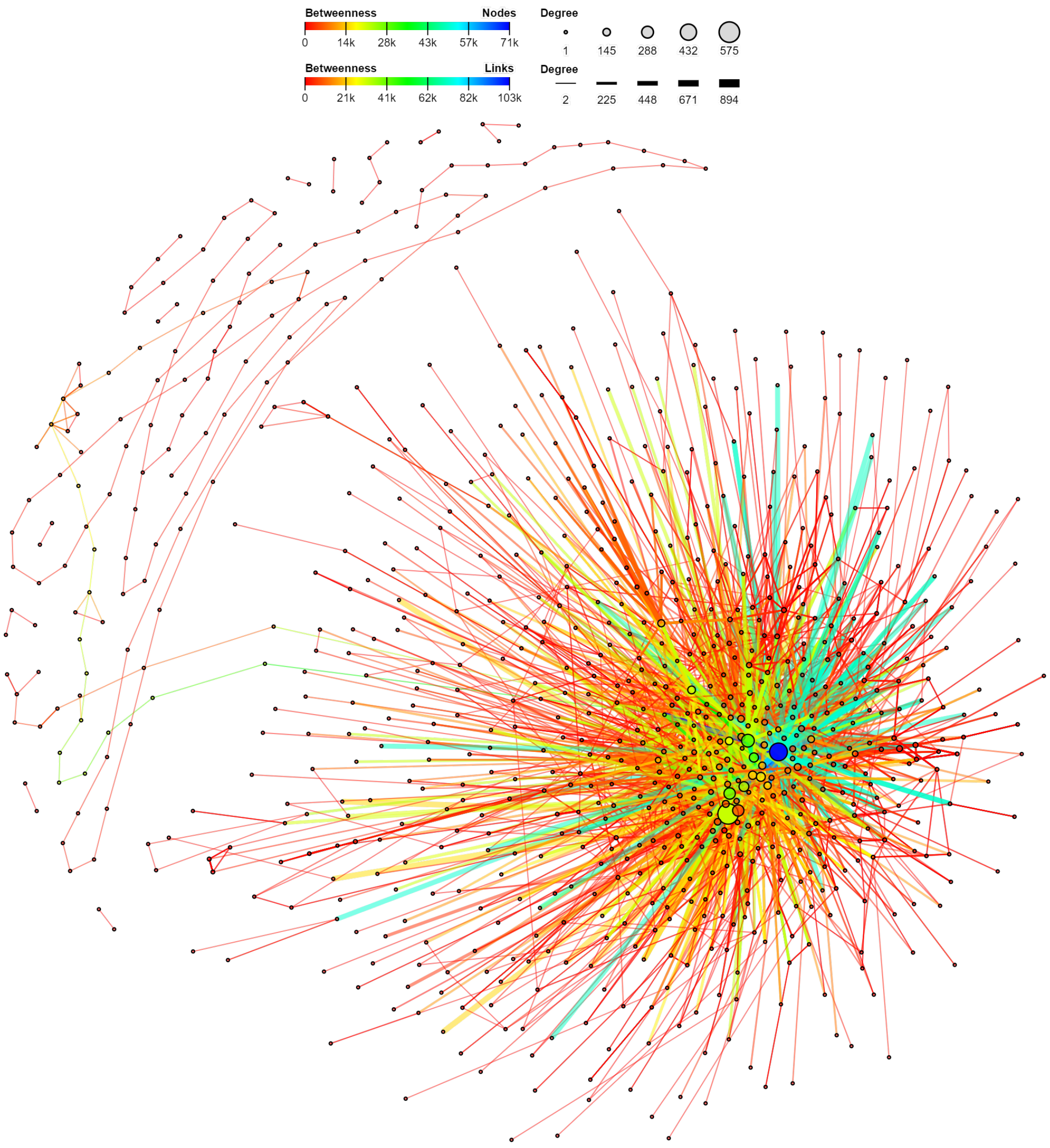


Figure 11: Plot of the network

3 SI Simulation Design

At this point, we were ready to implement the *Susceptible-Infected* (SI) epidemic spreading model on top of our temporal network, containing information about the timestamps of exchange of emails between users.

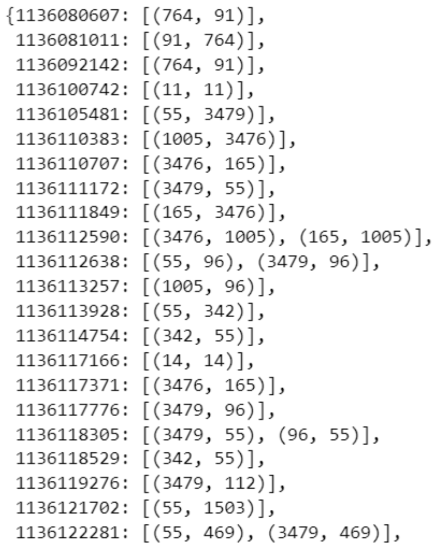
We performed a detailed study of the dynamics of the spreading and how it depends on where the process starts as well as on the infectivity of the virus.

We then used the network centrality measures (the ones we plotted before) to understand the roles played by specific nodes. This allowed us to analyze the speed of spreading and other dynamics by taking into account some specific nodes.

3.1 Data structure for performing the simulation

In order to simplify the SI simulation, we decided to read the network in a convenient way for our tasks.

In particular we defined a function (*read_network_temporal()*) in the *simulation_utils.py* file which reads the edges from our network file and builds a dictionary data structure where keys are represented by timestamps (ordered in increasing order), and each key points to the list of edges/contacts between pair of nodes that exchanged a mail message in that specific timestamp. Again, we took under consideration the nodes in the largest weakly connected sub-graph of the overall network. An example of entries of this data structure are shown in the Figure 12



```
{1136080607: [(764, 91)],
 1136081011: [(91, 764)],
 1136092142: [(764, 91)],
 1136100742: [(11, 11)],
 1136105481: [(55, 3479)],
 1136110383: [(1005, 3476)],
 1136110707: [(3476, 165)],
 1136111172: [(3479, 55)],
 1136111849: [(165, 3476)],
 1136112590: [(3476, 1005), (165, 1005)],
 1136112638: [(55, 96), (3479, 96)],
 1136113257: [(1005, 96)],
 1136113928: [(55, 342)],
 1136114754: [(342, 55)],
 1136117166: [(14, 14)],
 1136117371: [(3476, 165)],
 1136117776: [(3479, 96)],
 1136118305: [(3479, 55), (96, 55)],
 1136118529: [(342, 55)],
 1136119276: [(3479, 112)],
 1136121702: [(55, 1503)],
 1136122281: [(55, 469), (3479, 469)],
```

Figure 12: Dictionary data structure used to perform the SI simulations

As shown in the figure, we have that for each timestamp (key in the dictionary) is associated a list containing the links (as tuples) among nodes having a contact in that specific timestamp.

3.2 Susceptible-Infected spreading model

Once build the data structure to be used for running the SI simulation, we build the actual function implementing the SI model. In a SI model, each node can assume a state that is either:

- Susceptible (S)
- Infected (I)

When an infected node gets in contact with a susceptible node, this last one may become infected with a probability p in [0,1], which indicates the infectivity of the virus. Notice that in the case of our mail network we consider a spam mail setting: we suppose that an initial spam mail is sent by an initial infected person, and each receiver can get the infection if it opens that email, and this happens with a probability p.

An important characteristic about the SI model is that infected nodes remain infected forever.

In our specific temporal network, nodes are users (i.e. mails) and timestamped connections represent exchange of mails between them.

We'll take into consideration several hypotheses and perform different experiments.

- in particular, we'll consider an initial node (the *seed* node of the epidemic spreading) as the only infected node at the beginning, or we may also consider a group of initially infected nodes. Then, following the SI process, an email from an infected person (i.e. computer) may infect its susceptible destination with probability p in [0,1].

Infected nodes will remain infected for the rest of the simulation, and the simulation will be performed according to the provided timestamps, by following the order of exchanged messages.

3.2.1 Python implementation

The main function performing 1 full simulation of the SI model is defined in the file *simulation_utils.py*, and it is the function *simulate_SI()*, which takes as main inputs the dictionary data structure we showed in Figure 12, the list of seed nodes, the number of network nodes, the infection probability and the eventual list of immunized nodes.

The simulation is ran by building a dictionary which records the time of infection of each one of the nodes of the network: nodes are the keys and timestamps are the values, representing the infection time of the node.

At the beginning of the simulation the time of infection of the seed nodes is set to the smaller timestamp in the network (since they are infected from the very beginning), while the time of infection of the rest of the nodes is set to infinite, since they are not still infected.

We then perform an iteration over the ordered timestamps of our dictionary and we check the edges/contacts related to each timestamp. We infect a destination node of an edge if:

- the source node of the edge is already infected at the current timestamp
- the destination node is not already infected, i.e. its infection time is higher than the one of the source node
- the destination node is not an immunized node

If the above conditions are satisfied, we generate a random number between 0 and 1, and we compare it with the infection probability: in case it is lower than p we infect the destination node, i.e. we set its infection time to the current timestamp.

After each timestamp, we update the percentage of infected nodes of each timestamp (ρ).

At the end of the simulation we return $\rho(t)$, i.e. the list representing the evolution of ρ over the timestamps, and the dictionary containing the infection time of each node.

In order to reduce the variance due to the infection probability parameter, we repeated the simulation multiple times, and we averaged the ρ values of the different simulations: in order to do so we defined another function in the *simulation_utils.py* file performing the SI simulation multiple times and averaging the percentages of infected nodes over the various simulations.

After returning the prevalence $\rho(t)$, we plotted it by showing on the x-axes the time after the first event, i.e. the time after the first infection, starting from time 0. An example of prevalence plot is reported in Figure 13.

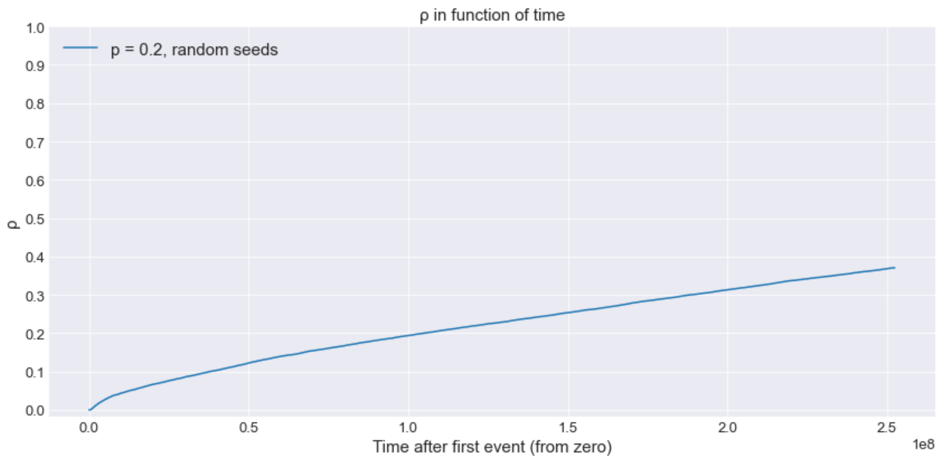


Figure 13: Example of prevalence plot

4 Experiments

We'll now explain the experiments and the results we obtained by running several SI simulations on the analyzed network.

In particular, we performed the following experiments, in order:

- Experiment 1: what happens if infection probability is 1
- Experiment 2: effect of the infection probability on the spreading dynamics
- Experiment 3: effect of the seed node on the spreading dynamics
- Experiment 4: find most protected nodes
- Experiment 5: predict nodes infection time by means of nodal properties
- Experiment 6: compare immunization strategies
- Experiment 7: virus spreading with and without immunized nodes

4.1 Experiment 1: $p = 1$

Goal: Study the simple case in which $p = 1$, i.e. the virus is always transmitted.

We performed the test for 5 random seed nodes and we plotted the combined plot. For each seed we ran the simulation 20 times and we obtained the plot with the averaged $\rho(t)$ over the 20 iterations reported in Figure 14.

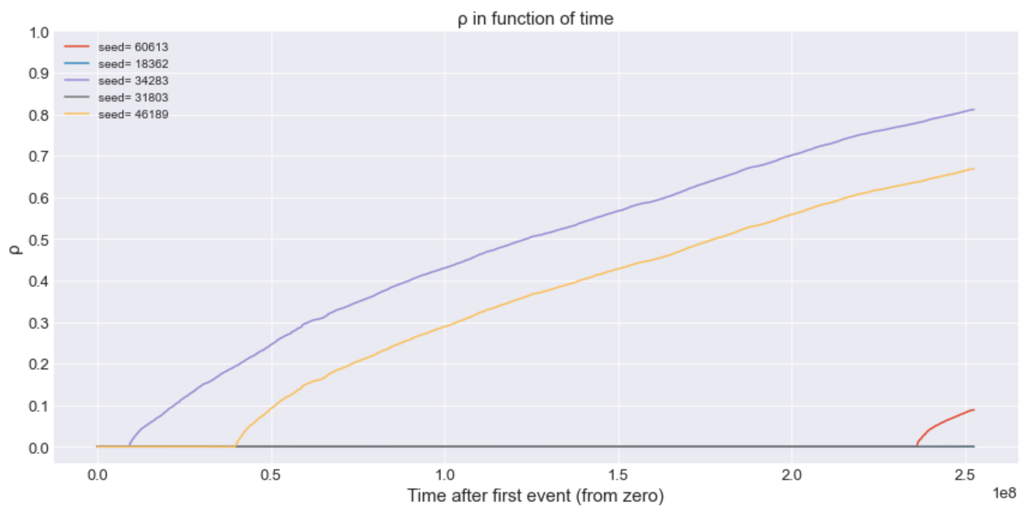


Figure 14: Evolution of $\rho(t)$ for $p = 1$

What we can see is that even with an high infection probability value it is very difficult to spread the virus along the nodes of the network: from the plot only 2 specific seed nodes allow to reach a prevalence value above 60%, i.e. with 60% of infected nodes among all the network nodes. For some of the seed nodes the infection starts very late, since this nodes start to contact the other nodes in the network after many timestamps. The fact that some curves reach low prevalence value is due to some seed nodes that do not contact too many neighbours and also because these neighbours may be pretty isolated.

Contextualization: Observation of the infection spreading over the whole network supposing that every person infects every contacts he made by mails. Could be a real situation in case a worm spreads automatically and no spreading mitigation strategies are in place in this community of people.

4.2 Experiment 2: Evolution of $\rho(t)$

Goal: Analyze the effect of the infection probability p on the spreading speed of the virus.

We ran our SI model 20 times for each different value of the infection probability and we used as seed node the node number 34 which is the one with highest degree centrality value.

We used the following list of infection probabilities: [0.01, 0.05, 0.1, 0.3, 0.5, 0.7, 1.0], and the resulting prevalence plot is reported in Figure 15.

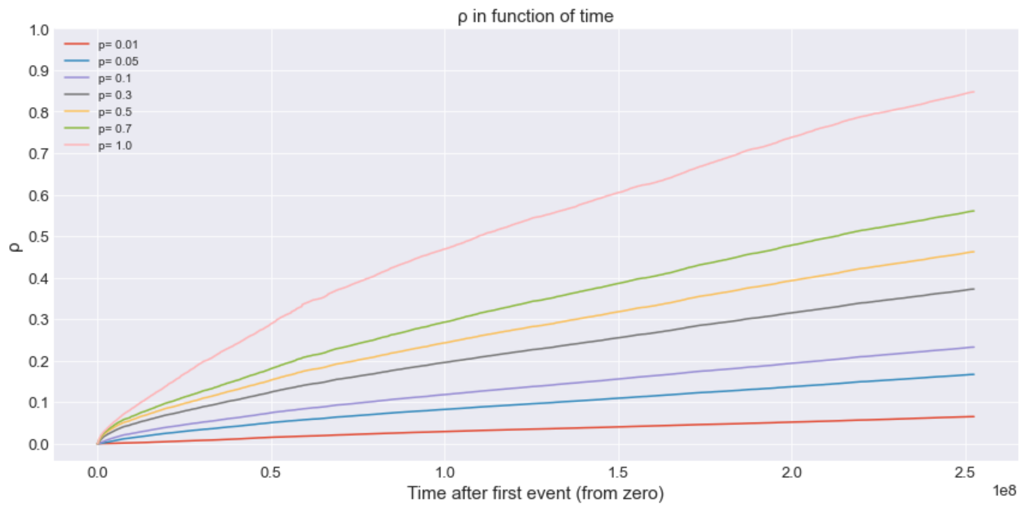


Figure 15: Evolution of $\rho(t)$ for different p values

The plot shows how an higher infection probability allows to reach an higher number of infected nodes, and also determines a faster speed in the infection propagation, as it can be seen in the very fist timestamps after the spreading, where the pink curve clearly grows very quickly. In particular, we can see that with $p = 1$ the $\rho(t)$ reach a value of almost 85% meaning that 85% of all the nodes of the network get infected.

Contextualization: Observation of the infection spreading over the whole network with different infection probabilities, i.e. each user has a certain probability of getting the virus, by using as seed node the most central one.

4.3 Experiment 3: Influence of seed nodes on $\rho(t)$

Goal: Study the influence of the seed node selection on the spreading dynamics.

More specifically, we ran the SI simulation using different seed nodes: 4 random nodes and one (node with id 34) that was the node with highest degree centrality. We ran the simulation averaging the results over 20 repetitions for each seed node, and by fixing $p = 0.1$. We then observed the evolution of infected probability $\rho(t)$ reported in Figure 16.

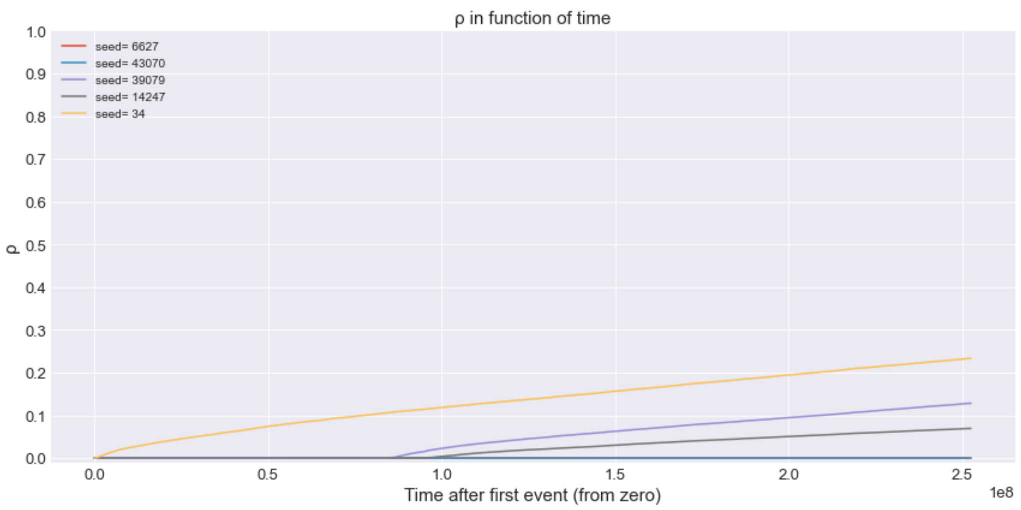


Figure 16: $\rho(t)$ for different seed nodes

As expected, we can see that starting the infection with the node having the highest degree centrality leads to higher infection rate than with the other nodes.

An interesting aspect that emerges from the above plot is that the main differences in spreading speed between different seed nodes are mostly visible at the beginning of the virus spreading. This is basically due to two aspects:

- the spreading speed is faster if the seed node is more connected
- the spreading speed strongly depends on the timestamps of contacts between nodes: as we can imagine the node number 14247 starts to spread the virus after many timestamps, and this is due to the fact that it starts contacting other nodes a lot of time after the beginning of the virus spreading.

As an overall conclusion for this experiment, we can say that the prevalence of the disease may change remarkably depending on the seed node.

Contextualization: Managing to infect a specific person first is the best way to ensure maximum infection over the network.

4.4 Experiment 4: Most isolated nodes

Goal: find the most isolated or protected nodes, that is the ones who are less prone to get infected when a virus starts spreading in the network.

To find the latter, we computed the median infection time of each node of the network on 100 SI simulations with one random seed node per simulation, by fixing $p = 0.7$.

On the Figure 17, firstly, we can see that there is high concentration of nodes infected in the same period (bottom left). Then, we can find our *most isolated nodes* on top of the plot, which are relatively sparse except for an interesting community in the top right.

Contextualization: These persons will be the least infected in the network and they may require a long time to get the infection; so, from a hacker point of view, if it is desired to infect one of those isolated nodes, then a different strategy from randomly infected person will have to be chosen.

4.5 Experiment 5: Best infection time predictor among node descriptors

Goal: Find the most correlated nodes descriptors with infection time.

In order to understand the best way to predict the infection time of a node by looking at its descriptors, we used the same results obtained by the simulation computed for Experiment 4.4 and we correlated the obtained median infection times with different nodal descriptors such as: betweenness/eigenvector centrality,

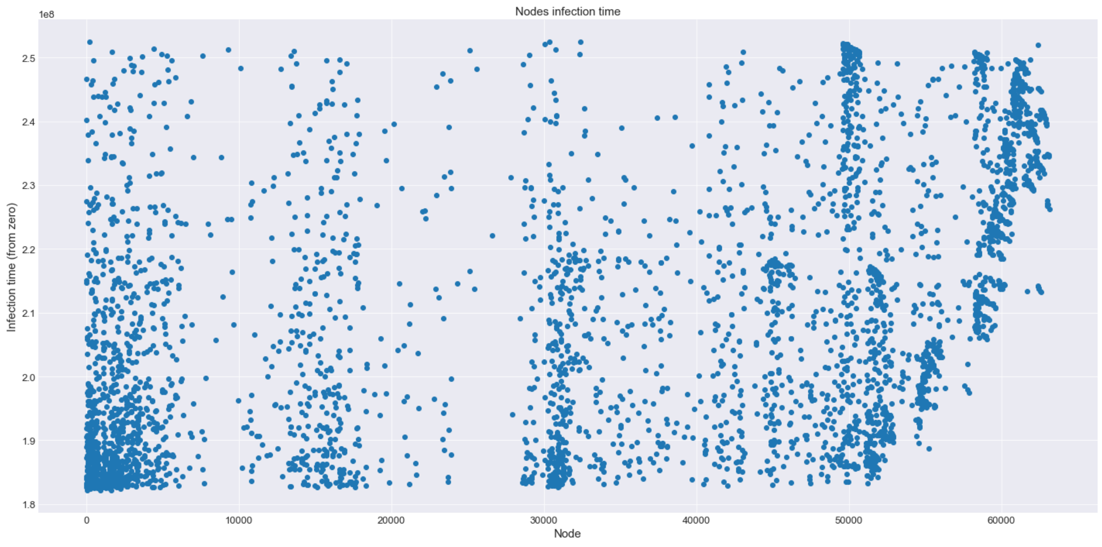


Figure 17: Median infection time for each node on 100 simulations

degree or strength. We therefore performed the *scatter plots* shown in the Figure 18.

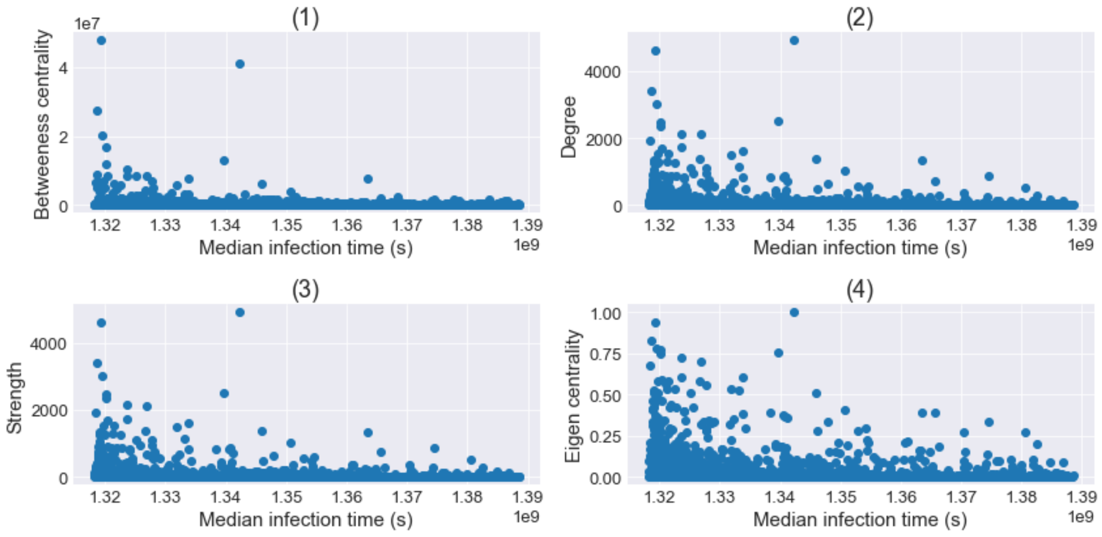


Figure 18: Correlation of node descriptors with median infection time

The first observation that has to be made is that there are a few extreme values that are out of the agglomeration of most of the nodes. Those are having the biggest strength, degrees, betweenness and degree centrality. Because of such nodes, it is hard to perceive any correlation on the Figure 18.

On top of that, there are nodes that have an infinite median infection time (meaning that they have never been infected on 50% or more of the simulations). These are not even plotted but will make the correlation computation harder for us. That being noted, we computed the Spearman rank-order correlation coefficient with and without these infinite values. As it is shown on the documentation, this correlation coefficient does not assume that both datasets to compare are normally distributed. Moreover, we cannot assume the dependency between the median infection time and the different nodal network measures to be linear. An important remark is that it assumes values between -1 and +1, with 0 implying no correlation.

The obtained correlation values are shown on Table 2.

	With inf. values	Without inf. values
Betw. centrality	-0.0307	-0.267
Degree	-0.0111	-0.3648
Strength	-0.0111	-0.3648
Eigen. centrality	0.0197	-0.4048

Table 2: Correlation between node descriptors and median infection time over all nodes

The highest absolute correlation coefficient that the median infection time has is the one with the **Eigenvector centrality** with a value of -0.4048 (without infinite values). What we can interpret here is that, the lower the eigenvalue centrality of a node is, the more it will take time to infect it, which makes sense as it is less likely to be in contact with potentially infected nodes if its centrality is lower.

About the infinite value skipping: without having the data processed in order to skip the infinite median infection time in the computation, the conclusions are harder to draw as all 4 correlation coefficient are very similar (and very low at the same time).

Contextualization: If somehow an hacker managed to compute eigenvector centrality of the network [9], he would be able to determine how to predict the infection time of a certain node, considering a random choice of initially infected nodes.

4.6 Experiment 6: Immunization strategies comparison

Goal: Analyze the possibility to immunize some nodes.

[9]For scientific transparency, authors of this report must say that it is very unlikely

We may be interested in limiting the spreading of the disease by immunizing some nodes, i.e. by putting to zero their probability of being infected when getting in contact with infected nodes. In particular, we tried to understand which strategy to adopt to immunize the considered nodes, by trying the following ones:

- immunization of random nodes,
- immunization of nodes with the largest values of the measures of centrality/importance we used in the previous experiment, i.e. the betweenness centrality, the degree, the strength, and the eigenvalue centrality. In addition, we considered also the nodes clustering coefficient.

The infection probability was set to 0.5 for non-immunized nodes and for each immunization strategy we averaged the results over 20 runs. In order to reduce the variance due to the selection of the seed nodes, the same seeds were used for investigating all immunization strategies.

In particular, we proceeded by following these steps:

- we first selected a list of immunized nodes for each strategy
- then, we selected 40 random seed nodes such as none of them belongs to the group of immunized nodes in any of the different strategies
- after performing the simulations, we plotted the prevalence $\rho(t)$ for the 5 different immunization strategies.

The obtained results are shown in Figure 19: as we can see (also by printing the actual prevalence values), the best immunization strategy is the one based on the strength of the nodes which allows to reach the lowest infection rate at the end of the simulation. As expected, the random strategy performs pretty poorly, the same happens for the one based on the clustering coefficient. Instead, all the other strategies reach almost the same value around 39.5% of infected nodes. Indeed, on the Figure 19, we can see that all descriptors plots seem to converge except for the clustering and random strategies lines that reach higher ρ values faster than others.

4.7 Experiment 7: Comparison between immunization and no immunization strategies

Goal: Compare the effect of nodes immunization on the epidemic spreading to non-immunized nodes.

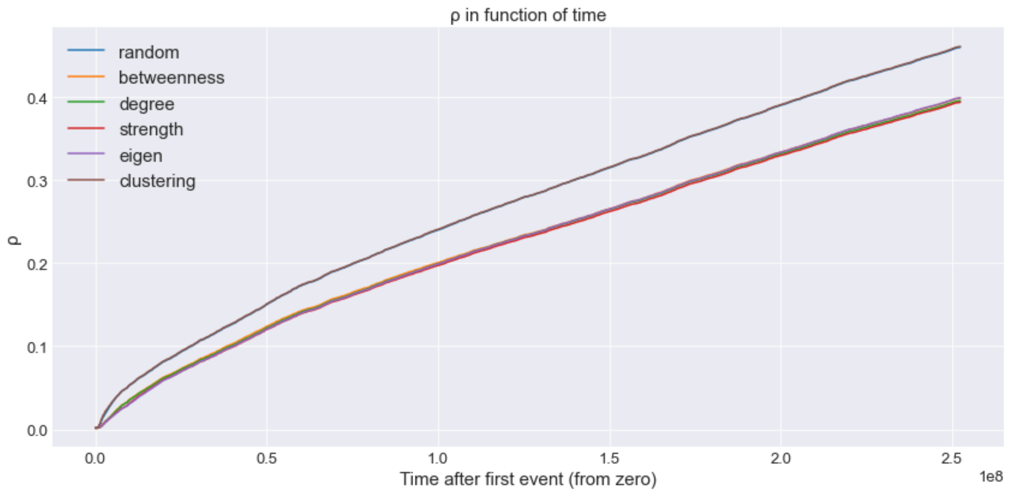


Figure 19: Evolution of $\rho(t)$ for different descriptors

We compared the 2 approaches by running an averaged simulation of 20 runs for each and by using the same random seed nodes we generated in the previous experiment. The infection probability has been set to $p = 0.5$ for non-immunized nodes.

Regarding the immunization strategy, we considered the one that performed the best in the previous experiment (i.e. the one based on the nodes strength, Experiment 4.6), thus we immunized the same list of nodes and we obtained the prevalence plot in Figure 20.

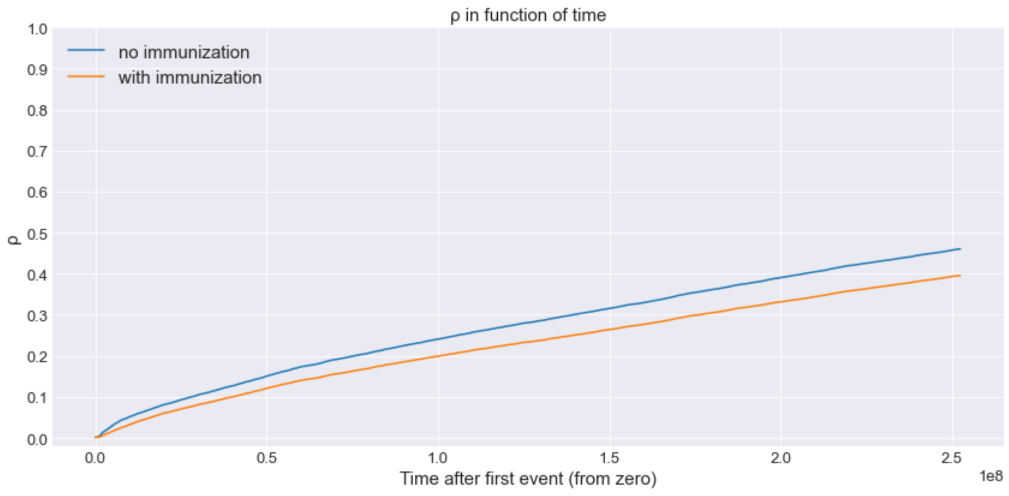


Figure 20: With and without immunization on $\rho(t)$ evolution

As expected, the proportion of infected nodes over time is lower when immunizing specific nodes. However, we could have expected to have a more significant divergence between both approaches as having an immunized node can have a strong influence if it is having a high out degree for example.

Contextualization: If, for some reasons, some nodes were found to be immunized (educated person that would be aware of such virus or having some kind of antivirus) in the network, there will be less spreading of the virus.

5 Conclusions

In this project, we studied the influence of various parameters on the spreading of a computer virus in a real temporal network: nodes descriptors influence (centrality measures, degrees etc.), immunization strategies, seed nodes influence or infection probability influence.

Because of network being real and massive (110k edges and 25k nodes), it has been interesting to process data and to optimize our algorithms in order to provide the fastest and most relevant computations.

To go further, we could have applied various community detection algorithms in order to study spreading dynamics in those specific communities.