Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Masters's Thesis Physics

Chair of Systems Design, ETH Zürich

Towards an Instantaneous Measure for Robustness of Temporal Networks

by
David Bach

Supervised by
Giona Casiraghi, Christoph Gote, Frank Schweitzer

June 2019 - December 2019

Contents

1 Abstract

Information exchange is the fundamental form of interaction in organizations that need to transfer knowledge and collaborate to achieve a common goal. However, the robustness of systems whose primary function is the exchange of information is not easily quantified.
In this work, we introduce Information Access, a measure to quantify how well information is accessible in a system represented by a temporal network, and show how we can use Information Access to study the robustness of such systems. We describe how to efficiently compute and incrementally reevaluate Information Access, allowing for real-time monitoring of the robustness of collaboration networks.
To our knowledge, the proposed measure is the first of its kind that does not rely on defining artificial time windows to study temporal networks. Time window parametrization can lead to erroneous results from assuming causality breaking transitivity. We further improve on the current state of the art algorithms in terms of performance by leveraging modern parallel computing frameworks and latency by allowing real-time evaluation.
We observe significant differences in the robustness of collaboration networks depending on the organizational structure, size, and perturbation strategy.

2 Introduction and Motivation

Real-world systems are complex in their dynamics and behavior. In this work we assume that a temporal network can lossless represent the systems that we study. Networks have proven to be a powerful modeling tool for real-world systems and have been studied intensively over the last decades. Recently the focus has shifted to study temporal networks, networks that incorporate time dynamics. They bring new challenges to how we analyze and understand them but open exciting new opportunities to study their underlying system. When viewing a temporal network as a static network, not all paths are necessarily causality preserving. This changes the evaluation of measures over the network, as shown here [1, 2].

Social organizations are best modeled by temporal networks. We want to know how communication is happening, how information is exchanged, or how community and opinion forms. The order in which interactions are happening has a significant impact on what we are measuring. In the context of communication and exchange in a social organization, how do we define robustness? We might say that as long as people still receive the information they need as quickly as possible, the system is robust against a given perturbation. If by removing a node, whole information pathways are deactivated, the system is not robust against perturbation.

In this work, we want to study temporal networks primarily stemming from collaboration networks with a focus on open-source software development. Every open-source project consists of a set of developers working on a joint code base. Links in the network are created when developers edit the same part of the code base, i.e., co-editing. We assume that this is a proxy for communication or information exchange. If a developer edits parts of the code-base that was written by another developer, we can assume that the information that is encoded in this piece of source code, is now present in both. We represent this exchange as a link in a temporal network.

We can use these co-edits and construct a sequence of timestamped edges, representing our temporal network.

How robust are these networks to developers leaving the project? Are parts of the code base owned by certain people? What is the trend? We know, [3, 4, 5], that especially open-source projects are vulnerable to instances of single contributors leaving the project.

We are proposing a novel temporal measure, called Information Access, which measures the access to information that nodes in the system have, discounting the time it took them to access it. When all nodes can quickly access information from all other nodes, we measure a high Information Ac-

cess and argue that such a network is less prone to single points of failure. Furthermore, we use Information Access to study robustness by measuring the change when we perturb the system. Are we potentially disconnecting the information access pathways of parts of the system when a single, central contributor leaves?
We will provide Information Access as a measure that is computable online, such that we can monitor in real-time how the robustness of a given system changes.

3 Problem Description and Outline

We want to quantify the robustness of systems whose primary function is the exchange and application of information. Ultimately we want to understand if a system can continue to exchange information even when facing perturbations. We cannot directly measure this. We only have access to a temporal network formed from interactions between nodes in the system. We, therefore, tackle the problem of first creating a measure that quantifies the information access based on observed, temporal interactions and second using this measure to analyze robustness.

Throughout this work, we assume a setting where we receive a continuous sequence, potentially unbounded, of ordered, timestamped edges as input to our computation. We then evaluate Information Access over this sequence. We will define robustness using Information Access, create different perturbations, and study the impact that these perturbations have on Information Access.

From an algorithmic point of view, we need to translate a temporal network measure into a computation that is incrementally evaluated and can be computed online.

We will proceed as follows:

In Section 4, we will provide the necessary background on robustness, the primary source of data we are using, temporal networks, and incremental computation. Section 5 will introduce Information Access, discuss its application to temporal networks, and validate it. Section 6 and 7 will describe the design and implementation of Information Access in a computational framework, Differential Dataflow, that supports online analysis. In Section 9, we will use Information Access to analyze a set of collaboration networks, consisting of data extracted from the `Git` version control software as well as others. Finally, in section 10 and 11, we will provide future improvements, other findings, and conclude.

4 Background

This section will cover the necessary background before we can define and discuss our proposed measure. We start with a generalized introduction to robustness and define what robustness means in the context of this work. Then we will discuss the primary source of data that is used in our analysis and its validity to our case. We go on and define temporal networks, paths and distances, which are the basic building blocks we use to define Information Access. We will discuss how time windows in the literature are used to analyze temporal networks and especially how we are treating them. We finish with a discussion of the stream processing paradigm which is necessary for real-time monitoring and give a short introduction to *Differential Dataflow* the incremental computation framework, which we use to phrase our measure as an instantaneous computation.

4.1 Robustness

Robustness of a given system is usually referred to as its ability to maintain high-level function while perturbation occurs [6].
Some studies specify robustness as a networks ability to protect nodes critical to the performance [7] or the function under removal of nodes or links [8].
Robustness is a concept that occurs in various fields and applications, most prominently in biology, the robustness of ecosystems [9] or organisms [10] against perturbation, but also in social environments, e.g., the robustness of communities or cooperations [11] , or in the broader sense of network science, computer networks or power networks [12]. It might also refer to the ability of our system to maintain certain high-level, stable states and under perturbation remain or change to another stable state [9, 13].
Maintenance of high-level function is an abstract concept. We must first define a measure that quantifies our definition of function and use it to analyze robustness.
The measure that is used is in general based on accessible data, such as the topology of the network, interactions, and paths. To quantify robustness, we compute our measure over the original unperturbed system as our point of reference. Then we choose an attack strategy and remove parts of the system. After each perturbation, the former measure is reevaluated. Robustness then refers to the ability of the system at hand to maintain the value of the original measure.
Different measures from network science are used to quantify robustness, such as the size of the strongly connected component or the global between-

ness centrality [14]. [7] and [8] measure robustness via a metric called efficiency, a normalized measure that quantifies the average, reciprocal distance between all nodes in the system. Robustness in temporal networks is defined similarly. In these studies, [7, 14, 15], known network measures are extended to cover the case of temporal networks.
We are interested in collaboration networks. We use co-edited links from large, open-source networks and investigate their robustness. The robustness of collaborative networks hinged on their interconnectivity, as shown in [3], and the knowledge transfer between people that contribute. Our measure for robustness must be able to capture these dynamics and distinguish such systems correctly.
We define the high-level function of an open-source software project to maintain the code-base, develop new features, and fix problems that occur. Developers leaving the project and removing expertise and knowledge threatens this. In the worst case of a single developer who is responsible for crucial parts of the system, leaving could lead to a complete stop in productivity and potential the end of a project. Some of such consequences are studied here [3]. Our measure of robustness must be able to capture the access that the organization as a whole has to knowledge located in a single developer.
We evaluate robustness by defining an attack vector, a sequence of nodes that are removed from the network, and the effect that has on our measure. Usually, we distinguish between random and targeted attacks. We will define an attack vector $\overrightarrow{P}$ that contains all perturbations. We evaluate the change of our measure given we apply the attack vector in order. We will receive dim_P many new values. We evaluate the initial measure against the measure after perturbation. Targeted attacks [8] use existing network measures like node centrality to realize an attack vector.
Robustness will results in a value between 0 and 1. The former meaning complete loss of functionality and the latter complete maintenance of functionality under perturbation.

4.2 Collaboration Data

The systems we primarily study are collaboration networks, extracted from open-source projects. The majority of these projects use `Git` as their version control system. `Git` creates a totally ordered sequence of edits in a code-base, called commits. Each commit either adds novel source code, edits existing source code, or removes source code. We extract timestamped co-editing links from an existing code-base using [16]. A co-editing link (A, B, t) describes the event of author A at time t editing a line of code that prior was

edited or created by author B. Our central assumption is that co-editing is a proxy for communication or information exchange. We, therefore, swap the order of source and destination and treat a timestamped link as the transfer of information from B to A.
This lets us extract a sequence of timestamped links from a given open-source project.
We reason that co-editing is a proxy for information exchange as the author editing a given area in a code-base must know how to change it correctly. The information that the prior author had when he was writing the code was either transferred by actual in-person communication, or the author already poses the necessary knowledge. In either case, the knowledge about that particular part of the code-base now exists in both authors, in turn making the system more robust.
There are certain cases where these assumptions do not hold. We can imagine a system where source code is primarily added. We would extract a small set of links, concluding that little communication is happening. It might still be the case that the developers share most of the code-base, and knowledge is shared uniformly. At the same time, when little co-editing is taking place, knowledge about parts of the system might be concentrated in single developers, making the system fragile. Overall we can conclude that the existence of co-editing implies a form of communication.

4.3 Temporal Networks, Paths and Distances

For this work, a temporal network is defined as $G(t) = (V(t), E(t))$ where $V(t)$ is the set of nodes at time t and $E(t)$ is the set of edges at time t. The set of edges $E(t)$ are a sequence of triples $((a, b), w, t)$ that each capture an observed link or interaction between nodes a and b of weight w at time t. We are focusing on unweighted links, but discuss this is Section 10. We assume that interactions happen instantaneously; they have no inherent duration.
Table 1 shows a sequence of example interactions.

Table 1: Sequence of timestamped links or edges in a temporal network

From	To	Weight	Time
a	b	w_1	t_1
b	c	w_2	t_1
d	e	w_1	t_2

We can define the state of the temporal network $G(t_1)$ at time t_1 as the set of interactions that happened up until and including t_1.

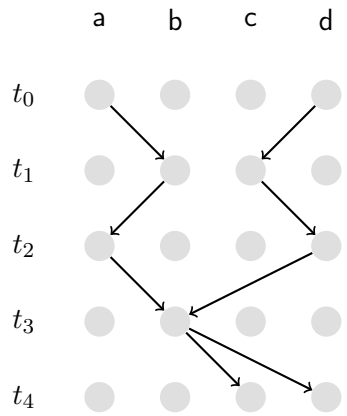


Figure 1: Time unfolded temporal network

A temporal path in this network is described by a time-ordered sequence of links connecting two nodes. Importantly they must be causality preserving. The temporal distance along a temporal path is the time difference between timestamp t_b and t_a. As an example suppose there are two interactions $i_1 = (a, b, 1, t_1)$ and $i_2 = (b, c, 1, t_2)$. By assuming transitivity forward in time there exists a path $(a, c, 1, t_2)$, connecting node a with node c with temporal distance $dist_{a,c} = t_2 - t_1$.
We define a shortest temporal path, also referred to as shortest path, as a causality preserving path with the shortest possible temporal distance, $dist_{a,c}$ between the two nodes a and c. Direct links result in a temporal distance of one.
Figure 1 shows an example network in its time unfolded state, where we can observe a set of nodes $\{a, b, c, d\}$, a set of discrete timestamps and a set of edges, drawn in black. To visualize we draw edges spanning over time. We always view the temporal network in its time unfolded state. Every path that is used to measure Information Access is a path through a time unfolded temporal network.

4.4 Impact of Time Windows

Extending static network measures to temporal networks typically involves defining aggregation windows in which the temporal links are aggregated into a classic graph $G(E, V)$ with edges E and nodes V of static topology. The analysis is done per each aggregated graph. The process of aggregating temporal links into a static topology can be erroneous as causality might

not be taken into account. There exist methods to mitigate that, [2], which introduces higher-order networks capturing causal paths in length up to the order. We still need a time window to aggregate the higher-order network and analyze it, and we still introduce a parameter into the system, the window size, that has an impact on the results of our measures.
Most related work treats finding the optimal time window as an optimization problem. Small time windows may not aggregate a large enough network to quantify meaningful measures, large to no time window introduces redundancies and covers the inherent timescale of the system at hand. By doing a correlation analysis, we can quantify the impact. Especially when not defining a time window at all, aggregate measures become monotonically increasing as links are only added and never retracted. That obscures phenomena at smaller timescales.
The view taken here, [15], is that the choice of a time window imposes an upper bound to the length of a path in the temporal network and should be chosen such that the loss of information of longer paths is according to inherent timescales.
In general, [3], chooses a time window, say 30 days, and an inherent interaction scale, say one day. The robustness, or for that matter any other metric, is evaluated on a sliding window of 30 days.
We see that by introducing a time window, we may still use known network measures to analyze the resulting graphs but the trade-off are potentially causality breaking paths and an additional parameter in our model, which we need to account for. Alternatively, we can preserve the full history by not aggregating the temporal network with time windows.
Contrarily to other literature [2, 17] we do not impose an observation period T to parameterize our temporal network. This is akin to a view often taken by information-theoretic analysis [18, 19] or control theory [20]. We keep the system in is so called time unfoleded state, c.f. [1].
By not imposing an observation period or time window, the full causal past may influence the state of the present system; we do not introduce artifacts from boundaries, and there exists no causality breaking paths. We always view the system in its time-unfolded state.
We actively seek to design a measure that does not impose a time window of aggregation onto the system. That removes one potentially impactful parameter from our models and removes the need for more sophisticated higher-order networks that preserve causality.

4.5 Stream Processing

`Stream Processing` refers to the architectural paradigm of computations over potentially unbounded inputs, also called streams. Contrarily to `Batch Processing` the complete input data is not available at the start, but rather arrives at particular, potentially nondeterministic intervals. This introduces uncertainty when a result of a computation is final and can be produced. Stream processing systems must be built to manage the arrival of novel data and the re-evaluation of computations. Most modern systems rely on giving inputs timestamps, these can be actual wall time or transaction time, i.e., the current assigned transaction number, to make progress in the computation and produce results. A result of a stream processing computation is true as of a given time in the system.

The `Dataflow` model refers to two different though related concepts. In its first sense, it means that the flow of data entirely coordinates computations. The arrival of data triggers the execution of operators whom themselves transform and pass on data. Secondly, the `Dataflow` model refers to the declarative description of data-driven computation. Akin to the functional programming paradigm, we describe our computation as a set of composable operators acting on data. This description is represented as a directed acyclic graph (DAG) - novel system, which we will make use of, allow cycles in the dataflow graph - and execution happens when data is propagated through the dataflow graph. Every operator, `Map`, `Filter`, `Reduce` must define a transformation rule when being supplied with new inputs. The dataflow graph is statically constructed, and data coordinates the execution. Every operator is localized and only needs to know its means of receiving data, its transformation rule, and its way of pushing data along. This *push*-based programming model stands in contrast to a *pull*-based model, e.g., querying a database.

After construction, the dataflow computation runs indefinitely, producing results continuously. Describing our computation in such a way allows, amongst other things, for a simple distribution of our computation. Edges in the formerly mentioned execution graph can span between multiple processes, via, e.g., a *TCP* connection. We can construct identical dataflows in two processes and shard the input data via a hashing function. Every process is responsible for producing results over a partial set of the input data. This is commonly known as data parallelism. We exploit the fact that data often can be partitioned into segments that have little relation. Some operators potentially need an input that does not reside in the memory of the local process. The same dataflow edges can be used to ensure that operators

spanning multiple processes exchange their data.
We are using Timely Dataflow [21] as our run-time for distributed, cyclic, low-latency stream processing. It is available as a framework written in the *Rust* programming language. It provides us with means to create operators, link them, and thus create dataflow computations.

4.6 Differential Dataflow

When our inputs are subject to high frequencies in changes, be it additions and retraction, we would like to update previous results of our computation incrementally, i.e., without recomputing the whole history. This is especially interesting for graph algorithms. Often new input leaves the resulting metric unchanged; adding a link that already exists cannot change the transitive closure of the graph. On the other hand, removing a link may cause a significant amount of change, as many paths are now potentially no longer possible.
This is generally referred to as *incremental computation*. In order to incrementalize a computation $f(X) = Y$ on some collection X we must find a corresponding δf s.t. $f(X + \delta x) = f(X) + \delta f(\delta x) = Y + \delta y$. To be of use, we require δf to only perform work proportional to the size of δx. We hold on to the latest version of the output collection Y to efficiently compute δy. This formalism is taken from [22].
When evaluating iterative or recursive computations, maintaining an incremental computation faces challenges. We must be able to distinguish between new upstream input, i.e., a novel link in the graph and results from previous iterations, i.e., a transitive link.
Differential Computation [23] generalizes incremental computation by allowing timestamps to be partially ordered. We can use this to model multiple different sources of inputs along different coordinates. Every coordinate in the partial order can independently experience change. In principal, the timestamp is multi-dimensional. In this work, we will only use two dimensions, one for new input and one to model the iteration number.
Differential Dataflow is a programming framework implementing differential computation on top of Timely Dataflow. It gives us access to incrementalized versions of known functional-relational operators, like `Join` or `Reduce`. When we phrase our algorithm as a differential dataflow program, composed of these incrementalized operators, the entire computation can efficiently respond to arbitrary changes in its input.
Operators maintain their historical updates in an indexed collection called `Arrangement`, which we use to quickly respond to the arrival of new data.

5 Information Access as a new Measure

5.1 Definition

The systems we are studying are comprised of a set of individuals, developers in a software project, or employees in a corporation whose primary interaction is a form of information exchange. We represent these systems as temporal networks, where nodes are individuals, and edges are recorded interactions. We assume that information is stored in nodes and is transferred or exchanged via causal-temporal paths, in the temporal network representing the system. We argue that these path statistics are a proxy for the underlying information transfer mechanisms. Our measure assigns a global value to every timestamp, measuring the Information Access of the whole system.

At any given point in time, we define the Information Access of a system as the sum of the reciprocal temporal distances of shortest temporal paths between all nodes reaching this time. In principal all temporal paths contribute, we select only the shortest one. We will discuss this further in Section 5.2. It is an extensive measure.

A high value in Information Access means that paths between nodes have, on average, a short time difference, low values correspond to long or nonexistent paths. When the time that accumulates over a shortest path between two nodes grows more substantial, we assume a decrease in Information Access. We are combining two properties into Information Access. The first one being the connectivity of the system. The existence of a causal, temporal path between two nodes in the system allows for information, in principle, to propagate through the network. The more pathways exist, the higher is the information access that the system as a whole has. The second part relies on the time it takes on these pathways. The faster a system is in accessing information from the past, the higher is the value of our measure.

For a temporal graph $G(t) = V(t), E(t)$ Equation 1 shows how we compute Information Access.

$$\mathrm{IA}(t) = \sum_{i \neq j} \frac{1}{\mathrm{dist}_{E(t)}(i, j)} \tag{1}$$

with:

- $V(t)$ the vertices at time t
- $E(t)$ the set of edges at time t

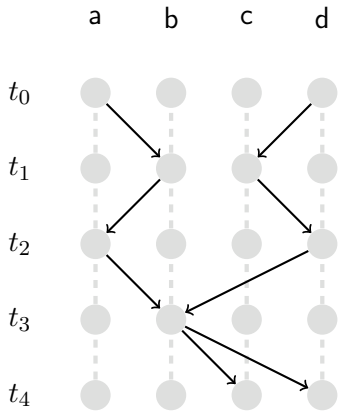


Figure 2: Information Access Example Network

- $\text{dist}_{E(t)}(i, j)$ being the temporal distance, or latency, along a shortest path from i to j for the set of edges at time t.

Taking the reciprocal of the temporal distance is an elegant way to model paths that do not exist. These have an infinite temporal distance and do not contribute. We also introduce a natural decay into our measure. As discussed in Section 4, we do not restrict the length of these temporal paths. There could exist a path between two nodes starting at the origin of the system reaching up to the present. Its contribution to our measure decays with $\frac{1}{t}$. In theory, we would assume our measure to decay according to $\frac{1}{\alpha t}$, whereas α would model a speed-up or slow-down in the access mechanism. For this thesis we set $\alpha = 1$.
Let us now observe how our measure works on an example network. Figure 2 shows a time unfolded temporal network, consisting of four nodes and five timesteps. For this example, we can assume the nodes to be people part of an organization, exchanging information. Direct links are recorded interactions between individuals. When evaluating, we assume the time difference between subsequent timesteps to be equal to 1.
There are two flavors of links. Direct links, representing interactions between individuals, which we observe in our data, in black, and links between a node and itself in the future. These are colored gray and dashed. The latter are memory links, representing our mechanism to simulate individuals remembering past interactions. What is their effect?
The following interaction is made possible:
Suppose there exists a shortest, temporal path between two nodes in the system at a point t_1. At a later point, t_2, there exists no path anymore in

the time-unfolded network. By adding the self-link, we allow for this path to exist again and contribute. From the point of the receiving node, we argue that it remembers the interaction it had seen before. In other words, at t_2 the receiving node still has access to the information transferred in the timestamp before. The temporal distance of this path will have increased. At the second timestamp then, t_2 the path will contribute less to our measure. We add self-links for all nodes for all timesteps. These then can be freely used to allow for pathways to exist.

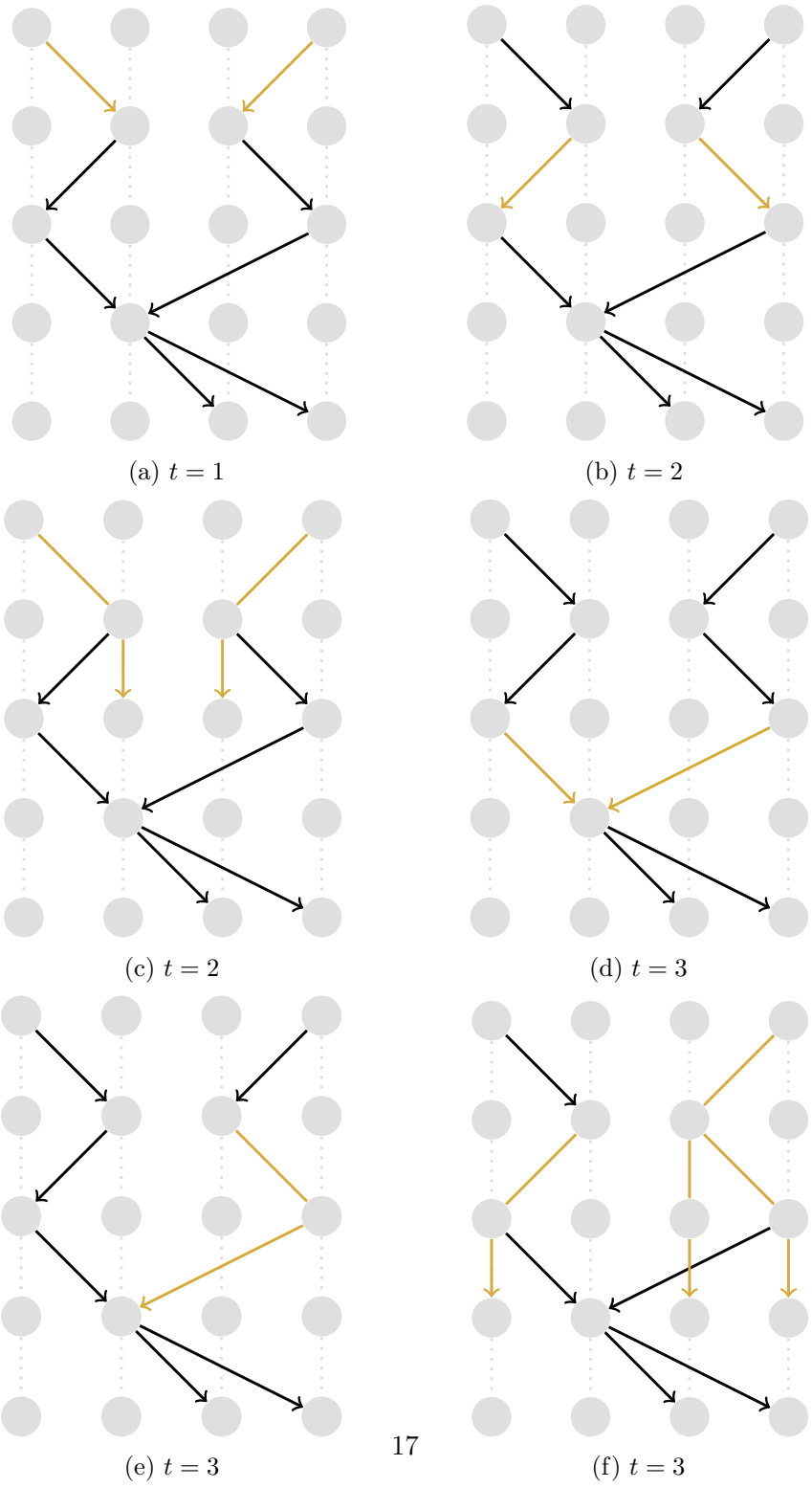
Before evaluating Information Access for the given example, let us examine our expectations in Figure 2. At discrete timestamps, nodes interact via direct links. Until time t_2, we observe disjunct interactions. Nodes $\{a, b\}$ do not interact with nodes $\{c, d\}$. Globally we expect these times to show small information access; half the nodes cannot interact with the other half. At time t_3 node b connects the two disjunct parts. From now on, there exist pathways between all nodes in the systems, and we expect an increase in information access. At time t_4 there nodes c and d can access information from nodes a and b. Our measure should increase in timestamps t_3 and t_4.

Figure 3 highlights a set of paths for the example network 2, paths are drawn in yellow, up until timestamp t_3 and Table 2 shows the corresponding Information Access values.

From t_1 to t_2, c.f. Figure 3a, we have two interactions, and the formerly mentioned memory mediated self-links. When summing all shortest paths, we only find the two direct links as our paths. Self-links do not contribute to the measure. That results in a value of 2. For IA($t = 2$) we again find two direct links, shown in 3b, but also two paths that are mediated via memory, 3c. The memory mechanism persists the formerly existing two paths (a, b) and (d, c). However, as the temporal distance along these two links is $t_2 - t_0$, which in our example equals to two, they only contribute $\frac{1}{2}$ per path. Again for t_3 we see paths in Figure 3d, Figure 3e and Figure 3f.

Table 2: Information Access for an example network. Time difference between subsequent timesteps is 1.

Time	Contributing Paths	IA
t_1	(a, b), (d, c)	2
t_2	(b, a), (c, d), (a, b, b), (d, c, c)	3
t_3	(a, b), (d, b), (c, d, b), (b, a, a), (c, d, d), (d, c, c, c)	3.8
t_4	(b, c), (b, d), (a, b, c), (a, b, d), (d, b, c), (c,d), ...	5.5



(a) $t = 1$ (b) $t = 2$

(c) $t = 2$ (d) $t = 3$

(e) $t = 3$ (f) $t = 3$

Figure 3: Shortest, temporal paths in an example network

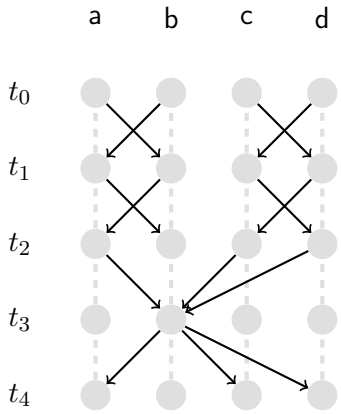


Figure 4: Second example network to compare Information Access

5.2 Comparison and Validation

We will now further compare our measure to three different path or link based measures. We will use the interaction example from Figure 2 and from a second constructed example Figure 4.

Comparison with link counting

Our measure is inherently path-based. We sum the shortest paths and their corresponding time differences. Alternatively, we can think of a simpler model built on the available observations, counting direct links. Link counting measures the amount of direct activity that is happening in a system. We could consider this model as it is simpler, and it does not rely on the assumption of transitivity of information access.
To show that we capture more of the underlying systems dynamics with Information Access, we will go on and construct an interaction pattern, where we compare our measure against counting direct links.
This link counting measure is described in Equation 2 with $L(t)$ being the number of links added at time t.

$$\mathcal{M}(t) = L(t) \tag{2}$$

In Figure 2 counting links would lead to a constant value, as every timestep has two direct links, our proposed measure increase while the link count is stable. We now construct an example where the link count decreases, but the measure increases. Figure 4 shows the same set of nodes with a different

pattern than before. We start with two timestamps with each four direct links. At t_3, the two prior disjunct parts of the network are connected via node b. At t_4 we count three direct links.
Table 3 and Table 4 respectively show the resulting values for, amongst others, the link counting measure. Our measure shows an inverse trend to the link counting model.
The link counting model takes only the number of interactions into account and cannot represent the history of the system. Information Access, on the other hand, is centered around nodes interacting and can model the whole causal past. The systems we are studying are comprised of individuals interacting. Our measure should account for these individuals. We conclude that we capture a more accurate view of the underlying system than link counting.

Comparison with reinforced link counting

Link counting does not take the history of the system into account. A refined reinforced link-based measure can be formulated as follows in eq 3:

$$\begin{aligned} \mathcal{M}(0) &= L(0) \\ \mathcal{M}(t) &= \gamma \mathcal{M}(t-1) + (1-\gamma) L(t) \end{aligned} \tag{3}$$

At every timestamp t we add the value of our measure at the $t-1$ timestamp dampened by a parameter γ to the current link count The parameter γ may be chosen how we see fit, tuning the amount of memory we want to have. This resembles a simple, reinforced model that takes memory into account.
We again can look at Table 3 and Table 4 to compare its values to our measure.
For the first example, the values are identical to the link counting model, because the link count is stable. In the second example, at time t_3 when link count drops, the reinforced model maintains a higher value, but still declines downwards.
This model too, cannot reproduce the same trend we observe in IA, albeit modeling memory.

Comparison with taking all paths into account

As discussed at the beginning of this section that when quantifying information access, all possible, not just the shortest temporal path, are pathways by which information can be accessed. To investigate the impact on our measure, we also compute the results given we take **all** paths into account.

Taking all causal, temporal paths into account should lead to values that are greater or equal to our measure. Again observing and comparing these values in Table 3 and Table 4, we see greater values as more paths contribute. The trend in both models is similar. We observe almost twice as large values when taking all paths into account.
The shortest path based model will always use the most contributing paths, due to the $\frac{1}{t}$ factor when summing. When the temporal distance grows larger, a substantial amount of paths in the all paths model will have no impact on our measure.
Shortest paths will result in a lower bound on Information Access. Taking all paths leads to an upper bound. When studying robustness, we argue that the lower bound is of more value as we in turn also measure the lower bound to robustness. We want to understand when systems are failing to maintain their function. A lower bound on robustness can model a worst-case scenario. We also encounter problems when adopting an all path model:

1. Computing shortest path is proven to terminate, and uniquely defined, contrarily to finding all paths. It is not clear when our algorithm should stop, given there are cycles in the graph.

2. The time complexity of finding the shortest path is smaller than all paths leading to shorter execution times.

Table 3: Measure results for Figure 2, $\gamma = 0.5$

Time	Naive Counting	Semi-Naive Counting	IA [All]	IA [Shortest]
t_1	2	2	2	2
t_2	2	2	3	3
t_3	2	2	5.5	3.8
t_4	2	2	10	5.5

Table 4: Measure results for Figure 4, $\gamma = 0.5$

Time	Naive Counting	Semi-Naive Counting	IA [All]	IA [Shortest]
t_1	4	4	4	4
t_2	4	4	4	4
t_3	3	3.5	8.25	4.5
t_4	3	3.25	15.75	7.5

We can conclude that our measure captures non-trivial and more accurate mechanics that simpler models cannot reproduce. The reinforced link count-

ing model results in larger values for Figure 4 from its memory effect but cannot reproduce an increase in the measure. Taking all paths into account increases the absolute value but is infeasible to compute. We argue that using the shortest paths is a lower bound on the potential Information Access in the system and, therefore, more suitable when studying robustness.

Validation on a synthetic data set

We use a synthetic data set containing 10 nodes and 100 distinct timestamps to investigate the value of our measure as well as the reaction to decrease in interactions. In the first 50 time steps, every node is connected to its neighbor. That means that every node is reachable from every other node. That leads to an IA value of 90. In the times $t = 50 - 75$ only node 1 interacts with every other node at every time, meaning that all other paths are supported by the memory effect. The system anneals to the equilibrium of $IA = 10$. Figure 5 shows the resulting time series. We expect a $\frac{1}{t}$ decrease in the ME from $t = 50$ until the new minimum value is reached, as the time difference is increasing with every step. At $t = 75$ we reactivate the prior interaction pattern and see an instantaneous rebound to the original value.

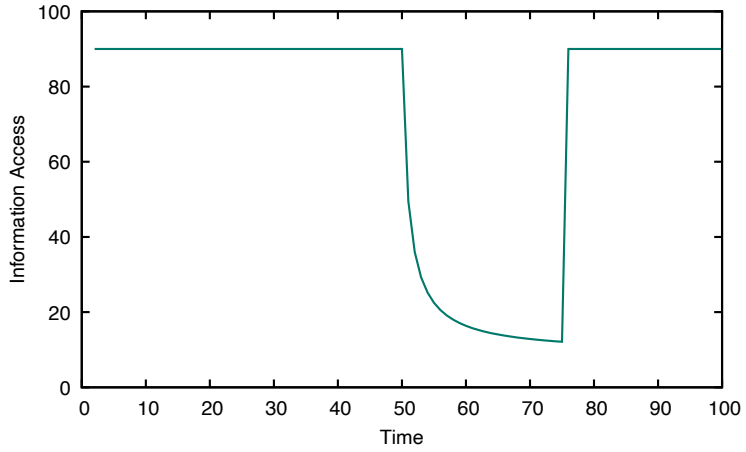


Figure 5: Synthetic Example

This is to be expected as we always look at the shortest temporal paths. As soon as we reactivate the initial interaction pattern, new shortest temporal pathways exist that contribute to our measure.

5.3 Conclusion

We introduced a measure based on shortest, temporal paths in a time-unfolded network. We showed that in small examples, it could reproduce expected trends when thinking in terms of access to information. The primary means of a node to access information from another node is a causal, temporal path. Memory is a mechanism to model the history of interactions between nodes and transfer it into the present. We considered two simpler models taking only the interactions into account, but concluded that these are unable to correctly represent individuals exchanging information and the evolution of the system. We discussed taking all paths into account but concluded to use the shortest paths as a lower bound. Finally, we showed how our measure reacts to changes in a synthetic dataset and fits our expectations. When interactions decrease, we observe a slow decay made possible from our memory effect, which is what we expect as the Information Access the system has will slow down but not instantaneously. When interaction increase, we see a sharp rebound from the new shortest temporal paths mediating Information Access. At these times the system again has quick access to the information stored in its nodes. We argue that this proposed measure uniquely captures temporal dynamics in the system and acts as a proxy to the global access to information that a system compromised of single agents or nodes have.

6 Design of an Instantaneous Measure

Instantaneous refers to the ability of a given measure to be evaluated upon receiving a single additional interaction. Robustness refers to a high-level function and thus cannot be computed by merely taking a single interaction at a time.

We refer to instantaneous rather as our ability to quickly react to changes. In section 4 we discuss *Differential Dataflow* and *Incremental Computation*. These are how we can maintain our measure incrementally against changes in its input. In summary, we are incrementally maintaining a stateful computation that evaluated the robustness of a temporal network.

Our proposed measure from section 5 is composed of finding all shortest temporal paths for all nodes in the system while summing their reciprocal temporal distances. This needs to be done for every potential time step. Finding shortest paths can be done with a recursive algorithm traversing the temporal network in a breadth-first search. For this to work instantaneously, we must maintain all already produces shortest paths for all node combinations. As soon as we receive new links, we have to evaluate our pathfinding algorithm until fixpoint such that we find potential new shortest paths. These are passed to a sum operation that sums all reciprocal distances.

The fixpoint is uniquely defined as the point when all shortest paths between all nodes have been found. The evaluation strategy is similar to semi-naive, bottom-up evaluation in the *Datalog* programming language [24, 25].

In essence, our problem can be transformed into efficient incremental view maintenance with recursion. This has been a research problem for decades [24, 26]. We have the added complexity of working with temporal units as a part of our data. This mostly changes the semantics of the shortest path between two nodes. We define the shortest paths as these paths having the shortest temporal distance, but we also need to find the latest path. When evaluating our algorithm at time t_a we might have already seen a shortest path between two nodes with a minimal temporal distance, but we need a path with the minimal distance that ends at t_a. This increases the amount of change that happens. As explained in 4, the fundamental axiom of differential computation is, that we only do work relative to the amount of change our computation sees.

The `DRed` and `Count` algorithm presented here [26], discussed here [27] and here [28], are similar in spirit to the incremental evaluation strategy we follow.

We keep track of the number of potential derivations for a certain path.

When an edge is added that was already present or resolves into a larger temporal distance, we do not need to update our computation as no new shorter paths are possible.
The partial order [23], allows for efficiency and correct computation of derivations resulting from fixpoint iterations, such as our measure.
Ultimately, we designed a recursive algorithm that incrementally maintains the computation of Information Access. This in principle allows us to deploy a real-time application, e.g., a dashboard monitoring Information Access, that consumes a stream of co-editing edges.

7 Implementation with Differential Dataflow

7.1 Architecture

An implementation of *Differential Dataflow* is written in the *Rust* programming language [29]. It comes with a set of operators and methods to construct and run dataflow programs. Naturally, the implementation of this work is done in *Rust* too.

Our computation must be described as a sequence of operators that transform incoming data and produce the desired results. Operators range from simple `Map` operations, transforming an incoming piece of data with a supplied function, to stateful `Reduce` operators that may compute aggregates like `Sum` or `Count`. As mentioned in 4, one crucial capability that the system has, is support for cyclic dataflows. A cyclic computation is necessary to implement recursion or iteration.

Figure 6 shows an exemplary execution graph for a `PageRank` [30] computation. Nodes represent operators, and edges represent channels where data can flow. Every operator has a unique id, displayed next to its name. `Input` operators take care of introducing novel data into the system. Often this is done by consuming, e.g., a TCP stream, querying a database, or reading from a file. `Sink` operators are their counterparts. They egress the results of our computation into external systems.

`PageRank` is an iterative algorithm. Results of a previous iteration are fed into the current iteration. In this case, the dataflow is greatly simplified as the computation is hidden in the `PageRank` operator. The `Probe` operator is needed to keep track of how far dataflows have progressed. This becomes important in the distributed setting. When producing a result we must ensure that every peer process has also finished up until and including that point. This gives us correctness guarantees.

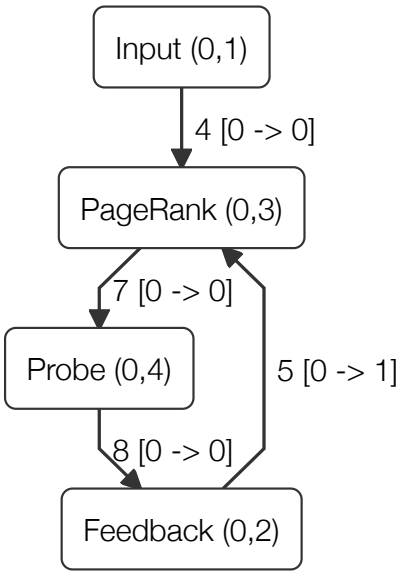


Figure 6: PageRank execution graph

The topology of this execution graph is known at compile time. At runtime, the graph is constructed, and data is fed into the system.
This architectural decision allows for easy distribution. Multiple instances of the same dataflow program can be constructed; data is sharded based on a hashing function, and operators exchange data via dataflow edges. This form of distribution is called data parallelism. Every peer is responsible for a subset of data. Data coordinates the execution.
Finding the shortest paths in a temporal network is a recursive computation. The execution graph of our algorithm will be similar in nature to Figure 6, having a feedback loop to find all shortest paths and propagating the results into an aggregation operator.
For the purpose of this work we are limiting `Input` or `Output` to reading from resp. writing to files in our local file system.
In our case triplets `(a,b,time)`, representing a timestamped edge between node `a` and node `b`, will be flowing through the dataflow.

7.2 Algorithm

To compute Information Access we need two parts. Find all shortest, temporal paths for all nodes and then reduce them by summing the reciprocal temporal distance.
We need to phrase a pathfinding algorithm using *Differential Dataflow* operators.
Listing 1 shows an excerpt of our shortest path algorithm.

```
let roots: Collection<T, (u64, u64), Path> = source.roots();
let edges: Collection<T, (u64, u64), Path> = source.edges();

roots.iterate(|roots| {
    roots
        .join_map(&edges, |_src, root, dst| (*dst, *root))
        .concat(&roots)
        .map(|(dst, root)| ((dst, root), ()))
        .reduce_core::<_, OrdKeySpine<_, _, _>>(
            "ShortestTemporalPath",
            |(dst, root), input, output, updates| {
                if(output.is_empty() || input[0].1 < output[0].1)
                    updates.push(((), input[0].1))
            })
})
```

Listing 1: Finding shortest, temporal paths

`roots` and `edges` are collections, i.e. streams that consist of tuples of integers, `(1,2)` indicating an edge between node `1` and node `2`. The `roots` collection carries the same value at both tuple positions. A `Collection` has a `Data` type, it is `(u64, u64)` in the example, for the rust primitive unsigned 64-bit integer, a timestamp `T` which we use to encode the timestamp that comes with our data, and a difference type, `Path`.
We are using stream and collection interchangeably.
The difference type indicates the multiplicities of a given tuple. Mathematically it is described by a semiring. `Path` is our own data structure resembling a causal, temporal path. To form a semiring we need to implement the two binary operations, multiplication and addition. The tropical semiring, `(min, +)`, minimum, and sum, can be used to implement the shortest path algorithm efficiently [31]. In our case, the `join` method applies the multiplication to the value field when joining a key, and the `reduce_core` will apply the addition for every key.

We start an iterative computation from our roots collection. The `iterate` method constructs a cyclic dataflow, feeding the output of its inner computation back into itself. We go on and `join` the `roots` collection with the `edges` collection. Every join we execute can be thought of as **one** hop in the temporal networks. We flip the order to `(dst, root)` to keep track of the starting point of our path. The `reduce_core` method assumes its input to be in the form of `(key, val)`. That is why before applying this method, we first map the collection from `(dst, root)` to `((dst, root), ())`. `()` is the unit type in rust.
We key our collection to tuples of destination and roots. In other words we apply the `reduce_core` to every combination of destination and root nodes, i.e., all possible paths. The `reduce_core` maintains the current shortest path and if and only if the binary comparison is true, `input[0].1` and `output[0].1`, will update the path. `reduce_core` maintains state, in our case the current shortest path for given tuple combination.
As we exploit the monodic structure of our problem, new links are only added, i.e., we have monotonicity, we need to only keep one single `Path` struct in the `output` array. Meaning we only keep one `Path`, the shortest path, for every possible `(node, node)` combination.
Section 10 will discuss a more general algorithm that can also handle non-monotonic inputs. This is important when removing nodes or links.
The inner closure of the `iterate` method is continuously applied until a fix-point is reached. Every iteration extends one hop starting from the `roots`. This is a breadth-first search starting from our `roots`. A fix-point is reached as soon as the collection that is iteratively developed does not change anymore.
The output of this algorithm is a stream of shortest, temporal paths.

```rust
pub struct Path {
    /// Accumulated temporal distance.
    pub temp_dist: u64,
    /// Initial time of this `Path` segment.
    pub left: Time,
    /// Destination time of this `Path` segment.
    pub right: Time,
}
```

Listing 2: Path data structure

Listing 2 shows the data structure that is used to keep track of the temporal

distance. Every time there exists a shorter or more recent path between two nodes, this one will be produced as our result. We keep track of the associated timestamps to ensure that only causal paths are produced and to compute the resulting temporal distance.
This allows us to incrementally maintain the shortest paths between all nodes and react to novel input. A new link might allow for new shorter paths, or it might be redundant.
The second part of computing our measure reduces the sequence of paths by summing the reciprocal temporal distance. For every timestamp, we add all paths that have this time as their `right` timestamp.
This results in a global value for our system quantifying the Information Access that is possible at that time.
Listing 3 shows the overall computation. The method `information_access` takes a collection of shortest paths and computes the information access measure. It returns a collection only having a `Float` value, i.e., the Information Access measure.
We pass the shortest paths to our aggregation method `information_access`. Besides a stream of shortest paths, we also pass a time scale that changes the aggregation interval, i.e., the resolution on the time axis.

```
let shortest_paths = shortest_paths(&roots, &edges);

let result = information_access(&shortest_paths, TimeScale::Month);


pub fn information_access(shortest_paths: &Collection<_, (Node,
Node), Path>, resolution: TimeScale) -> Collection<_, Float, isize>
{
    /* SNIP */
}
```

Listing 3: Information Access in Differential Dataflow

The output is a stream of Information Access values for every timestamp with a resolution of `resolution`.

7.3 Scaling

Finding the shortest temporal paths can be compute-intensive, depending on the number of nodes, edges, and discrete timesteps. Distribution allows us to compute our measure in parallel.

We are using an Intel® Core™ i7-6700 Quad-Core with 64 GB of RAM. We compute our measure over the coediting data extracted from three different open source projects. The *Rust* programming language, *openBSD* and *libdav*. Table 5 shows a summary.

Table 5: Datasets for scaling benchmark

Dataset	Nodes	Edges	Timestamps
Rust	2524	496854	43257
OpenBsd	78	681291	103909
LibDav	495	220668	27781

We evaluate Information Access over every dataset five times in all variants of number of processing cores and display the average. The execution times in seconds for the three datasets are shown in Table 6.

Table 6: Horizontal scaling for different datasets in seconds

Cores	Rust	OpenBsd	Libdav
1	2405.33	556.39	434.80
2	1350.47	294.72	231.66
4	820.52	149.24	130.59

We can see almost linear scaling for all datasets. The scaling depends on the skew in the data, as we achieve multi-core capabilities by means of data-parallelism. If there are nodes that are significantly more active, the processor responsible for computing this part of the data is busy, while some other processor might idle. Adding more cores also adds communication overhead as the data must now be sharded between more cores and potentially exchanged more often. That is why we usually see sub-linear scaling.
Figure 7 shows the speed up we achieve by exploiting multiple cores.
This allows us to tackle larger datasets and keep the latency low when evaluating our measure in real-time.

7.4 Conclusion

We show how a dataflow architecture is used to execute computations in parallel and that it is feasible to compute a recursively defined measure like Information Access incrementally.
This allows us to:

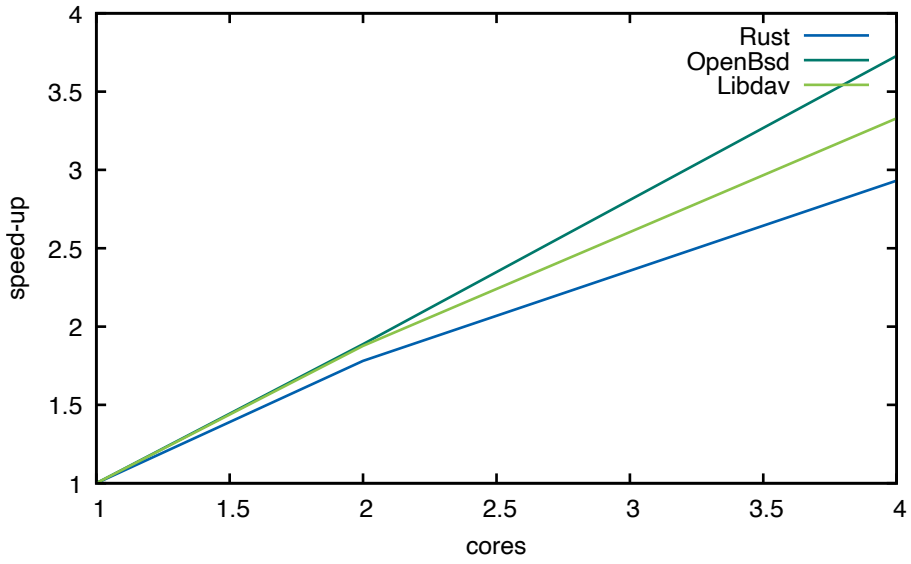


Figure 7: Speedup with multiple cores for different datasets

1. Tackle larger datasets and horizontally scaling to keep execution times low.

2. React quickly to new data allowing to maintain Information Access of a system in real-time.

8 Robustness

8.1 Definition

As described in Section 4 robustness refers to the ability of a system to maintain its function while being perturbed. Perturbation, for the purpose of this work, is defined as removing a percentage of nodes, the failure rate, from the system. Function defined as maintaining the same Information Access.
We focus on the system's response to deactivating nodes, similar to [14]. We define a set of nodes $D \in V$, which is a subset of all nodes in the system to be deactivated. A deactivated node cannot participate in any interactions. Paths that formerly where leading over this node are no longer possible. The ratio of D and V is denoted by the failure rate f. The result of deactivating the set D of nodes leads to a value in our Information Access measure denoted by IA(D, t). IA is a function of time. It assigns every discrete timestep an extensive value. To compute robustness we first aggregate the time dimension, by summing, to arrive at a single value for a given system which we will denote as $AIA(D)$, meaning the Aggregated IA measure of a given temporal network for a set of deactivated nodes D:

$$AIA(D) = \sum_t IA(D, t) \tag{4}$$

Robustness then is defined as the ratio between *Aggregated Information Access* after perturbation and *Aggregated Information Access* before perturbation also called the baseline.

$$R(f) = \frac{AIA(D)}{AIA} \tag{5}$$

As such robustness $R(f)$ is a value between zero and one. In this framework $R(f) = 1$ means that the given perturbation had no effect on our *Aggregated Information Access*. $R(f) = 0$ means that the perturbation led to a complete loss of all *Aggregated Information Access*.
To visualize the effect of different failure rates and attack strategies we invert the robustness to arrive at the fragility denoted as $\mathcal{F}(f)$.

$$\mathcal{F}(f) = 1 - R(f) \tag{6}$$

The fragility $\mathcal{F}(f)$ quantifies the loss in function, whereas the robustness $R(f)$ quantifies the remaining function.
We measure the robustness of a given system in three ways:

1. Compare the effect of five different perturbation strategies on robustness and fragility
2. Compare robustness and fragility to a random topology with similar metrics
3. Compare the normalized area under fragility for all perturbation strategies to the random topology

8.2 Attack Strategies

In most real-world systems, some nodes are more relevant to the function than others. To establish a baseline of perturbation response, we first measure the impact of randomly selecting nodes that ought to be removed. Targeted attack strategies are based on ranking nodes in the system via a given measure. As shown in [32], the ranking may change after removing a set of nodes. We are **not** recomputing the ranking after removing nodes but rather compute the set once, ad-hoc. In this work, we focus on ranking nodes via:

Temporal Betweenness Centrality

Following [2] we define the temporal, unnormalized betweenness centrality of a given node as the total number of shortest paths that pass through that node. Nodes ranking high in betweenness centrality are mediating Information Access between different parts of the system. Using shortest paths as the basis of this measure we expect a large impact when attacking the system with this strategy.

$$g_i = \sum_{j \neq i \neq k} \sigma_{jk}(i) \tag{7}$$

where:

- $\sigma_{jk}(i)$ being a shortes ttemporal path between j and k that passes through i

Temporal Closeness Centrality

We are following [17] in defining temporal closeness centrality as

$$C_i = \sum_j \frac{1}{\tau_{i,j}} \tag{8}$$

where:

- $\tau_{i,j}$ is the temporal distance between node i and node j

We are ranking nodes by how quick any other node in the system can reach them. Using this as our attack strategy, we will be removing nodes that are quickly reached, but might not necessarily be on the critical path of Information Access. We expect a less severe impact compared to using temporal betweenness centrality.
For both these strategies, we also reverse the order and analyze robustness when removing nodes from the bottom of the ranking first.

8.3 Topology Comparison

For each network, we generate a random network with the same number of nodes, number of edges, number of timesteps, but randomly generated edges. We uniformly draw from two random number generators to create an edge between two nodes. Listing 4 shows an excerpt of `Julia` code creating the random topology.

```
for t in range(t_start, t_stop, step=convert(Int, floor((t_stop -
t_start) / timesteps)))
    for _ in 1:ratio
        push!(edges, [rand(0:nodes), rand(0:nodes), t])
    end
end
```

Listing 4: Generating a random topology

We define `ratio` as the ration between the number of timesteps and the number of actual edges in the original temporal network. For every timestep, we create as many temporal edges to end up with the same number of edges and timesteps.

8.4 Evaluation

We compute our measure over different failure rates and attack strategies. This results in a set of time series of our measure, one for each failure rate and attack strategy combination. We aggregate the measure over time, which results in a single value per timeseries. The unperturbed aggregated measure serves as the baseline to compare all other values. This will give us Table 7.

Table 7: Evaluating robustness

failure rate f	r	bt	bb	ct	cb
10	0.77	0.56	0.88	0.66	0.90
20	0.55	0.2	0.85	0.33	0.89
		...			

where:

- r: random perturbation
- bt: betweenness centrality starting from the top of the ranking
- bb: betweenness centrality starting from the bottom of the ranking
- ct: closeness centrality starting from the top of the ranking
- cb: closeness centrality starting from the bottom of the ranking

For all the systems that we study, we first compute the ranking of nodes. Running a perturbation analysis means computing our measure with increasing failure rate. For every analysis, we increase the failure rate from zero in 10 percent steps until we reach 90%.

9 Results

We measure robustness via Information Access in a variety of different temporal networks. We start with two non-collaborative networks, the *London Tube Network*, representing passenger flow in the London tube network, and the *Manufacturing Email Exchange*. The remaining five networks are extracted using `git2net` [16].
Table 8 lists all examined networks and some statics about them.

Table 8: Temporal Network Overview

Name	Nodes	Edges	Timespan
London Tube Network	132	225374	5 days
Manufacturing Email Exchange	154	82929	9 month in 2011
Commercial Software Project	17	108056	2004 - now
Igraph	25	4301	2006 - now
LibDav	495	220668	2002 - now
OpenBsd	78	681291	1996 - 2015
Rust	2524	496854	2011 - now

Every analysis will be done in the following way:

1. Analyze the time series of Information Access.
2. Compare the original network to the random topology over different attack strategies.
3. Analyze the effect on the fragility of different strategies of the original network.
4. Compare the normalized fragility over different strategies to the random topology.

For every computation of Information Access, we choose a respective time resolution.
The robustness evaluation consists of three figures for each analyzed network. The first (a) compares the robustness of the original network, in blue with prefix `o`, to the random network, in red with prefix `r`, over different attack strategies. The second figure (b) displays the vulnerability of the original network to different attack strategies by measuring the fragility in a radar chart. The last figure (c) will compare the normalized fragility, which is the normalized area in the radar chart over the five strategies, of both the

original, in blue, and the random network, in red. The area of the pentagon is computed via Equation 9.

$$\text{Area} = \frac{a * b + b * c + c * d + d * e + e * a}{5} \tag{9}$$

9.1 London Tube Network

The Tube network, taken from [1], does not describe a system where information is exchanged or accessed. Instead, people are moving through the transportation network of the London underground. Links are timestamped movements between stations. We still included this network as it shows interesting behaviors. Our measure is fundamentally based on shortest temporal paths, and these still exist in the tube network. Instead of thinking in terms of information, we think of transportation that happens. We will not impose any memory effect as described in Section 5. A high value in our *Access* measure means that a more significant number of stations are reachable, or the temporal distance on these paths is shorter. A smaller value indicates that either there are less stations reachable via a causal temporal path or the temporal distance is higher.
Figure 8 shows the time series of Information Access. The observation period includes four days in a single week. The initial time is reset to 0. The resolution is a single value per hour.

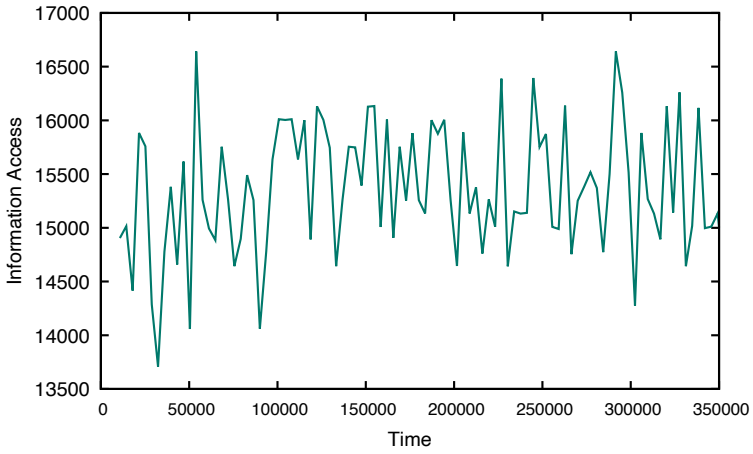
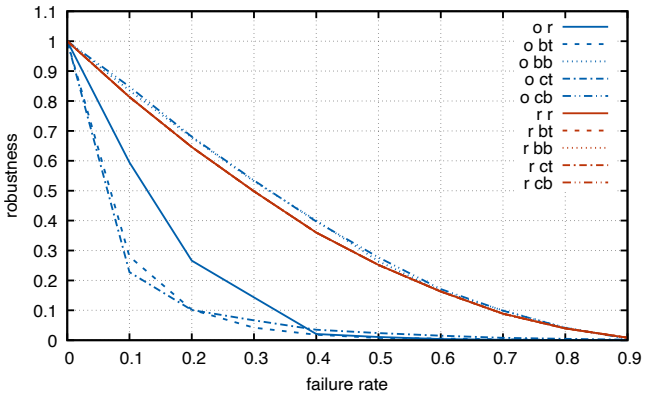


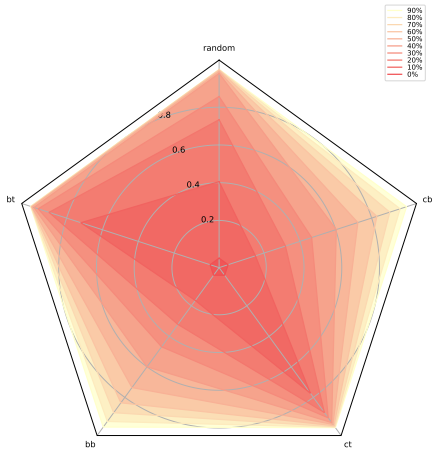
Figure 8: Information Access for the London Tube Network

The pattern seems stable around its equilibrium, with fluctuations in the order of hours. Fluctuations are in the order of 5 to 10 percent. In general,

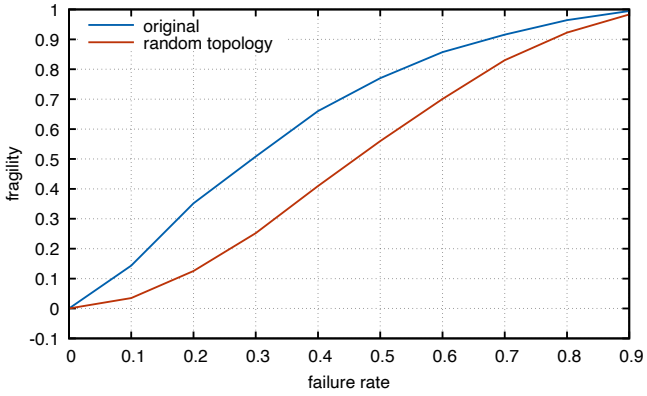
at any given hour, the number and latency of shortest paths that reach from the past to the present are stable. That is what we would expect as the observation period consist of a few days and in such there system should not change significantly.
What to we expect regarding the robustness of such a network? We assume an underground public transport system to be as efficient as possible, but not necessarily robust to failure of single stations. In general, whole stations are not removed.
Figure 9 shows the in Section 8 described three methods to analyze robustness using Information Access. Comparing to a random topology 9a shows almost strictly smaller robustness for all attack strategies. The different strategies do not significantly change the robustness at different failure rates for the random topology shown in red. This is expected as we randomly generate a network where all nodes should be identical in their ranking. As expected in 8 removing nodes that rank highly in closeness and betweenness centrality is significantly more successful in dismantling the function of our system. This results in lower robustness values. Removal of high ranking nodes leads to robustness values of 0.2, meaning we only maintain 20 percent of the original function. Even removing nodes that are not important in our rankings leads to a significant impact on robustness. It appears that almost all nodes in the system are essential to the function. Figure 9b supports that again. We can see large fragility values for all strategies. That is also supported by Figure 9c. The normalized fragility over all attack strategies of the random network is better than the original network.
This is what we expected. The tube network is highly performant, but not robust to perturbation. Every removal of a station decreases the value of our measure significantly.



(a) Random network comparison



(b) Vulnerability to different attack strategies



(c) Normalized fragility

Figure 9: Results for different robustness evaluations for the London Tube Network

9.2 Manufacturing Email Exchange

This data set consists of email exchanges in a manufacturing plant, spanning over nine months [33]. In contrast to the authors, we do not distinguish between different management roles inside the organization, but merely look at the timestamped email correspondence between individuals.
Links in this network do not carry any weight or unit; they indicate that an email was exchanged. Temporal paths exist by assuming transitivity of communication. If a person A exchanged an email with person B and later person B with person C, we assume that there exists a causal, temporal path between A and C.
High values show that more information is accessible from the past along causal-temporal paths at this time. Low values, in contrast, show that access to information takes a long time or is not possible at all. The resolution is set to a single day.
We expect to see time-dependent patterns as email exchanges varies day to day. Especially on Sunday, communication will be slow or not existing. We expect to see significant decreases in our measure on this day. Otherwise, we do not expect any significant non-equilibrium patterns as no event of notice are described in [33]. It is especially noted that no changes to the structure of the people involved happened in the observation period.
Figure 10 shows the resulting time series.

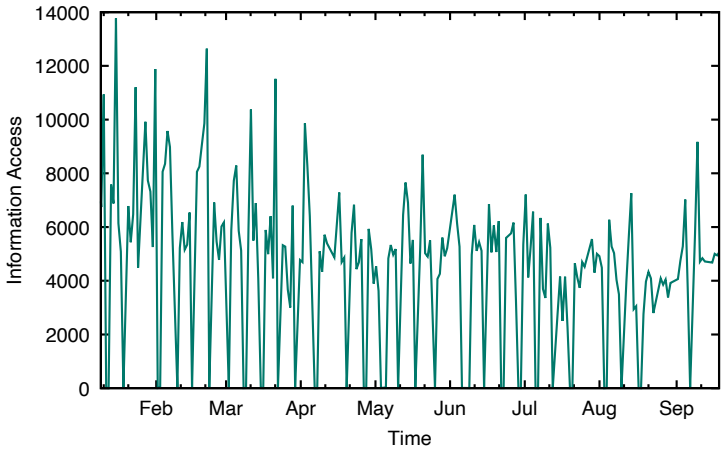
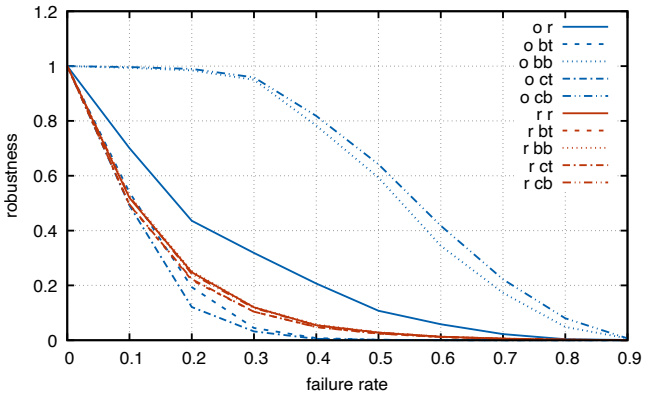


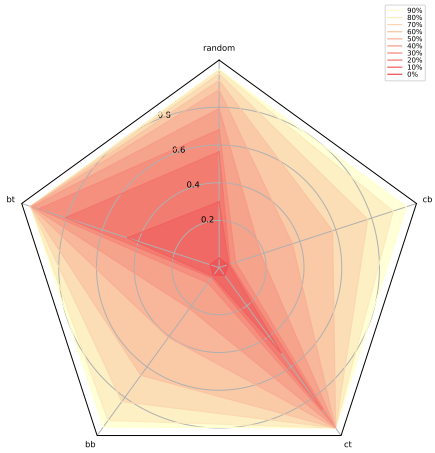
Figure 10: Information Access for the Manufacturing Email Exchange

As expected we can see daily patterns but overall certainly fluctuations around the equilibrium as our measure is sensitive to the amount of direct

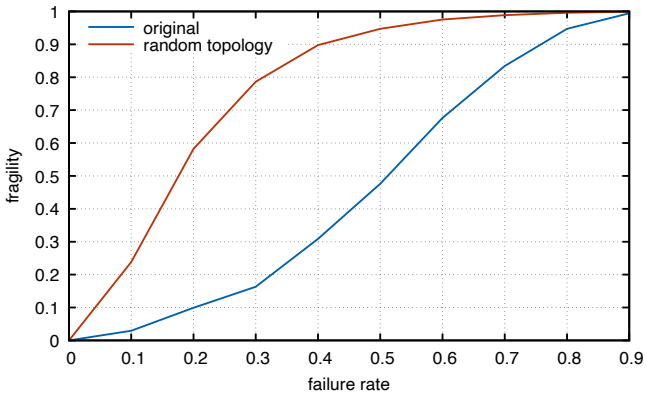
communication that happens.
Nodes in this network are employees fulfilling different positions. A company is a hierarchical structure. There exist a board, manager, and regular workers. At a given size, the manufacturing company employs 300 people in total from which we have a dataset containing 167; we expect separate departments that have hierarchical structures. Managers are being reported by a set of regular employees, and themselves report up the chain of command. We expect some nodes to be highly connected in an email communication network and some less. We expect significant differences depending on our attack strategy. Especially when removing nodes in their ranking bottom up. Nodes ranking low are less connected and are usually not mediating any communication. They are not vital to the functioning of the whole.
Figure 11 shows the resulting robustness analysis. As expected Figure 11a and Figure 11b show significant differences in robustness and fragility, given different attack strategies. We need to remove up to 30% of all nodes for the worst attack strategies to see effects in our measure. Surprising is the bad performance of the random topology, which again is near identical over different attack strategies. It performs almost as bad as the targeted attacks on betweenness and closeness centrality. This can also be seen in the stark contrast in normalized fragility in Figure 11a. This is in direct contrast to the prior Section 9.1 where we saw the random topology perform better over all attack strategies.
We can say that the manufacturing network is significantly more robust to loss of nodes when measuring Information Access then a comparable random network. As expected, some nodes have a more critical role when mediating IA between parts of the system. When we target these nodes, we have a significant impact on the system. This matches the organizational structure generally seen in such systems. As described in [33] there are 12 different departments in this company. Inter-department communication is usually mediated by a single higher-level individual. Our targeted attacks remove these nodes, and we disconnect departments from accessing information between one another.



(a) Random network comparison



(b) Vulnerability to different attack strategies



(c) Normalized fragility

Figure 11: Results for different robustness evaluations for the Manufacturing Email Exchange

9.3 Commercial Software Project

The commercial software Project dataset, stemming from [16], are co-edits extracted from a commercial software project. Development started in 2006, and it is still in progress. The number of developers over the whole course of the project is 17, with small variations.
Again we expect individual yearly variations when there is more communication, usually in January or less, usually in December with a potential break in the summer. As we do not have any information about the release cycle or other events in the time period, that is all we can expect.
Figure 12 shows the IA time series for the commercial software project dataset. The time resolution is one month. We can see fluctuations, some even dropping close to zero, and some maxima with up to twice the measured value. From 2015 the patterns settle in an equilibrium around a value of 100.

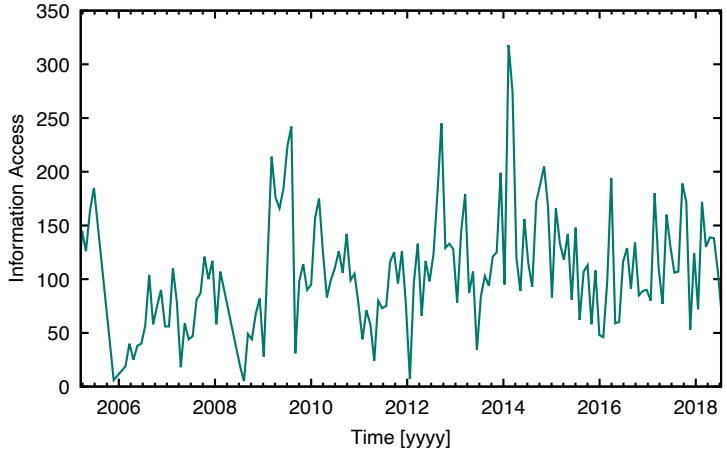
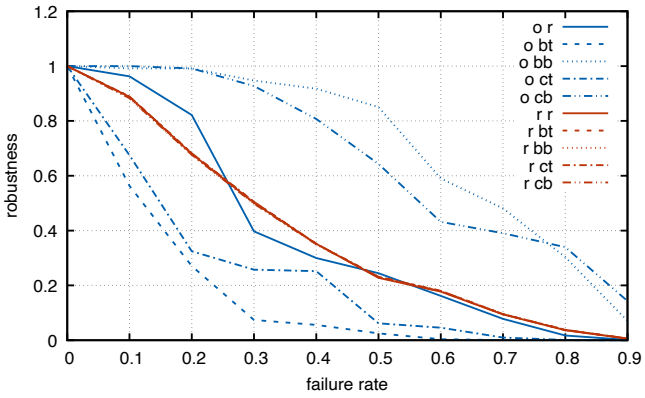


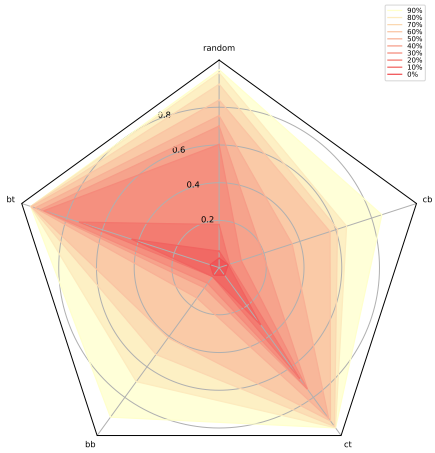
Figure 12: Information Access in the Commercial Software Project

What do we expect when we perturb this collaboration network? Independent of the organizational structure that is used, we still, even for a small team, expect a difference in the responsibility of developers. Some will interact with the project at a higher-level then others. We expect that removing nodes with targeted attacks will damage the system significantly more. We expect to see a better performance than a random topology, as this is a team that works in a commercial software project, and its incentive is to produce correct and working software. We assume that the team will take care of introducing new members while taking responsibility for leaving de-

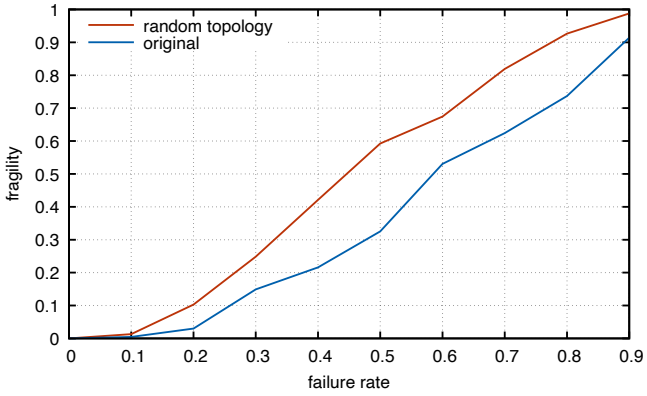
velopers. This should be seen as an effect on the co-editing pattern and as a consequence on Information Access.
Figure 13 shows the resulting robustness analysis. The difference in attack strategies is as expected. There are some interesting points where the robustness does not change when increasing the failure rate. This means that the network fully maintained its function. We also can see substantial differences in Figure 13a for attacking based on closeness centrality (the blue dashed and dotted line) and betweenness centrality (the blue dashed line). That means we have a few nodes in the system that have a small temporal distance to every other node in the system but are not as relevant when mediating information access. In other words, there exist different means of the network to achieve the same Information Access value. The random topology performs as well as the random attack strategy. Figure 13c shows that the random topology and the original network are not far apart in normalized fragility. We would have expected a more significant difference.



(a) Random network comparison



(b) Vulnerability to different attack strategies



(c) Normalized fragility

Figure 13: Results for different robustness evaluations for the Commercial Software Project

9.4 Igraph

Igraph is an open-source library for analyzing networks. We choose this as our first open-source project as it is in size and timespan, similar to the network from Section 9.3. It consists of 25 contributors and, at most in a single month, 14 developers. This project is also analysed in [16].
The corresponding GitHub page gives us some additional information about releases. There are six releases, the last one on December, 28th and the first one in March third, 2013. Table 9 shows all six releases.

Table 9: Releases for the Igraph project

Date	Release
28.12.2016	0.7.1
06.02.2014	0.8.0-pre
04.02.2014	0.7.0
28.10.2013	0.6.6
24.09.2013	0.5.8
03.03.2013	0.6.5

In the IA measure, we would expect these to be visible. In the months leading to the release, developers presumably are working more closely together. Looking at the contributors, we see that two authors create most commits with a long tail of contributors with 1 to 100 commits.

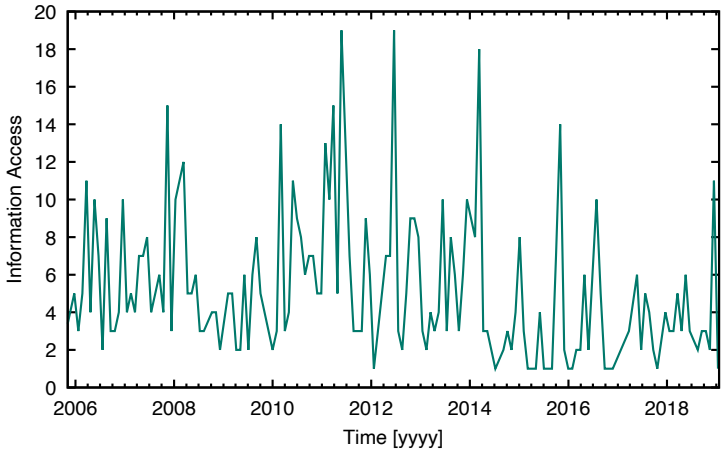


Figure 14: Information Access for the Igraph project

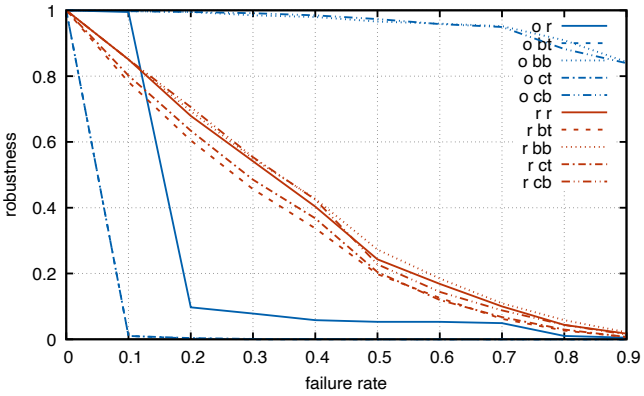
Figure 14 shows the resulting time series. We notice the absolute values of

our measure are almost an order of magnitude smaller than these in Section 9.3. Both projects have a similar number of contributors. Next, we see that for the first releases in 2013, there seem to be no increase in IA. The two releases in February of 2014 seem to coincide with a peak in our measure. The last release at the end of 2016 does again not show anything unusual in our measure. Overall we cannot conclude that releases are visible in our data. At only 4600 edges, c.f. Table 8, the absolute amount of co-editing is significantly smaller than in the comparable commercial project. As we are not controlling for lines of code, programming language, and other factors, we are not drawing any conclusions, but it seems that most of the project is depending on a few central contributors.

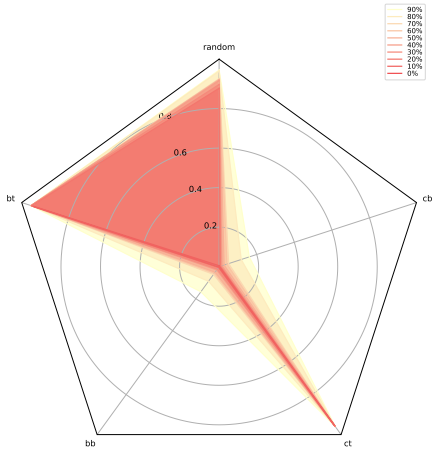
For a project that relies on two developers to maintain it over a long period of time, we expect that the IA values drop down very quickly for targeted attacks resulting in small robustness values. The converse implies that removing nodes that rank low in our targeted attacks has virtually no impact on our measure.

Figure 15 shows our robustness analysis. First as expected in Figure 15a we see that for our two optimal targeted attacks at the first failure rate of 10 percent the robustness drops to zero, meaning complete loss of functionality. Choosing the reveres attack strategy, even at a failure rate of 90 percent, does not strongly impair the system. The random strategy, solid blue line, seemed to have removed the critical nodes at the second failure rate. The random topology has an almost linear dependency on the failure rate. The different strategies are spread around 0.1 in robustness. This is due to the randomness in the topology. We can see the effect even better in Figure 15b, where all of the damage to the network is from three strategies.

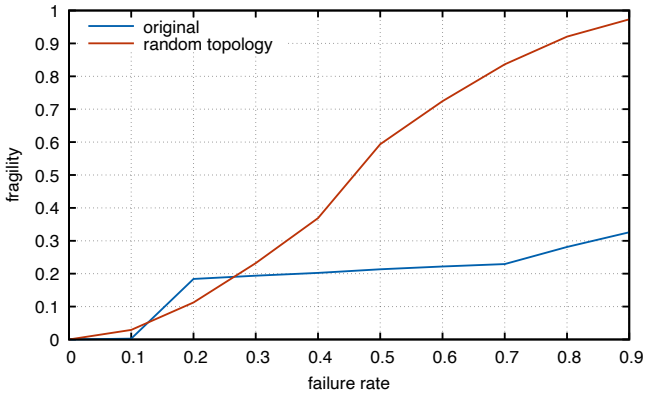
Figure 15c shows the resulting normalized fragility. Due to the heavy skew in node centrality and differences for the five attack strategies, it seems that for higher failure rates, the original network performs better than the random one.



(a) Random network comparison



(b) Vulnerability to different attack strategies



(c) Normalized fragility

Figure 15: Results for different robustness evaluations for the Igraph project

9.5 Libdav

Libdav is an open-source project providing tools to work with multimedia formats and protocols. It counts 496 contributors. We choose this to analyze a large and long-running open-source project.
Without any additional information, we are not able to make any prediction of how our measure will behave over time. Figure 16 shows the time series. In the time between 2006 until approximately 2015, the IA values are significantly larger than in the rest. We observe large peaks.

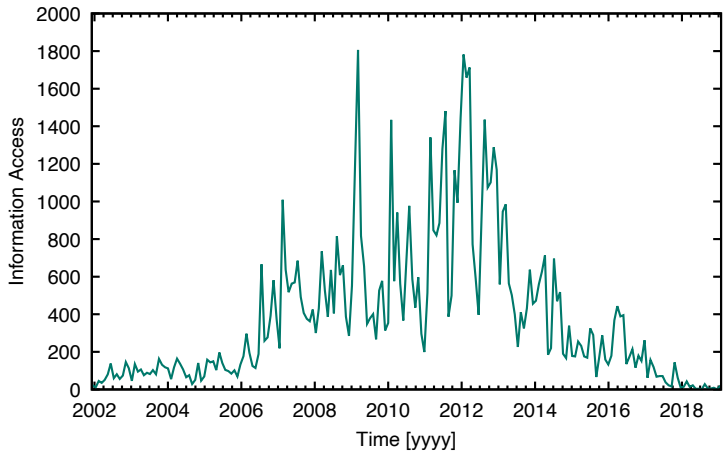
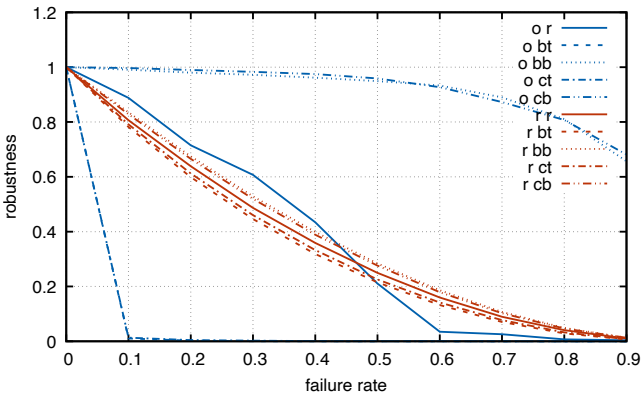
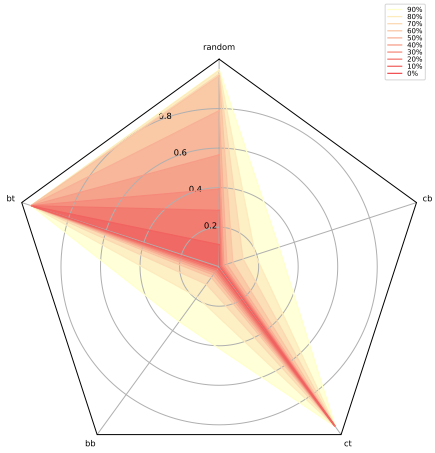


Figure 16: Information Access for LibDav

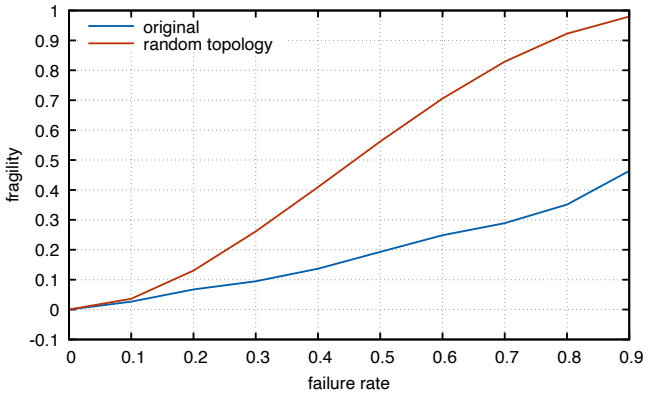
As in the last Section 9.4, we expect a significant vulnerability to targeted attacks. Again the reason for that is the usual distribution of authors contributing to an open-source project.
Targeting nodes following a betweenness or closeness centrality ranking results in a complete loss of function when removing 10 % of these nodes, c.f. Figure 17a. If we scale the failure rate up to 60%, we also can see that our robustness drops for both reverse attack strategies. Over all attack strategies, our original network performs better than the random one, c.f. 17c.
Overall it performs similar to the Igraph network, 9.4.



(a) Random network comparison



(b) Vulnerability to different attack strategies



(c) Normalized fragility

Figure 17: Results for different robustness evaluations for Libdav

9.6 OpenBSD

OpenBSD is an open-source UNIX-like operating system. Its official GitHub page states 78 contributors with 201542 commits, c.f. 8.
The random topology for this network leads to very long computing times. Due to the limited scope of this thesis, we did not include the comparison to the random topology. The reason for that is the construction of the random network. We uniformly draw edges for all timestamps. In a real-world system, the distribution is skewed, which reduces the memory footprint.
Figure 18 shows the resulting time series. We observe an increase in IA staring around the early 2000s. Up until 2012, the pattern is stable around an equilibrium of 4000, with peaks up to 12000. From 2012 through 2013, there appears to be little activity. From 2014 up until 2016, the IA increases to an equilibrium of around 5000. With the global maximum at the end of 2014.

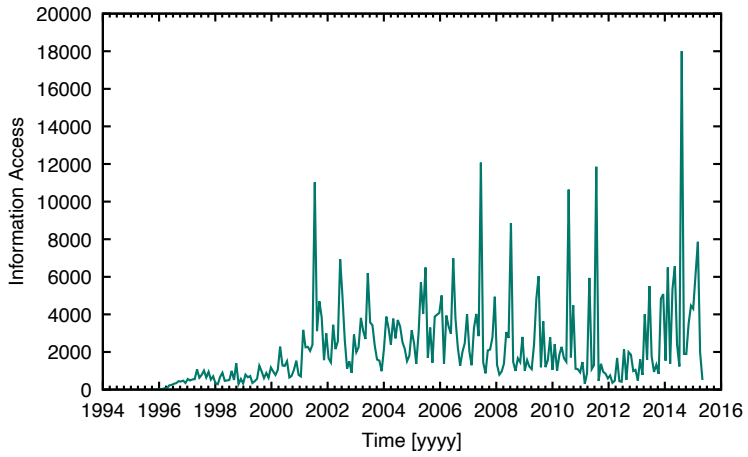
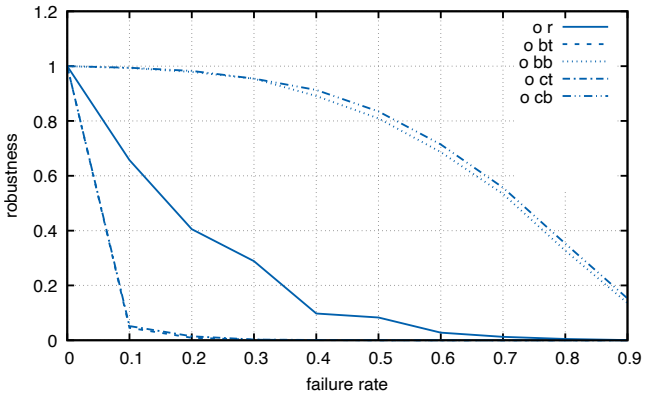
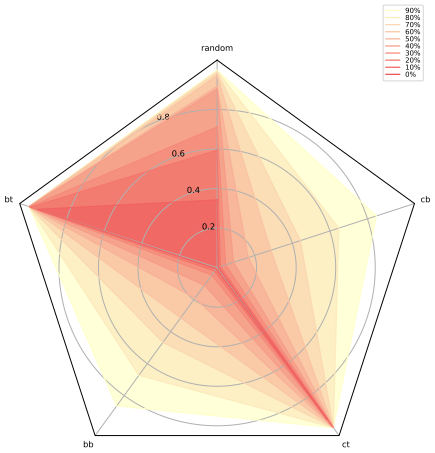


Figure 18: Information Access OpenBsd

In comparison to the Libdav project in the previous section, c.f. Section 9.5, we have fewer contributors but higher IA values. We expect to have more core contributors that are more integrated. That means that we expect the impact of different strategies to regress to the mean of a random attack. Again this is an open-source project, so there will be a long tail on contributors that have little commits and, therefore, little communication.
Targeting nodes by betweenness or closeness has still a large impact, c.f. Figure 19a, but there is non-zero robustness after removing the first 10 percent of targeted nodes. We can also see this in Figure 19b. At a failure rate



(a) Random network comparison



(b) Vulnerability to different attack strategies

Figure 19: Results for different robustness evaluations for OpenBSD

of 40 percent for the targeted attacks, we can already see an impact on the robustness.

9.7 Rust

The Rust programming language is a systems programming language focused on safety and speed. The implementation of all algorithms presented in this work is written in Rust.
The GitHub states 2553 contributors with 103015 commits. We extracted 480000 co-edits, c.f. Table 8. The rust project is maintained by a committee deciding on new RFCs, and we assume this to have an influence on the robustness. We expect a fairly strong group of maintainers and a long tail of contributors.
Its development started around 2011 with the 1.0 release in May 15, 2015. We expect a higher IA value for the time leading up to the release.

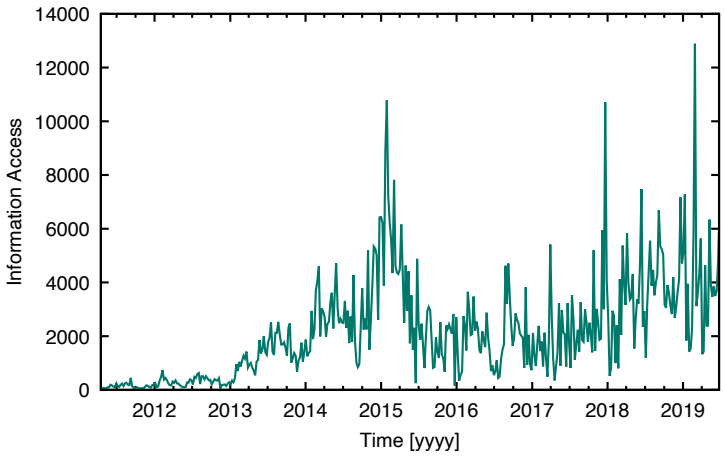
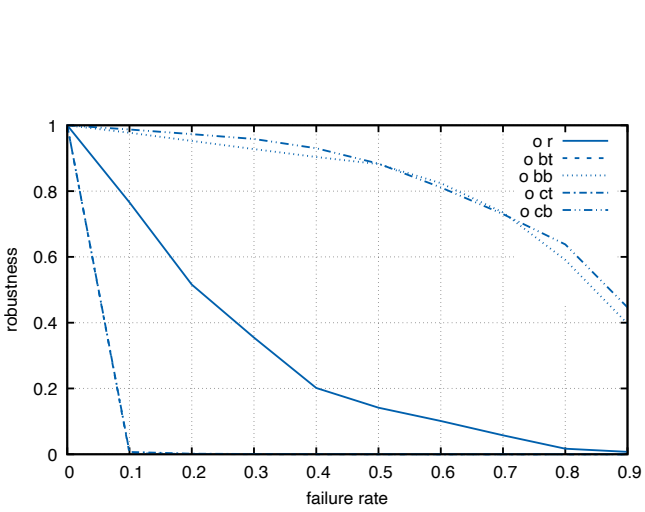
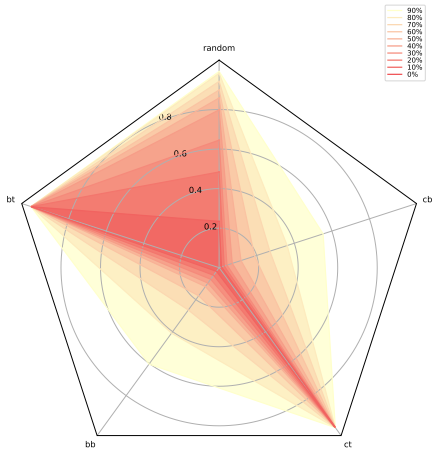


Figure 20: Information Access for the Rust Programming Language

Figure 20 shows the IA measure for rust. As expected, we see a sharp peak around the beginning of 2015. From this, up until the middle of 2017, the system appears to be in a state fluctuation around an equilibrium. From the end of 2017, we can see a transition into a higher equilibrium state.
With the largest number of nodes in all studies networks in this work we especially expect a significant vulnerability to targeted attack and no significant impact when using the reverse ranking to remove nodes.
Again for this network, the random topology was out of scope for our employed hardware, and we only analyze the original network.
Against our expectations, the impact of removing 40 % of the reverse ranked nodes has a measurable effect, comparable to the OpenBSD network, c.f. 9.6. As expected, removing 10% of the central contributors leaves no function



(a) Random network comparison



(b) Vulnerability to different attack strategies

Figure 21: Results for different robustness evaluations for Rust

system 21a. The random strategy lies between worst and optimal strategies, c.f. 21b.

9.8 Conclusion

We have analyzed seven different temporal networks and studied their robustness. In summary, we can conclude that the open-source projects are comparable more vulnerable to targeted attacks than commercial software projects. Interesting is our observation that the four open-source projects, albeit very different in size, show a similar response to our perturbations. Interesting is the difference of Igraph to the commercial software project. We can see that Igraph depends on mostly two developers to maintain the codebase. The commercial software project is less prone to single node failure. The London Tube Network shows a case where the random topology is more robust than the original network and validates that we may use a form of Information Access for different systems. The Manufacturing Email Exchange Network is an example that mostly validates our assumptions about how Information Access is related to communication and information exchange but also opens interesting new ways to study the robustness of communication networks in corporations. Using normalized fragility for the open-source project posed problematic, as the long tail in contributors skewed the fragility values.

10 Other Findings and Future Work

10.1 Weights

In this work, we were only concerned with unweighted links. In general, we can assume a measure that takes into account the amplitude of an interaction. Memory links connecting nodes to themselves in the future were assumed the same as direct interaction links. We might want to treat them differently, potentially weighting them less or more depending on our model.

10.2 Generalized Measure

A more abstract version of our measure would be the following: Instead of a unit function that indicates that a path with a given temporal distance exists between two nodes, we can think of an arbitrary function being integrated over the path. A simple extension would count the number of hops that happen over a path.

$$\text{Access}(t) = \sum_{i,j} \frac{\mathcal{T}(i,j,t)}{\text{dist}_{E(t)}(i,j)} \tag{10}$$

- $\mathcal{T}(i,j,t) \rightarrow \boldsymbol{D}$ being the access function along a causal-temporal, shortest path from i to j ending at time t.
- $\boldsymbol{D}$ being the domain into which the function maps.
- *dist* being the temporal distance along that path.

This allows us to model different categories of what is accessed or refine the definition of information access. This mechanism can also be used to change the rate at which information decays. Maybe we like to scale the memory mediated information access by some factor γ to simulate that actual interaction links are stronger.

10.3 Temporal Robustness

When studying robustness, we aggregated the time series data over the temporal dimension. This gave us a single value per network and failure rate. Robustness itself is a property of a system that evolves. There are times where we observe higher or lower levels of robustness. This is a natural next area to study.

10.4 Online source and sinks

Because of the limited scope of this work, we only looked at sources and sinks being files in the local file system. By the nature of our computational frameworks, we can extend this to use actual online sources, e.g., a continuous stream of co-editing edges. At the same time, we only considered ad-hoc perturbations. We defined a node ranking beforehand and then used it to compute the different failure rates. In a fully online system, we also need to compute perturbations dynamically. The ranking itself, temporal betweenness centrality and temporal closeness centrality, can also be computed using *Differential Dataflow*. This would also mean that the ranking computation itself would be dynamic, such that we would reevaluate the centrality rankings after removing nodes.

10.5 Algorithm for arbitrary addition and retractions

The algorithm in Listing 1 exploits the monotonicity of the problem of finding shortest temporal paths. When we want to perturb the system dynamically we must account for nodes and edges to be removed and not only added. The extension to incorporate this problem is straightforward in *Differential Dataflow*, but comes with a performance tradeoff in time and space complexity. As we compute the minimum of the temporal distance over all paths between all nodes, to account for nodes and edges to be removed we must in principal remember all previous minimum, such that if the current minima is no longer valid we can again evaluate our algorithm until fixpoint and produce the last shortest path.

10.6 Normalization

Our proposed measure is extensive, but we investigated different normalizations. A first natural normalization is done here [14] and here [8]. Adopting it leads to a measure similar to the metric called efficiency. We normalize with the combinatorial number of possible connections $N(N-1)$. This leads to a value between 0 and 1. The extrema mean either that between all nodes, there exists a path with a temporal distance of 1 or respectively, the network is fully disconnected. The problem this incurs is the variability of nodes themselves and that in networks with a large number of nodes, the normalization can diminish the actual measure value. Especially for long-tailed contributor distributions, as we have seen in Section 9.
In the case of studying collaboration networks from software development we can instead of normalizing by the combinatorial factor, we could take

additional information about code ownership into account and weight the importance of a node by its relative importance to the rest of the project.
At any given point in time, we know who owns what pieces of the codebase. This information can be extracted from GitHub, resulting in a stream of additions and retraction for authors. This, in turn, defines a set of **active** nodes that we should consider when looking at any network metric. This refines the combinatorial factor, because we only consider nodes that are currently part of the codebase.
Alternatively we can transform these nodes again into a set of **prioritized** nodes. These **hot** nodes consist only of nodes that share a given part of the whole codebase. E.g., the main developer. This solves the problem of long-tailed contributor distributions. On the other hand, it incurs a new problem. We need a parameter that dictates how much contribution a node must make in order to count as a hot or central node.

11 Conclusion

We motivated, defined, and discussed a measure, Information Access, that acts on the time-unfolded view of temporal networks. We implemented it in *Differential Dataflow* allowing us to leverage parallelism and instantaneous evaluation. We analyzed seven temporal networks with Information Access and measured their robustness against perturbation.
In closing, we want to summarize our findings and highlight novelties and improvements.

1. Information exchange is inherently dependent on causality. We proposed a measure that does not need a time window and thus more accurately describes the underlying system.

2. Our measure leverages shortest, temporal paths in a time unfolded network to compute Information Access, and we have validated it on small scale examples.

3. We designed a mechanism to model memory in a temporal network by introducing self-links of nodes with themselves in the future.

4. We implemented a recursive algorithm, computing the measure in an incremental setting, allowing for online analysis and monitoring.

5. We have shown the performance and multi-core scaling of our algorithm.

6. We analyzed seven temporal networks and have used Information Access as a measure for robustness.

We identified several areas for future work, especially around redefining robustness in a temporal setting, potential normalization for our measure, and proposed a generalized variant. We believe that we provided a strong foundation on which future studies of information exchange in temporal networks and its study for robustness can build.

References

[1] Ingo Scholtes, Nicolas Wider, René Pfitzner, Antonios Garas, Claudio J. Tessone, and Frank Schweitzer. Causality-driven slow-down and speed-up of diffusion in non-markovian temporal networks. *Nature Communications*, 5(1):5024, 2014.

[2] Ingo Scholtes, Nicolas Wider, and Antonios Garas. Higher-order aggregate networks in the analysis of temporal networks: Path structures and centralities. *The European Physical Journal B*, 89(3):61, 2016.

[3] Marcelo Serrano Zanetti, Ingo Scholtes, Claudio Juan Tessone, and Frank Schweitzer. The rise and fall of a central contributor: Dynamics of social organization and performance in the gentoo community. In *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 5 2013.

[4] Yulin Xu and Minghui Zhou. A multi-level dataset of linux kernel patchwork. In *Proceedings of the 15th International Conference on Mining Software Repositories - MSR '18*, - 2018.

[5] Ingo Scholtes, Pavlin Mavrodiev, and Frank Schweitzer. From aristotle to ringelmann: a large-scale analysis of team productivity and coordination in open source software projects. *Empirical Software Engineering*, 21(2):642–683, 2015.

[6] Hiroaki Kitano. Towards a theory of biological robustness. *Molecular Systems Biology*, 3(1), 2007.

[7] Xi Lu, Hongping Wang, and Yong Deng. Evaluating the robustness of temporal networks considering spatiality of connections. *Chaos, Solitons & Fractals*, 78:176–184, 2015.

[8] S. Trajanovski, J. Martin-Hernandez, W. Winterbach, and P. Van Mieghem. Robustness envelopes of networks. *Journal of Complex Networks*, 1(1):44–62, 2013.

[9] Hiroaki Kitano. Biological robustness. *Nature Reviews Genetics*, 5(11):826–837, 2004.

[10] Jose A. Fernandez-Leon. Robustness as a non-localizable relational phenomenon. *Biological Reviews*, 89(3):552–567, 2013.

[11] Genki Ichinose, Yuto Tenguishi, and Toshihiro Tanizawa. Robustness of cooperation on scale-free networks under continuous topological change. *Physical Review E*, 88(5):052808, 2013.

[12] Xiangrong Wang, Yakup Koç, Robert E. Kooij, and Piet Van Mieghem. A network approach for power grid robustness against cascading failures. *CoRR*, 2015.

[13] Nan Yao, Zi-Gang Huang, Hai-Peng Ren, Celso Grebogi, and Ying-Cheng Lai. Self-adaptation of chimera states. *Physical Review E*, 99(1):010201, 2019.

[14] Matthew J. Williams and Mirco Musolesi. Spatio-temporal networks: Reachability, centrality and robustness. *Royal Society Open Science*, 3(6):160196, 2016.

[15] Salvatore Scellato, Ilias Leontiadis, Cecilia Mascolo, Prithwish Basu, and Murtaza Zafer. Evaluating temporal robustness of mobile networks. *IEEE Transactions on Mobile Computing*, 12(1):105–117, 2013.

[16] Christoph Gote, Ingo Scholtes, and Frank Schweitzer. Git2net - mining time-stamped co-editing networks from large git repositories. *CoRR*, 2019.

[17] Raj Kumar Pan and Jari Saramäki. Path lengths, correlations, and centrality in temporal networks. *Physical Review E*, 84(1):016105, 2011.

[18] Jakob Runge. Quantifying information transfer and mediation along causal pathways in complex systems. *Physical Review E*, 92(6):062829, 2015.

[19] Peishi Jiang and Praveen Kumar. Information transfer from causal history in complex system dynamics. *Physical Review E*, 99(1):012306, 2019.

[20] Márton Pósfai and Philipp Hövel. Structural controllability of temporal networks. *New Journal of Physics*, 16(12):123055, 2014.

[21] Frank McSherry. Timely dataflow. `https://github.com/TimelyDataflow/timely-dataflow`.

[22] Nikolas Göbel. Optimising distributed dataflows in interactive environments. *ETH Zurich*, 2019.

[23] Frank Mcsherry, Derek G. Murray, Rebecca Isaacs, and Michael Isard. Differential dataflow.

[24] Todd J. Green. Datalog and recursive query processing. *Foundations and Trends® in Databases*, 5(2):105–195, 2012.

[25] Carlo Zaniolo, Mohan Yang, Ariyam Das, Alexander Shkapsky, Tyson Condie, and Matteo Interlandi. Fixpoint semantics and optimization of recursive datalog programs with aggregates. *Theory and Practice of Logic Programming*, 17(5-6):1048–1065, 2017.

[26] Ashish Gupta, Inderpal Singh Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data - SIGMOD '93*, - 1993.

[27] Timothy Griffin and Leonid Libkin. Incremental maintenance of views with duplicates. In *Proceedings of the 1995 ACM SIGMOD international conference on Management of data - SIGMOD '95*, - 1995.

[28] T. Griffin, L. Libkin, and H. Trickey. An improved algorithm for the incremental recomputation of active relational expressions. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):508–511, 1997.

[29] Frank McSherry. Differential dataflow. `https://github.com/TimelyDataflow/differential-dataflow`.

[30] Larry Page, Sergey Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web, 1998.

[31] Mehryar Mohri. Semiring frameworks and algorithms for shortest-distance problems. *J. Autom. Lang. Comb.*, 7(3):321–350, January 2002.

[32] Petter Holme, Beom Jun Kim, Chang No Yoon, and Seung Kee Han. Attack vulnerability of complex networks. *Physical Review E*, 65(5):056109, 2002.

[33] Radosław Michalski, Sebastian Palus, and Przemysław Kazienko. Matching organizational structure and social network extracted from email communication. In Witold Abramowicz, editor, *Business Information Systems*, pages 197–206, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Towards an Instantaneous Measure for Robustness of Temporal Networks

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Bach	David

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 05.12.2019	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.