aTLAS: a Testbed to Examine Trust for a Redecentralized Web

Valentin Siegert
Distributed and Self-organizing Systems
Technische Universität Chemnitz
Chemnitz, Germany
0000-0001-5763-8265

Mahda Noura
Distributed and Self-organizing Systems
Technische Universität Chemnitz
Chemnitz, Germany
0000-0002-5105-2463

Martin Gaedke
Distributed and Self-organizing Systems
Technische Universität Chemnitz
Chemnitz, Germany
0000-0002-6729-2912

***Abstract*—The redecentralization of the web introduces new challenges on trusting data from other sources due to many unknown or even hidden parties. An application working trustworthy in a decentralized web must evaluate trust and take trust-aware decisions autonomously without relying on a centralized infrastructure. This autonomy and the huge amount of available applications necessitates the web to be modelled as an open dynamic Multi-Agent System (MAS). To evaluate the trust of web agents, the most suitable trust models need to be identified and used. Despite the various trust models proposed in the literature for evaluating a web agent's trust, the examination of them with different scenarios and configurations is not trivial. To address these challenges, we showcase aTLAS, a Trust Laboratory of Multi-Agent Systems which is a web-based wizard testbed for researchers and web engineers to evaluate trust models systematically. aTLAS will enable future research regarding trust evaluations in a decentralized web. We show the different features provided by aTLAS through a set of experiments.**

***Index Terms*—Web Redecentralization, Trust, Multi-Agent Systems, Testbed**

I. Introduction

Current initiatives such as EU's Next Generation Internet[1], and Tim Berners-Lee Social Linked Data (SoLiD) [1] project have introduced the idea of *re-decentralizing the web*. Decentralizing the web enables increased data privacy so that every user has more control where his data is stored and who has access. One of the imminent challenges in web-redecentralization is enabling the means to make decentralized application trustworthy [2]. The entities in such an architecture need to decide which others to trust and which one not. A popular technique to deal with this challenge is trust management [3].

In the current centralized web, applications and consumers rely on centralized authorities to establish trust relationships and make trust-aware decisions. The centralized authority classifies third-party applications as trustworthy as long as predetermined artifacts are successfully validated or a human-given permisson is valid for this entity. However, traditional centralized mechanisms do not suffice given the characteristics of the decentralized web. Due to a change on data acquisition and processing, the data will no longer come from a limited set of sources but instead from arbitrary number of data providers as in SoLiD [1]. Further, a lot of these data will origin from an unknown or even hidden source. It is clear that, not all potential data providers can be known from the very beginning, as anyone can provide high valuable information and no structural authority establishes knowledge exchange for trust factors to evaluate trust for a certain information or within the context of the web a message. These distributed data providers and sources improve privacy and freedom of data [1], but also increase doubts regarding the data's possible malevolence and harmfulness.

A redecentralized web on the other hand requires each application to make *trust-aware decisions* and their respective *trust evaluations* [4] on its own. Such individual decisions and evaluations enables the web applications to handle trust autonomously. Moreover, a web application requires to establish dynamic trust relationships without any external authority mediating them. Due to the freedom of entities in a decentralized web and the high diversity of information in terms of e.g., topic, truth, harmfulness and malevolence, a trust decision requires to not only take the data source into account but also the current content and context [5]. A change of foreign data sources impacting any trust input dimension will always influence the trust even if a healthy trust relationship was already established regarding other content and/or in another context. Considering the large amount of data and the high traffic load in the web, such an autonomous dynamic trust handling can be a way to establish trustworthy web applications in a redecentralized web.

Due to the autonomousity, structural independence and the large number of web applications, the redecentralized web can be modeled as a Multi-Agent System (MAS) on a conceptual level as envisaged by the proposition of *hypermedia MAS* [6]. In MAS, a *web agent* (web application) can flexibily take autonomous trust decisions with the cooperation of others.

Further, the web could be specified as an open dynamic MAS since (1) no control mechanism exists on whether an agent is joining or leaving this MAS, (2) agents belong to different groups of interest, and (3) their decision making is independent of each other [4].

To enable web agents make trust-aware decisions autonomously, a crucial step is to identify the most suitable trust models. Despite the various sophisticated trust mechanics, scales

[1]https://ec.europa.eu/digital-single-market/en/policies/next-generation-internet

and models proposed in the literature for evaluating an agent's trust, the examination of them with different scenarios and configurations is not trivial. For instance, (1) the enviornmental conditions should stay the same for comparision, (2) scenarios require to be synthetic first and then also observered from practice, (3) a minor change in how to evaluate trust could lead in a complex scenario to immense different trust-aware decisions, and (4) different trust models should be applied in one scenario to fully exhaust the principles within the game theory [7] behind trust. In most cases, the researchers manually compare the trust models with each other or against their own solution which results in a high engineering effort and involves human judgement leading to unreliable evaluation. On the other hand, the existing tools for comparing trust models in MAS are either not related to the web, concerned with direct or reputation based trust and do not consider content- and context-related trust dimensions, or restrict trust comparison to only one agent in the MAS which makes it challenging to determine the appropriate models and limits the development of trust in a redecentralized web.

To remedy this issue, it is vital that researchers use a generic testing environment that is repeatable and supports the trust analysis of existing solutions. A testing environment provides insights to users and assists them to compare different models by experimenting with alternative trust dimensions and scenarios. Therefore, this paper shows our effort towards addressing this issue and contributes to improved repeatability and easier evaluation of trust models within web-related scenarios. We showcase aTLAS, a Trust Laboratory of Multi-Agent Systems which is an open-sourced web-based wizard testbed to support researchers to evaluate diverse trust models. aTLAS enables further research towards autonomous trust handling in a redecentralized web. The testbed is capable of spawning MAS scenarios, simulating content- and context-related trust dimensions, facilitating trust comparison among all web agents, and is independent from the trust models. Our contributions are as follows:

1) We present an automated and extensible solution to setup, run, and evaluate trust for MAS web scenarios which is independent of the employed trust model.
2) Both content and context trust dimensions are implemented in this testbed. To the best of our knowledge, this is the first trust testbed that takes these dimensions into account.
3) We implement a prototype of the testbed[2] and conduct evaluations with different configurations and scenarios to demonstrate its feasibility.

II. Related Work: Trust Testbeds

In most cases, the researchers in the trust community set up experiments and perform manual comparison with existing trust models, which is difficult to reproduce and interpret the results. Therefore, the community have considered the analysis and effects of different trust mechanics and models through the development of simulators and testbeds.

Some solutions like SuppWorld [8] and TRAVOS [9] built testbed environments adapted to the special features of their own trust models. Although such tools can effectively show the quality of the proposed models, they are restricted to specific dimensions of the trust model and cannot be used as a general purpose testbed. One of the most common but outdated testbed in the MAS domain is ART [10] which focuses on the development of a general purpose testbed. This testbed is based on the competition between the agents within one trust game and between the games themselves. As the ART testbed does not support general evaluation of trust technologies many researchers implemented their own testbeds to assess the performance of their trust models [4]. The TREET [11] testbed was inspired by the ART testbed to provide a more general-purpose experimentation of trust and reputation technologies. The authors state that TREET solves many of the shortcommings of the ART testbed such as enabling agents to randomly join or leave a simulation run, it is independent of the format of trust values and supports both centralized and decentralized trust models. Nevertheless, TREET is also domain specific and coupled to the marketplace domain similar to the ART testbed. In addition, the Alpha testbed (ATB) [12] evaluates trust models that have a decision making mechanism and ones that do not. However, ATB can only test the alpha agent while the other agents are simulated. In contrast, our solution can observe and compare several agents at a time to identify the behaviour of a group of agents.

Another testbed which is worth mentioning is DART [13] which employs the prisioner's dilemma approach. The agents are represented as a undirected graph where the nodes are the agents and the edges represent the communication between agents. Therefore, every agent keeps a neighbourhood list which represents its connections. Then the agents play different rounds of the game with the agents in their neighbourhood. However, approaching trust with the prisoner's dillema game has many disadvantages as stated by the authors of [14]. Agents have no opportunity to isolate untrustworthy opponents, since they are forced to interact with all competitors.

In summary, aTLAS is different from the existing testbeds in the following: (1) aTLAS is a generic framework that can be used with different trust models even if those models were designed for other domains. Some frameworks like DART are also domain-independent but most are focused on a specific problem and are not generic enough to analyze trust properties of a decentralized web. (2) Our solution can observe and compare several agents at the same time to identify how the behaviour of a group of web agents evolves in the decentralized web, unlike the ATB or some other testbeds. (3) Due to the open architecture design we can support multiple trust models per scenario as long as they are implemented and executable in aTLAS. (4) aTLAS support different trust scales, which makes it possible to also combine the output from trust models' evaluation results. (5) The integration of the web's open and dynamic MAS characteristics is not enrooted in none of the

[2]available at https://vsr.informatik.tu-chemnitz.de/projects/2020/atlas/

reviewed work. Yet, they are highly required for contentual and contextual trust evaluations on data in a decentralized web. (6) Most trust testbeds concentrate on evaluating the trust decisions, but for evaluation it is also important to be able to check the trust evaluation. The most prominent example from the literature also suporting this is ATB. (7) The majority of testbeds is not available online, aTLAS is available via its project page[2], as running demo and by code.

III. aTLAS Testbed

This section proposes a web-based wizard testbed, named aTLAS, a Trust Laboratory of Multi-Agent Systems. To ensure that the experiments are reproducible, aTLAS is publicly available on its project page[2]. In aTLAS, the redecentralized web is modelled as MAS in which an agent represents an individual web application that can have different business logics.

The testbed enables researchers to test web-related scenarios based on observations and examine trust. It provides a web user interface to allow the users interact with the testbed easily. In every evaluation run, one specific scenario is executed which involves running a MAS with all the involved agents. In this process, aTLAS simulates the applications involved in the scenario and the messages that are communicated between them by executing the observations. The observed messages are sent from an agent representing the sending web app in a certain observation to the receiving web app, again represented by an agent in the system. It is worth noting that the simulation of the applications' business logic is not required because the observed messages simulate their interaction.

A. aTLAS Architecture

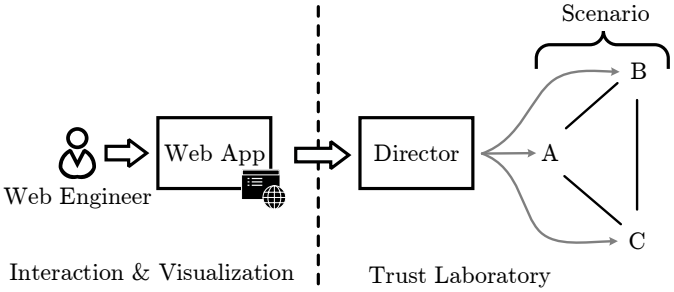

Fig. 1: aTLAS Big Picture

The big picture of aTLAS is shown in Figure 1 which consists of two main parts: *Interaction & Visualization* and *Trust Laboratory*. The web enginner interacts with the *Interaction & Visualization* part which is responsible for generating information about the agent's trust behaviour in a graphical and/or textual format via aTLAS Web App.

The *Trust Laboratory* consists of all the agents and a *Director*. The included agents, each agent's characteristics and the scnearios observations are described beforehand by the web engineer. The web engineer has the option to either describe an observed scenario based on practise or design one synthetically. The system also offers a set of predefined scenarios, which can be used as a basis of adaption towards the required scenario description. The Director acts as an interface for the runtime, and is responsible for preparing a MAS for the scenario run, monitoring the execution phase and also deconstructing the MAS. In the preparation phase it initialises the spawn of agents and takes care of setting up all other environmental conditions related to the scenario. During the execution it keeps track of agents' states with focus on their trust evaluations and trust-aware decisions and supervises the agents to execute the run as defined by the web engineer. After each scenario runtime the Director visualizes all gathered information about different trust values and relationships via the Web App for the web engineer and takes care of the MAS deconstruction. To reduce the waiting time until a first response is outputted, intermediate summaries of the current situation can be collected and shown to the web engineer.

For a realization of a distribution of agents on different machines as in a redecentralized web, the director has a general API for all hosts to register dynamically during runtime. To perform such a host registration, every applied host runs a Supervisor. Each Supervisor knows its maximal capacity, thus knows the number of agents possible to be spawned on its host machine in parallel. It works hence mainly as a Director proxy on each host machine to spawn responsively agents, supervise the agents on the host, collect their trust logs and takes care of the cleanup procedure on its host after each scenario run finished. Therewith it can spawn on any machine with an internet connection. Each supervisor requires at startup only some environmental information as where to find the Director, how much agents can spawn under its control and what its public adress is for establishing scenario run MASs.

Accordingly, each Supervisor has an interface to spawn and communicate with the agents, besides its connection to the director. The agents in turn have additionally two interfaces for ingoing and outgoing connections to other agents.

B. Scenarios in aTLAS

One component of aTLAS which connects all the concepts together is the scenario. An evaluation scenario represents a formal description of all the involved agents, their characteristics and the events that occur within the scenario run. In the following, we provide a conceptual model for an evaluation scenario in aTLAS.

Definition 1 (aTLAS Scenario): The aTLAS scenario S is an evaluation mechanic defined as a 7-tuple $S = \langle A, Sc, O, M, H, P, \mathrm{trace} \rangle$ where: A is the set of agents. Sc is the set of trust scales used by an agent. O is the observations which is a set of events with a certain content between two agents. M is the set of trust metrics used by the agents where each metric maps the triple agent, content and context to a real number of one trust scale. The triple can also exist with only one of its elements, e.g. a certain metric may only calculate the popularity of another agent without considering any content or context. H is the set of trust history descriptions per agent towards other agents. Each metric requires certain preferences according to its factors or dependencies. Each preference P maps a metric M of an agent to a certain setting, modelled

by the power set of the preference states $\mathbb{P}$. trace maps each observation to a timestamp.

$$O = \{\text{event} \mid \text{event} \in A^2 \times \text{Content}\} \quad (1)$$

$$M = \{m_{a,k} \mid a \in A,\ k \in \mathbb{N},\ m_{a,k} : A \times \text{Content} \times \text{Context} \mapsto \mathbb{R}\} \quad (2)$$

$$P = \{p(m_{a,k})\ \forall\ m_{a,k} \in M \mid p : M \mapsto \mathcal{P}(\mathbb{P})\} \quad (3)$$

$$\text{trace} : A^2 \times \text{Content} \mapsto t \quad (4)$$

While the trace mapping is described as a mapping to timestamps, the conceptual importance for aTLAS are not the exact timestamps but the sequentual dependencies between observations. To reproduce a scenario observed in practice, we require to keep the order of the observations as they happened. One order change could lead to a different outcome in trust evaluations and thus change the result of a trust-aware decision. A strict order of observations in a linear fashion without one timestamp being taken several times could on the other hand also lead to corruption in the context of closeness to the observed reality. Either the observations may have occured in parallel, or were not observed correctly or two observations were completely independent of each other in a scenario.

Due to this, the aTLAS scenario observations follows a soft order which means that multiple executions of a scenario may have a different final sequence of executed observations. This is realised with a reverse annotation such that each observation knows the other observations required to be executed before itself. With this knowledge, a dependency graph $D : (O, \text{dependency})$, is constructed, where the nodes are the observations and the edges represent the directed dependencies. To prevent dead locks during a scenario run, the dependency graph in a scenario requires to be loop-free.

C. aTLAS Process

An overview of the aTLAS *Trust Laboratory* process is illustrated as BPMN in Figure 2. The actors involved in this process are the *Director*, *Supervisors*, and *Agents*. The process consists of 8 main steps which are discussed in the following.

Step 1. Validate scenarios: Initially, the Director receives the aTLAS scenario from the Web App. The Director is then responsible to validate the scenario for runtime by: (1) assessing whether the available Supervisors have the capacity to run the scenario with respect to the required number of agents. The check fails if there is no free Supervisor, the total capacity of all available Supervisors is too low or the demanded number of agents per Host is not applicable to the capacity and availability of Supervisors. (2) the scenario is validated to resolve syntactical and consistency issues. The process will abort in case any of the checks fail. Upon sucess, the Director will create a distribution of agents per Supervisor.

Step 2. Distribute scenario: After the scenario is validated, all the Supervisors in the automatically created distribution receive the scenario from the Director.

Step 3. Setup agents: The Supervisor will setup the agents according to the configurations specified in the scenario. As the agents might spawn on different hosts, the supervisors establish an agent discovery via the director to identify the corresponding IP address and port number.

Step 4. Distribute discovery: The director gathers the Supervisors' local discovery and distributes it back to the agents.

Step 5. Supervise scenario: The director surpervises the scenario to ensure that the observations are executed in the correct order. It starts with a synced starting signal to all the agents, overwatches the scenario progression and ends the scenario run when all observations are executed. The end signal is required because all agents will have to run as long as all events specified in the Observations O are processed. Even though, they might have finished their observation executions, some may still be required for passing trust testimonies according to the requests of other interacting agents.

Step 6. Run scenario: After the start signal, each agent will execute according to the scenario configurations until the end signal is received from the director. As soon as an observation is done, the agent reports via its supervisor to the director, who distributes this event to all other agents. With this supervision, aTLAS provides a correct playback of the observations.

Step 7. Send logs: During the scenario run, the Supervisor gathers the logs of its spawned agents and sends updates regularly to the Director until the scenario is finished.

Step 8. Create report: In parallel with the scenario supervision, the Director continuously receives logs from all supervisors until the scenario finishes. For each executed observation, it will update the final report for the web engineer and push this update also to the web UI.

IV. Implementation & Evaluation

We implemented a prototype of aTLAS in Python, which is accessible via our project page[2] as live demo and by source code. The testbed is implemented according to a two-level architecture to seperate the business logic on each host from the director and the web app. aTLAS exploits Django Web framework for developing the web app and Django Channels framework for it's asynchronous support with websocket capability. While the director communicates with clients or supervisors via websockets, the agents communicate via TCP sockets to be closer to most web communication.

```
0  AGENTS = ['A', 'B']
1  OBSERVATIONS = [
2    {'observation_id': 1,
3    'before': [],
4    'sender': 'A',
5    'receiver': 'B',
6    'message': 'Redecentralization'}]
7  HISTORY = {'A': {'B': 1.0}, 'B': {'A': 0}
8  METRICS_PER_AGENT = {
9    'A': ['direct experience',
10         'popularity',
11         'recommendation'],
12   'B': ['direct experience',
13         'popularity',
14         'recommendation']}
```

Listing 1: Scenario Snippet

The scenarios are described according to a domain-specific language (DSL) which is realized with a Python configuration file. A snippet of a scneario is shown in Listing 1.

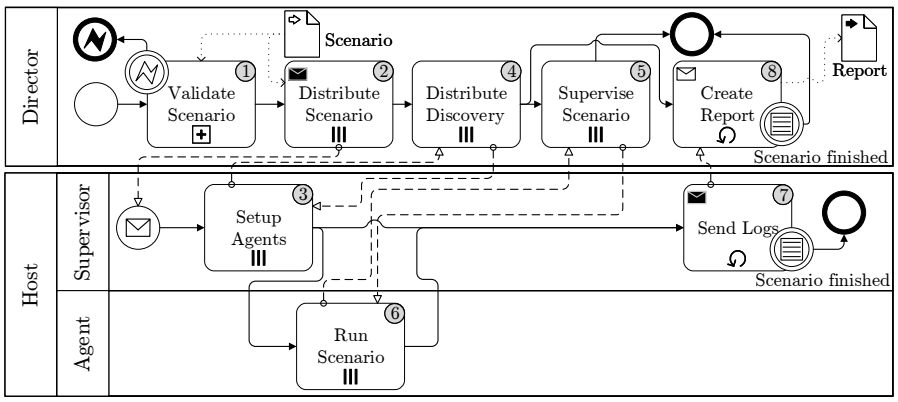

Fig. 2: aTLAS Process

In the following, we demonstrate the feasibility of the proposed aTLAS testbed using different scenarios and explore the scalability with different number of agents and observations.

A. Feasability

The first part of the evaluation aimed to evaluate the feasability of aTLAS for examining trust evaluation and trust-aware decision making. Three different scenarios are created to illustrate how different trust metrics impact the final trust value. For each unique scenario the process demonstrated in Section III is tested to analyze the behaviour of the approach on trust. These scenarios are availble at the running demo[2].

The different scenarios rely on a common setting with four agents {A, B, C, D} and are distinguished in the metrics used by them. We use the *direct experience, popularity, recommendation, authority* and *topic* metrics defined in [15]. Each scenario consists of six observations, where A first sends three messages to B and then C sends three other messages to B. The fourth agent (D) is included for recommendation purposes. Further, all agents in our scenarios rely on the trust scale from Marsh and Briggs [16], which ranges from -1.0 (full distrust) over 0.0 (ignorance) to 1.0 (full trust). In the scenarios' settings, we assume that all agents trust each other fully from the beginning, except B which intially does not know neither A nor C. Thus, all history values are 1, but B's history towards A and C is set to 0. To make a final decision upon trust evaluation, the final trust value is calculated as a weighted average of respective trust values per metric and agent. For demo purposes all metrics are weighted with 1.

Scenario 1: In the first scenario all agents use the direct experience, popularity, and recommendation metrics. Due to the very positive recommendation of D (for all agents 1.0) the trust values of A and C continuously increases. Since both agents have the same conditions, the trust values from B towards them increases per observation by the same amount.

Scenario 2: The second scenario extends the previous one by adding the authority metric. C therefore gets an authority status. In this case, even though C is an authority, B still has the same trust history towards C. With this status, C increases its trust at B faster than A.

Scenario 3: In the third scenario, we further add a topic metric. Accordingly, the topic *web engineering*, which is included in the other two scenarios is also considered for calculating the trust value. We assume that all agents fully trust each other on topic Web Engineering, except for B, which does not know A and C, and for that reason does not fully trust them on any topic. This means, that all topic trust settings are initialized with 1.0 for the topic Web Engineering, besides at B where A and C are trusted at this topic with 0.5. The metric is feeded with new values for each final trust value calculated for an agent as this was a new trust value for a certain topic. It interacts therewith a bit like a direct trust but dependend on the topic. The results show that it impacts the values for A and B differently. Both recieved 0.5 at the beginning, which gave them a boost at the first message send to B, but A is in the final value still under the 0.5 and thus the topic trust also drops slightly. C in comparision hits the 0.5 marker and thus can improve the topic trust value over time without decrease.

To summarize, the results verify that the aTLAS works for the different scenarios and can provide valid trust values depending on the different scenario preferences.

B. System scalability

One concern with respect to all testbeds is how they scale with the growing number of nodes and messages. Therefore, we explored the scalability of aTLAS to show its performance depending on the number of agents and observations.

The experiments were carried out on two PCs in a private LAN. PC 1 is a Windows 10 PC with an Intel Core i7 8700K CPU, 16GB DDR4 RAM and a NVIDIA GeForce GTX 1080 Ti connected via cable. The second PC (PC 2) is a MacBook Pro A1990 (EMC 3215) with Core i9 connected via Wifi. During the experiment, each PC runs one supervisor and each supervisor takes half of all agents involved, e.g. one scenario might involves 10 agents, hence each supervisor sets up 5.

To test the scalability we take the first scenario of the feasability tests as a basic scenario and create new scenarios with an upscale in either the number of agents or observations. For each experiment, the basic scenario is adapted to increase the number of agents or observations, thereby we used the

amounts of 10, 20, 50, 100, 200, 500 and 1000. The process of adding agents does not change anything regarding the trust values in the scenario, but are only initialized and later cleaned up again. Other tests have been performed to upscale the number of observations in the basic scenario. The worst case for the runtime of observations is used, which is a long chain of observations depending always on the one before and that the supervisor changes for each observation. Therewith the highest waiting time for each supervisor is added as it has to wait for the other one to finish an observation and for the director to communicate this before it can resume.

For each scenario, we measure three different times: *preparation, execution* and *clean up*. The preparation and execution time is the time it takes to perform step 1-4 and 5-8 respectively in the aTLAS process and the clean up time is an implemented logic to dismantle the created MAS and free capacity at the supervisors. For better precision of our measurment, we run each testing scenario 10 times and calculate for each of the three times the average and visualize the results in Figure 3. Fig. 3a compares the different time with respect to different agents and Fig. 3b shows the corresponding time for an increase number of observations. In Fig. 3b, the preparation plot is not visible because the clean up plot covers it completely. This happens as both of these measurements are as an average below 1 second whereas the Execution time is significantly higher.

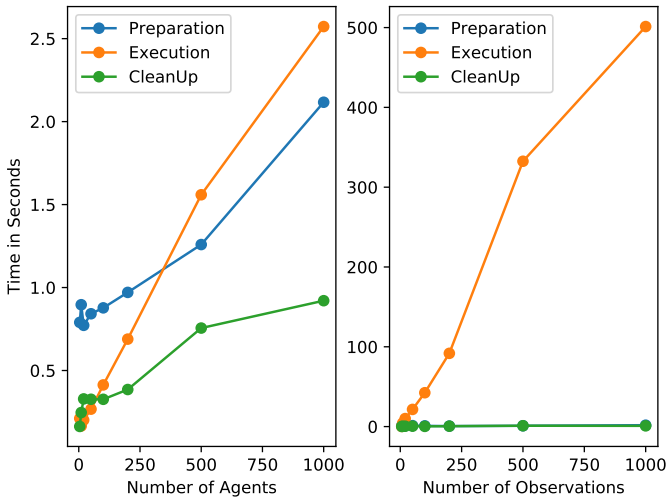

Fig. 3: aTLAS Scalability

The results show that the agents do not increase the timings as much as the observations do for the execution. The overtaking of the execution time at the upscaling of agents is attributable to the long list in each lookup where agents are the key. This happens often due to discovery purposes. Overall the timings underline the scalability of aTLAS, as the only long waiting time is coming at a long chain of badly connected observations. Even in such a worst case aTLAS executes 2 observations per second in average.

V. Conclusion

In this paper, we presented a web-based wizard testbed named aTLAS[2] for examining trust in a decentralized web. The main contributions of aTLAS are its web relation and openness for several trust scales, metrics and models. In addition, it supports examining both contentual and contextual trust dimensions. The evaluation prooved its feasability and scalabilty.

In future work, we want to improve the current prototype by considering the following aspects: (1) The UI should allow defining scenarios, (2) the trust examination results should be presented to the web engineer using a graphical visualization, (3) and to support more trust models, metrics and scales, an interface for uploading those on the fly would improve the systems examination power.

References

[1] A. V. Sambra *et al.*, "Solid: A platform for decentralized social applications based on linked data," Technical report, MIT CSAIL & Qatar Computing Research Institute, Tech. Rep., 2016.

[2] L.-D. Ibáñez *et al.*, "Redecentralizing the web with distributed ledgers," *IEEE Intelligent Systems*, vol. 32, no. 1, pp. 92–95, 2017.

[3] J. Sabater and C. Sierra, "Regret: reputation in gregarious societies," in *Proceedings of the fifth international conference on Autonomous agents*, 2001, pp. 194–195.

[4] H. Yu *et al.*, "A Survey of Multi-Agent Trust Management Systems," *IEEE Access*, vol. 1, pp. 35–50, 2013.

[5] V. Siegert, "Content- and Context-Related Trust in Open Multi-agent Systems Using Linked Data," in *International Conference on Web Engineering*. Cham: Springer International Publishing, 2019, pp. 541–547.

[6] A. Ciortea *et al.*, "A Decade in Hindsight: The Missing Bridge Between Multi-Agent Systems and the World Wide Web," in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 2019.

[7] D. G. Mikulski *et al.*, "Trust dynamics in multi-agent coalition formation," in *Unmanned Systems Technology XIII*, vol. 8045, Orlando, 2011, pp. 221–267.

[8] J. Sabater, "Evaluating the regret system," *Applied Artificial Intelligence*, vol. 18, no. 9-10, pp. 797–813, 2004.

[9] W. L. Teacy *et al.*, "Travos: Trust and reputation in the context of inaccurate information sources," *Autonomous Agents and Multi-Agent Systems*, vol. 12, no. 2, pp. 183–198, 2006.

[10] K. K. Fullam *et al.*, "A Specification of the Agent Reputation and Trust (ART) Testbed: Experimentation and Competition for Trust in Agent Societies," in *Proceedings of AAMAS '05*. New York, New York, USA: ACM Press, 2005, pp. 512–518.

[11] R. Kerr and R. Cohen, "Treet: the trust and reputation experimentation and evaluation testbed," *Electronic Commerce Research*, vol. 10, no. 3-4, pp. 271–290, 2010.

[12] D. Jelenc *et al.*, "Decision making matters: A better way to evaluate trust models," *Knowledge-Based Systems*, vol. 52, pp. 147–164, 2013.

[13] A. Salehi-Abari and T. White, "Dart: A distributed analysis of reputation and trust framework," *Computational Intelligence*, vol. 28, no. 4, pp. 642–682, 2012.

[14] A. A. Adamopoulou and A. L. Symeonidis, "A simulation testbed for analyzing trust and reputation mechanisms in unreliable online markets," *Electronic Commerce Research and Applications*, vol. 13, no. 5, pp. 368–386, 2014.

[15] Y. Gil and D. Artz, "Towards content trust of web resources," *Journal of Web Semantics*, vol. 5, no. 4, pp. 227–239, Dec 2007.

[16] S. Marsh and P. Briggs, "Examining Trust, Forgiveness and Regret as Computational Concepts," in *Computing with Social Trust*, J. Golbeck, Ed. London: Springer, 2009, pp. 9–43.