P2P Petition Drives and Deliberation of Shareholders

Marius C. Silaghi[1] and **Song Qin**[1] and **Khalid Alhamed**[1]
Toshihiro Matsui[2] and **Makoto Yokoo**[3] and **Katsutoshi Hirayama**[4]
[1]Florida Tech, [2]Nagoya Inst. of Technology, [3]Kyushu Univ, [4]Kobe Marine Univ.

Abstract

We introduce protocols for a fully decentralized synchronization of data items among equal peers in an argumentation framework (for applications such as debates in *petition signature drives*, and *deliberation among shareholders*). Here, an argumentation is a structured exchange of *justifications* around issues (or motions). The *motions* are relevant to organizations, and the *organizations* are rules for constituents. A *constituent* has a certain voting weight (e.g., based on her shares in the organization). A *peer* is a group of agents representing a user. Each peer separately stores all the data items of interest to her.

To achieve data coherence at convergence for connected sub-graphs in such a framework we introduce a method for generating unique identifiers for peers, organizations and their constituents, as well as for data items generated by them. This avoids conflicts between semantically distinct data.

Social networks are very successful applications but few have proven or even formalized their properties. Our proposal is a framework and protocol for a new type of P2P Social Network, where peers can congregate into fully decentralized virtual constituencies mirroring externally existing organizations. The membership in such organizations is based on constituents witnessing about each other. A reference implementation in Java is made available and can be used to validate claimed basic functionality.

Introduction

Social networks have emerged into one of the most successful applications of computer science. Significant research has focused on studying the complex relationships captured in data recorded by such systems. However, the most famous problems relate to the fact that their administrators have the possibility to not meet expectations in terms of privacy, censorship and balance in influence through advertisements and media.

Here we undertake the task of formally defining a particular social network application, its desirable properties, and to provide protocols that can be proven to meet these expectations. The result is validated both theoretically and using a reference implementation.

The application we address consists in the fully decentralized creation and management of virtual constituencies for existing organizations.

We view an *organization* as an immutable set of foundation rules for deliberating on a set of issues and for defining the decision makers. Examples of organizations that can have virtual counterparts are: a company, a department in a university, a neighborhood, a club, a country, an ethnic group.

A *constituency* is the body of all the voting members of an organization. The functioning of a constituency, such as the frequency of its meetings, the identity and weight of the votes of each of its members, and the privileges and limits on the scope of their voting powers, are defined by the statutes or constitutions of the organization. While the relevant issues in a statute of a virtual constituency may be different from traditional ones, they are encoded in the *parameters* defining the virtual organization.

Not all the eligible constituents of a society may be using software to be active on a virtual constituency. Moreover, some persons who are not eligible constituents may subscribe to the argumentation as observers, can help in dissemination and storage of replicas, can even contribute *observer arguments and motions*, or maliciously try to attack the deliberations.

The main concrete application we foresee are debates in *petition signature drives*, and *deliberation among shareholders*. The addressed application is related to deliberation techniques described in Robert's rules of order (Roberts 1989). From these rules we employ the concept of *motion* as a formal proposal submitted to vote. While within common Internet fora the arguments are threaded in a tree structure where each one answers at most one prior argument, the addressed application defines arguments as *justifications* of one's vote. As such, their main classification is by the vote being justified.

For approaches to this application we identify, define and address the following potential *properties*: equal opportunity (of initiative, management, and voting), non-repudiation of sender, repudiation of false attribution, coherence, self-interest, soundness, scalability, grassrootedness, verifiability, relevance, openness, and

resilience (from censorship and attacks).

We propose here a *protocol* for exchanging messages such that data is synchronized efficiently between peers found in long-term connections as well as with peers that have recently joined. The protocol is described as a set of message definitions together with a recommendation of how these messages should be processed. The protocol is expected to be robust or gracefully degrade in the situation where peers do not react according to the recommendation.

A database schema was developed and a reference implementation was made available on an open source code repository. The implementation validates the capability of the protocol and schema for achieving some of the desired properties (function behind NATs, seamless bootstrap of organizations and deliberations, correct synchronization in a dynamic set of peers).

First we introduce the related work and main background knowledge. Then we formalize the employed concepts. Subsequently we define the desirable properties expected from the designed protocols, based on the motivating application. We then detail the proposed protocol in terms of messages and their semantic. We also detail the algorithms used in the implemented system to handle and generate messages respecting the described protocol. Some experiments with the system are detailed in (Qin *et al.* 2013). A theoretical analysis is performed proving that some of the introduced desirable properties are achieved by the new protocol. Most other defined properties are shown to be reasonably handled.

Background

Identification over the Internet has been addressed from a variety of perspectives (Douceur 2002; Yahalom, Klein, & Beth 1993). Skype is a successful P2P social system where the main motivation of the P2P structure is efficiency (Ford & et al. 2005).

Several P2P Social Network Systems like OStatus (Status Network), and Diaspora (Federation Protocol) are being currently developed by various volunteer teams as a reaction to the privacy concerns raised by popular providers like Facebook and Twitter. OStatus is developed by a W3C group, but it did not yet release any document. Its stated scope is the interaction between social networks, rather than interaction between end-users. Like OStatus, the Federation protocol is meant for collaboration between providers. (W3C ; Federation). Deliberation of stakeholders is discussed in (Fedoruk 2006).

P2P Systems P2P systems have been used for a multitude of applications, such as efficient videoconferencing (Skype (Xie & Yang 2007)), efficient downloads (Bittorrent (Cohen 2003)), file sharing, and ad-hoc routing. Besides efficiency, the P2P paradigm can also bring about decentralization and an inherent fair (or democratic) structure. The best known prior usage of such decentralization has so far been at the limits of legality (Nieminen 2003; Gu, Jarvenpaa, & others 2003). However, this decentralization can be put to more popular tasks, such as ad-hoc routing and the application suggested here. The P2P social network infrastructure PeerSon for private exchange of data is proposed in (Buchegger *et al.* 2009). They use emails or public keys hashes as global identifiers. Like Skype, they differentiate between peer and agent concepts, and solutions for handling multiple agents per peer. The P2P Java platform JXTA also provides needed NAT support, but the peer groups are centralized by a parent group structure.

Concepts

Now we detail the concepts used here, and implemented in the *DirectDemocracyP2P (DDP2P)* system: `github.com/ddp2p/DDP2P`. To uniquely identify items exchanged by the protocol, we design methods for generating *Global Identifiers (GID)* that are supposed to uniquely identify each semantically independent piece of information. When a piece of information can have several versions (like several votes of a participant on the same issue at different moments, where the newest vote overrides older votes), then the design is such that the GIDs of all versions are identical. The following definitions make use of the function $HASH(M)$ which is a secure hash function (such as SHA-256). Given a public key $\mathcal{P}$, the function $SK(\mathcal{P})$ is used to indicate the secret key associated to $\mathcal{P}$. The function $SIGN(\S, m)$ returns a secure digital signature of the message m with the secret key $\S$. The function $VERIF(\mathcal{P}, m, \sigma)$ returns the Boolean result of the verification of the digital signature σ for the message m using the public key $\mathcal{P}$.

As discussed further, the GID of an item is sometimes built using the $HASH()$ function on some parameters. In that case those parameters are not allowed to change. When the item is a virtual representation of a human person (e.g., of a peer, constituent, authority), then we use her public key as her GID. Each time that the public key has to be revoked, all the items she generated and signed with her public key have to be regenerated by her agent using her next public key. Note that, when communicating GIDs based on public keys for purposes other than signature verification (e.g., references), then we exchange $HASH(GID)$, to reduce the message size.

Peers and Agents Most users are familiar with software agents for tasks such as communication over Skype and management of emails (email agents). The concepts of peer and agent are not new but here we need to be specific as to the exact definition used here. A software agent acts on behalf of its owner by communicating with remote servers and maintaining local data.

The concept of peer is used in the community of peer-to-peer computing referring to a software that is simultaneously a server and also a client to other similar

servers. While not all the peers have to function identically, many systems strive to achieve such a behavior.

Formally, a **peer** is specified by a tuple $\langle \mathcal{P}, p, a, d, \sigma \rangle$ where $\mathcal{P}$ is the GID of the peer selected as its public key, p is the set of properties declared by the peer, such as name and email, and d is the date and time when p was declared. A signature σ is computed as: $\sigma = SIGN(SK(\mathcal{P}), p)$. A set of Internet addresses a is also specified with the peer and is not signed, as it may contain any IP address ever used by this peer and known attacks on it does not significantly disrupt the protocol. Since a peer is identified by a public key, it can be represented by a group of software agents working for the same user (e.g., on different devices) and using the same public key. Similarly, a peer can be the input point for data from a set of users (for which the main user has proxy rights of representation).

Each peer separately stores all the data items of interest to her, and can share them with her direct connections. Even as current experiments only test local storage (which can pose challenges for devices with limited memory), a peer can store its data in the cloud.

Organizations, Constituencies, Majority voting Countries and businesses are organizations. While in some countries there are large differences in the way a typical business is run versus the way the country is run, most governing mechanisms are encountered for both businesses and countries (ranging from democratic companies to privately owned countries) (Kagawa 1936; K. Raaflaub & Wallace 2007). In referendums the constituency can vote on pre-established choices for an issue but cannot select other choices or raise new issues. The raising of new issues by constituency members is sometimes made available via different mechanisms: the *petition* or *citizen initiative* processes.

In some systems, the constituency is required to transfer its power to proxies via an election, and these proxies are the ones running the daily activities. Function of the type of organization, the proxies are called: mayor, congress, presidents, committees or board.

As per the above competing models, we support:

- *authoritarian organizations* that are managed by an authority to make decisions as to who is a constituent of the organization (potentially the elected representative/proxy for these decisions).
- *grassroot organizations* where no user has any particular rights or delegation and each and every decision (including on the acceptance of each candidate to be a constituent) is always taken directly by the whole constituency by vote. In this case, even when they see the same data, constituents may draw different conclusions as to the composition of the constituency.

Formally, we define a **grassroot organization** as a tuple $\langle \mathcal{O}, p \rangle$ where $\mathcal{O}$ is the GID of the organization and p is a set of parameters describing it and its constitution/statute. For grassroot organizations,

$$\mathcal{O} = HASH(p).$$

An **authoritarian organization** is a tuple $\langle \mathcal{O}, p, r, d, \sigma \rangle$, where its GID $\mathcal{O}$ is the public key of the authority that controls p and decides the composition of the constituency, r is the revocation status of $\mathcal{O}$, d is the date and time when these parameters were set by the authority and σ is the signature of $\langle p, r, d \rangle$ with the secret key associated to $\mathcal{O}$:

$$\sigma = SIGN(SK(\mathcal{O}), \langle p, r, d \rangle).$$

We handle two types of constituents:

- *Active constituents* are data items representing the users that created these items and that can generate other data items, that they digitally sign. We say that the active constituent has generated these items.
- *Inactive constituents* are data items representing a person other than the one generating it, and cannot generate other items itself. They may stand for sick, elderly or other persons that are not able to participate themselves in such a process but should be counted when one makes decisions.

The formal definition of an **active constituent** is a tuple $\langle \mathcal{C}, \mathcal{O}, i, d, r, \sigma \rangle$ where $\mathcal{C}$ is the GID of the constituent, $\mathcal{O}$ is the GID of the organization of the constituent, i are the identity details of the constituent (name, email, etc.), and d is the timestamp when i was declared. $\mathcal{C}$ is specified as a public key, r is the revocation status of $\mathcal{C}$, and the constituent data is signed with $\sigma = SIGN(SK(\mathcal{C}'), \langle \mathcal{C}, \mathcal{O}, i, r \rangle)$. An **inactive constituents** is a tuple $\langle \mathcal{C}, \mathcal{O}, i \rangle$, whose GID is $\mathcal{C} = HASH(\mathcal{O}, i)$.

Motion, Justifications While the currently entrenched practice on the Internet fora is to provide threaded discussions around articles, one of the most studied and practiced argumentation mechanism for deliberation by constituencies is defined by Robert's rules of order (Roberts 1989). Robert's rules of order pivot around the notion of *motion*. A motion is a formal proposal around which arguments take place and that is eventually voted on. Several Robert's rules of order (like tabling a motion) are arguably irrelevant to Internet venues where participation is asynchronous, but we find that the general concept remains the most appropriate for a virtual constituency.

Formally, a **motion** is a tuple $\langle \mathcal{M}, t, o \rangle$ where $\mathcal{M}$ is the GID of the motion, t is its text, and o is a list of possible answers. The GID is computed as:

$$\mathcal{M} = HASH(t, o).$$

In our motivating application, the arguments are not just answers to other arguments but they are justifications of a vote. We start from the principle that information is relevant only if it can be retrieved efficiently (i.e., if it is structured). As such, the main classification of the justification is given by the vote answer submitted by its supporters. Since each user has a single vote for a motion, she can support a single

related justification. We note that the same justification can be used for different answers. For example, given the motion: *Should a tunnel be built under the Gibraltar straits?* with possible answers *Support* and *Oppose*, both possible answers could be justified with: *It would make the travel safer for African immigrants to Europe!* For pro-immigration minded people, this may be a justification for a tunnel, and for anti-immigration minded people this may be a justification against its construction.

Formally, a **justification** is a tuple $\langle \mathcal{J}, \mathcal{M}, t \rangle$ where $\mathcal{J}$ is the GID of the justification, $\mathcal{M}$ is the GID of a motion, and t is the text of the justification. The GID $\mathcal{J}$ is computed as:

$$\mathcal{J} = HASH(\mathcal{M}, t).$$

A **vote** is a tuple $\langle \mathcal{V}, \mathcal{M}, c, \mathcal{C}, \mathcal{J}, d, \sigma \rangle$ where $\mathcal{V}$ is the GID of the vote, $\mathcal{M}$ is the GID of a motion, c is the answer selected (one of the values in the element o of the motion), $\mathcal{C}$ is GID of an active constituent, $\mathcal{J}$ is the GID of the justification (potentially empty), and d is the date and time of the vote. The GID $\mathcal{V}$ is computed as:

$$\mathcal{V} = HASH(\mathcal{M}, \mathcal{C}, \mathcal{J}).$$

and the digital signature σ is computed as:

$$\sigma = SIGN(SK(\mathcal{C}), \langle \mathcal{M}, c, \mathcal{J}, d \rangle)$$

Desirable Properties

Desirable Properties Given the target application, management of constituencies, we define and address the following properties, that are desirable for solutions:

- **equal opportunity of initiative:** any peer/constituent has equal opportunity to start a virtual constituency (various instances of virtual constituencies for the same external organization can be used simultaneously for their different properties, or users can converge naturally towards the most used instance, just as currently most users converge towards a few social networks despite the existence of many alternative platforms).
- **equal opportunity of management:** any peer/constituent has equal decision power and priority in defining who is an eligible member of that constituency (the aggregation of thereof is outside the scope of this article).
- **equal opportunity of voting:** any peer/constituent has equal weight and priority in submitting motions, justifications and in voting.
- **non-repudiation of sender:** the peer/constituent generating an item can be identified.
- **repudiation of false attribution:** it is impossible to falsely attribute an item to another peer.
- **coherence:** a total order should exist between items with similar semantic (e.g., vote, version of the profile of the same constituent or of the same organization), to ensure that all peers coherently store the same one.
- **self-interest:** each peer only needs to store the data of interest to her, and only helps disseminating the data that she wants to support.
- **soundness:** each peer eventually gets the data of interest for her.
- **scalability:** each constituent only needs to communicate with a limited number of peers.
- **grassrootedness:** each constituent only needs to be known and supported by a limited number of other constituents.
- **verifiability:** the identity of virtual constituents one knows is verifiable with reasonable effort (using confirmation via email, phone numbers, or visit to an address).
- **relevance:** relevant argumentation items are visible and distinguishable from spam (e.g., using collaborative filtering or intelligent classifiers).
- **openness:** if any number of peers fail, leave, or join, the remaining ones are able to continue their deliberation without lossing data generated by themselves and with a coherent view of data generated by disappeared peers.
- **resilience from censorship:** no peer/constituent can block the dissemination of data between other directly connected peers.
- **resilience from attacks:** the supported properties hold even in the case of active attackers (independently of whether the software of other users follow the desired protocols or not).

Protocol

Now we describe at a high level the communication protocol used by peers in DDP2P. For each message exchanged we provide a logic description the parameters passed and of their semantic. Each message also receives as parameter the address $\mathcal{R}$ of the remote peer, and the transport layer protocol **prot** used for communication (e.g., TCP or UDP).

Each software agent maintains a list of known peers and a list of directory servers which help it advertise its current address when this changes. A software agent of the peer consists of:

- a database replica (storing its data of interest),
- a human interface (e.g., GUI, for visualization, control and configuration),
- a client (who takes initiative to contact other peers),
- a TCP data server (for synchronization by streaming),
- a UDP data server (for communication behind NATs),
- a TCP directory server (for queries about addresses of peers), and
- a UDP directory server (for helping to pierce NATs using a version of STUN)

Each of these components of the agents are optional, except for the database. The database schema, not detailed in this report, can be normalized according to desired trade-offs between the costs to store the data with reduced replication, and to access fast the desired information. Each `item` stored into the database is associated with a timestamp of arrival, returned by the function `arrival(item)`. Similarly, each item can be marked as `blocked` to tell that one does not want to store related items, `broadcastable` to tell whether it should be shared with others, or `interest` to tell that related items should be requested from others.

The human interface is optional and certain modes of operation do not employ it. For example, a human user can start a directory service as a background process (e.g., UNIX daemon) in which case the only active components are the database and the two (TCP and UDP) directory servers.

Algorithm 1: Directory Server Procedures

```
   on registration (prot,R,⟨P,I,A,σ⟩) do
1.1    ⟨P,I',A',σ',date'⟩ ← db_retrieve (P);
1.2    if (prot=UDP) then
1.3        A← ⟨UDP : R⟩ ∪ A; // for NAT piercing;
1.4    db_store (⟨P,I,A∪trim(A',date'),σ,getDate()⟩);
1.5    send(confirm_registration, prot, R, (R));
   on address(prot,R,P_t,P_s,t,σ) do
1.6    ⟨P_t,I_t,A_t,σ_t,date_t⟩ ← db_retrieve (P_t);
1.7    NAT_adr ← extract_NAT_address(A_t);
1.8    if (prot=UDP) ∧ (prot(NAT_adr)=UDP) then
           if (sample drawn for full test) then
1.9            send(pierce_NAT, UDP, NAT_adr,
                   from_dir,⟨P_t,P_s,R,t,σ⟩);
           else
1.10           send(pierce_NAT, UDP, NAT_adr,
                   ⟨⊥,P_s,R,⊥,⊥⟩);
1.11   send(answer_address,prot,R,
           ⟨P_t,I_t,A_t,σ_t,date_t⟩);
   on alive(prot,R) do
       if accounting open NATs then
1.12       put(open_NATs, R, getTime());
```

The directory servers start by binding to their well-known predefined port numbers. Both directory servers handle **registration**, **address**, and **alive** requests as per Algorithm 1. The **registration** request has as parameters a tuple: $\langle \mathcal{P}, \mathcal{I}, \mathcal{A}, \sigma \rangle$, as detailed in Algorithm 1. Here, $\mathcal{P}$ stands for the global identifier (GID) of a peer that is registering its addresses $\mathcal{A}$. The request can contain up to a certain number, $MAX\_PEERINFO$, of bytes of extra searchable information $\mathcal{I}$ about the peer and is signed with

$$\sigma = SIGN(SK(\mathcal{P}), \langle \mathcal{P}, \mathcal{I}, \mathcal{A} \rangle).$$

The **registration** procedure, receives as additional information the perceived address of the remote peer, $\mathcal{R}$, and the transport layer protocol **prot**. If the protocol is UDP, then $\mathcal{R}$ is added to the list of addresses advertised by the peer. The operator $\cup$, employed at Lines 1.3 and 1.4, combines two vectors of addresses by appending to the first operand the new elements found in the second operand, in the order of appearance in the second operand. At Line 1.3, the perceived address of the client is added to the front of the addresses, to enable potential future participation in NAT piercing requests. The function $trim()$ used on Line 1.4 reduces from the vector of addresses $\mathcal{A}'$ found in the local storage (Line 1.1) a number of addresses function of how old those addresses are (the $date'$ parameter). A confirmation is sent at Line 1.5 to the client, informing it of its address $\mathcal{R}$ as perceived by the directory server. Based on this address $\mathcal{R}$, the client is able to detect whether it is located behind a NAT.

When a peer queries another's peer address using an **address** message, the directory server answers with the corresponding procedure in Algorithm 1. The parameters passed with the **address** request are the GID of the requested peer, $\mathcal{P}_t$, the GID of the peer issuing the request, $\mathcal{P}_s$, the time t of the request, and the signature of the request $\sigma = SIGN(SK(\mathcal{P}_s), (\textbf{address}, \mathcal{P}_t, t))$. After retrieving the current stored addresses $\mathcal{A}_t$ (Line 1.6), the answer is sent to the requesting peer at Line 1.11. If the protocol used by the request is UDP, and if the stored address can potentially come from a NAT translation, being a UDP address (function $prot()$ in Line 1.8), then a message is sent to the target peer (Line 1.9) to let it know it should open its NAT translation table for a connection with $\mathcal{P}_s$ at address $\mathcal{R}$. While all these parameters can be used to identify certain attacks from requesting peers, for efficiency this is done only with a small probability (probabilistic check of security) and most often the actual sent parameters contain just $(\mathcal{P}_s, \mathcal{R})$ (Line 1.10).

Peers found behind NATs will send frequent UDP **alive** messages to keep their NAT translation tables available. As it can be observed in Algorithm 1, the directory servers need not react to this messages. In practice, the directory server can keep a hashtable $open\_NATs$ storing for each $\mathcal{R}$ address the time of the last contact time (Line 1.12), to enable an evaluation on whether the NATs to a given address are currently open.

The data servers render the software agent available to other peers (see Algorithm 2). Data servers start by attempting to bind to a predefined port (Line 2.1). The detected addresses of the local host and the received port for the UDP and TCP listening sockets are stored in $server\_addresses$, and are sent to the set of directory servers $listing\_directories$ supporting this user (Line 2.2).

If the registration confirmation signals that this host is behind a NAT, the address perceived by the directory is stored in a list of NAT addresses NAT_state at

Algorithm 2: Data Server Procedures

Data: listing_directories: a set of directory server addresses that support me
Data: my_GID: GID of this peer
Data: info: declared information of this peer
on *startup* **do**
2.1 server_addresses ← bind server to a local port, preferably PORT;
foreach *directory* ∈ *listing_directories* **do**
2.2 **send**(**registration**,$\langle my\_GID, info, \mathcal{A}, \sigma\rangle$);

on confirm_registration(**prot**,$\mathcal{R}$,*perceived_address*) **do**
if (*perceived_address* ∉ *server_address*) **then**
2.3 NAT_state.add($\mathcal{R}$,perceived_address);
2.4 setIntervalTimer(NAT_DELAY, **send**(listing_directories, **alive**)));

on pierce_NAT(**prot**,$\mathcal{R}$,*src*,($\mathcal{P}_t, \mathcal{P}_s$,*peer_addr*,$t,\sigma$)) **do**
verify σ if available and quit on failure;
2.5 **if** *src* = `from_dir` **then**
2.6 **send**(**pierce_NAT**, UDP, peer_addr, from_target,($\mathcal{P}_t, \perp, \perp, \perp, \perp$));
2.7 **if** *src* = `from_src` **then**
2.8 **send**(**pierce_NAT**, UDP, peer_addr, from_target, ($\mathcal{P}_t, \perp, \perp, \perp, \perp$));
if *src* = `from_target` **then**
sendDataRequest();

on data_request(**prot**,$\mathcal{R}$,$\mathcal{P}_s$,$\mathcal{P}_t$,*items_src, interests, filter,horizon,flags*,σ) **do**
2.9 validate peer $\mathcal{P}_s$ and signature σ;
2.10 integrate(items_src);
2.11 req_items ← buildAnswerItems($\mathcal{P}_s$,interests);
2.12 time_horizon ← buildAnswerHorizon($\mathcal{P}_s$, filter, horizon);
2.13 available ← buildAnswerAvailable($\mathcal{P}_s$,time_horizon, filter);
2.14 **send**(**data_answer**,**prot**,$\mathcal{R}$,req_items,available, time_horizon, σ');

on data_answer(**prot**,$\mathcal{R}$, *requested_items, available, horizon,* σ) **do**
2.15 integrate(requested_items, horizon);
2.16 **foreach** *(GID,time)* ∈ *available* **do**
if *full data for GID not available since time'* **then**
schedule (GID,time') for the next request;

Line 2.3, and an interval timer is set to keep the NAT translation tables alive by re-sending an **alive** messages at an interval NAT_DELAY (Line 2.4).

When a peer gets a **pierce_NAT** message, it can learn via the *src* parameter that it may come from a client (`from_src`), from a directory server (`from_dir`), or from a target that was contacted (`from_target`). If the sender is a client (Line 2.7) then the NAT was already pierced and the client has to be announced about it with a reply **pierce_NAT** message sent at Line 2.8. In case the message comes from the directory servers (Line 2.5), the NAT with the requesting client may not yet be pierced and the message at Line 2.6 will open it. Either this message will reach the client (if the client has already sent its request), or the direct request from the client is sent later and then the message at Line 2.8 reaches it. If the message comes from the target, it signifies that the NAT translation tables are set for direct connection with the target, and the current peer can send its request.

When a **data_request** message is received, first the $\mathcal{P}_s$ is verified (Line 2.9). If it is blocked then the message is discarded. If this peer is unknown, based on configuration settings, the user may be questioned about whether it should be blocked and/or served. If the peer is not blocked then any data item it delivers on request are integrated in Line 2.10 (in case they are not blocked based on other criteria, such as of pertaining to blocked organizations, neighborhoods, constituents or motions). Peers can specify particular items by listing their GIDs in the parameter *interests* of **data_request**. If these items are found in the local storage, at Line 2.11 a subset of them are prepared for sending back to the requester (subset selected such as not to create an answer message that is too large).

Time Horizon The messages exchanged are limited in size to an arbitrary bound in order to enable the use of reasonable buffer sizes and to increase simplicity when communicating over UDP. Nevertheless, message sizes are not bounded by the size of the UDP datagrams, since the implemented communication engine is able to break a message into several datagrams and reassemble the message at the destination (with a simple mechanism of flow control and reliable delivery that we do not detail here). Each **data_request** message provides in the parameter *horizon* a value specifying that it, $\mathcal{P}_s$, thinks it already knows all the data items that have arrived at this peer, $\mathcal{P}_t$, at times `arrival(item)`$\geq$*horizon*. In case there is still space left in the answer message, in terms of a number of items that can be sent out given the aforementioned bound on message size, then a time horizon is computed at Line 2.12, based on the available data in the local database. This *time horizon*, *time_horizon*, is computed such that every item with arrival timestamp t preceding it, $t \leq time\_horizon$, and not yet sent to this destination, $t > horizon$, would still find space in the answer message. The obtained *time_horizon* is sent with the **data_answer** message such that the remote peer may communicate it back in future requests.

The GIDs of all items with arrival time between *horizon* and *time_horizon*, and of interest to this peer

(based on her declared set of interests, *filter*, in terms of GIDs of organizations and motions marked with the `interest` flag in its database), are extracted from the database (Line 2.13) and sent to the data requesting peer 2.14. In the *available* data structure sent in the **data_answer** message, all such GIDs are first grouped by the organization to which they pertain, and then by the type of item they represent (peer, organization, motion, etc.). In case the remote peer has lost some of the items sent to it in previous communication or changed its interests in organizations and motions, it can retrieve them back by specifying an appropriate value for *horizon* or by listing their GIDs in the parameter *interests*. The way we structure the *filter* parameter allows for specifying such horizons separately for different organizations. The full data for the items whose GIDs are included in *available* (Line 2.13) and are advertised in this way can be requested using the *interests* parameter of future **data_request** messages from this peer, in case the peer does not yet have them.

When sending available GIDs for items that can change in time, such as votes, constituents, peers or organizations, the peer also packs their creation date, such that remote peers can check whether they have the most recent version.

The peers can be configured to immediately pack and send the full data items rather than just their GIDs. This is more efficient for synchronizing with newly starting peers, but more inefficient with seasoned peers that already have most data from other sources. Peers can signal their desire to receive the full data immediately by setting the corresponding flag in the **data_request** message.

When a **data_answer** message is received by the data server (event which can happen only for the UDP version of the data server), the peer saves items in *requested_item* that are not related to blocked organizations, constituents, neighborhoods or motions (Line 2.15). If any of the data $(GID, time)$ advertised by this peer in *available* is unknown since certain time $time'$, then the corresponding GID are stored with the associated organization structure (Line 2.16) and will be requested from any peer serving that organization. If no instance of the item identified by GID is known, $time'$ is set to $\perp$ (empty). Queried peers will provide versions of the item for GID that have creation data newer than $time'$ (any, when $time' = \perp$).

Note that the requested creation time for the requested version of the item is not set to *time* but to $time'$. Setting it to *time* could have enabled a *freeze attack* where a valid instance with creation timestamp $time' < t < time$ cannot be requested since the peer is waiting for a newer inexistent version with creation timestamp *time*.

Data Client The client (whose procedures are in Algorithm 3) starts by setting up a TCP socket (Line 3.1). For UDP communications, the client uses the socket of its UDP server. The client proactively connects in an infinite loop (Line 3.2) to all the peers configured by its user for polling. For each such peer (Line 3.3), the client attempts to connect using all addresses it knows (Line 3.4). Addresses have types: e.g., directory.

If an address of the polled peer corresponds to a directory server, then an **address** message is sent to that server. All the addresses returned by the directory server are inserted into the head of the list of addresses to be tried next (Line 3.5). If needed, the directory server also contacts the remote peer using a **pierce_NAT** message, initiating eventual NAT piercing procedures. For any UDP address, a **data_request** message can be sent directly. If the UDP address can be behind a NAT, or to avoid useless work preparing a request when the remote peer is off-line, the communication can be started by a **pierce_NAT** message sent by this peer at Line 3.7. The reply from the remote peer will trigger the sending of the **data_request** message by the UDP data server, as seen before. TCP addresses are attempted with TCP connections and, if successful (Line 3.6), a **data_request** message is immediately

Algorithm 3: Data Client Procedures

```
   on startup do
3.1    set up TCP client socket;
3.2    forever // each loop defines a round
           wait timeout or data to send;
3.3        foreach peer ∈ peers_to_poll do
               contacted[peer] ← FALSE;
3.4            foreach a ∈ addresses(peer) do
                   if (!contacted[peer]) then
                       if (type(a) = DIR) then
                           addr ← send(TCP, a,
                                  (peer, my_peer_GID,
                                   getTime(), σ));
                           if (addr has NAT entry) then
                               send(UDP, a,
                                   (peer, my_peer_GID,
                                    getTime(), σ));
3.5                        put addr in addresses(peer);
                           continue;
                       if (prot(a)=TCP) then
3.6                        if connect(a) then break;
                       else
3.7                        send(pierce_NAT, UDP,
                                  a, peer, myself, σ);
3.8                        send(pierce_NAT, UDP,
                                  directories(peer), peer,
                                  myself, σ);
                   if (peer contacted via TCP) then
3.9                    sendDataRequest();
3.10                   integrate answer;
```

sent over the obtained channel (Line 3.9). Eventual results from the peer, if not blocked, are integrated at Line 3.10.

Verification of Constituent Identity When a user wants to verify that the public key used as GID by an active constituent item really corresponds to the real human with the name, address and email specified in its identity details, the user can meet that human and they can compare the public keys visually (as with PGP fests). Alternatively, the DDP2P software agents can generate a file containing a formal query $\langle C, n, s, a, d, r\rangle$ to be pasted/sent via a trusted authenticated channel (e.g., built on email/SMS) to the known constituent. Here C stands for the GID of the constituent being verified, n is its declared name and information, s is the name of the peer performing the verification, a is the set with its addresses, d is a timestamp and r is a random challenge. The user being verified can pass this file to its agent which can generate an answer (σ, r'), where $\sigma = SIGN(SK(C), \langle C, n, r, r', d\rangle)$ and r' is a random number. The answer is sent to a in a **verification_answer** message by the agent of the system being verified, and the agent of the verifier can automatically verify the answer, comparing r, and generate the witness stance item for C. An attacker cannot generate this answer since it cannot learn r.

Alternatively, the answer can also be copied and pasted into an email by the user being verified, from where the verifier can load it into its own agent for accomplishing the verification and witnessing.

Attacks Note that, for security reasons discussed below, we currently do not allow for the public key of the authority in an authoritarian organization to be automatically replaced in case it is revoked. Similarly, the constitution of a grassroot organization cannot change, without migrating to a completely new organization.

Attack on global identifiers The above mentioned security considerations refer to the attacks based on generating new organizations with the same global identifier as the attacked organization but with different parameters. Since the organization parameters define the ontology of the activity and the employed criteria for eligibility of constituents, such an attack can mislead the constituents of the attacked organization to generate data that is inconsistent with the constitution and rules of the organization. For example, a constituent A of an attacked grassroot organization O_1 could be misled by the constitution of the organization O_2 of an attacker to support the eligibility of another participant B (eligible with O_2 but not with O_1), support that would be transferable as valid into the attacked organization, O_1.

Other Attacks on Authoritarian Organizations The *Freeze Attack* is an attack possible only on authoritarian organizations where the secret key of the authority is destroyed. The *State Coup Attack* is also an attack possible only on an authoritarian organizations, where an attacker steals the secret key of the authority while destroying all the copies available to the original authority. The *Confuse Attack* on an authoritarian organization is where the attacker gets a copy of the secret key of the authority and issues contradictory parameters for the organization.

In the case of the Confuse Attack, the original authority can issue a `revoke` message (a new definition of the organization with the r parameter set) and the organization is blocked, hinting the constituents that they have to move their activity to a new organization. For Freeze Attacks and State Coup Attacks, the constituents cannot be warned automatically.

Attack on Grassroot Organizations An *Identification Attack* is where an attacker creates an organization with similar but not identical parameters, to the attacked organization. Since not all parameters are easily visible in graphical widgets, attacker organizations where the most visible part (in some GUI) is similar to the view of the attacked organizations can create confusion among users. This attack also works against authoritarian organization. The *Creator Attack* on a grassroot organization is an attack whereby the secret key of the creating peer of the organization is compromised. Such an attack can facilitate an *Identification Attack* by setting the name of the creator to values that are not recognizable to users.

To detect storage attacks, any received item is tagged with the GID of the peer from which it is received.

Theoretical Analysis of DDP2P

Not all the desirable properties (scalability, non-repudiation, etc.) discussed in the previous sections can be easily proven for the described protocol. Now we address independently each of them. Some of these properties hold by construction while others are not yet supported.

Property 1 (Opportunity) *The DDP2P protocol provides* equal opportunity of initiative, equal opportunity of management *and* equal opportunity of voting, *as defined here.*

The above properties follow immediately by construction, since DDP2P provides no mechanism that could stop any user from creating an organization, witness for other constituents, or submit or vote a motion. Even in the case where a newly submitted motion is identical to an existing one (identical GIDs), the second instance of the motion will be automatically assimilated and merged to its original instance.

Limits to opportunity for visibility What is impossible is for a user to act (initiate an organization, witness or submit a voting item) using somebody else's

secret key. There is no privileged administrator or user and all items have equal opportunity to get visibility. There can be other differences in opportunity stemming from the date when the item is submitted and the network of peers that disseminate and provide the supporting votes to make a new motion or justification visible.

Property 2 (Sender) *The DDP2P protocol provides* non-repudiation of sender *for senders whose identity is verified and not revoked.*

Limits of non-repudiation of sender This property is not trivially ensured by the usage of digital signatures since users can only trust those senders whose key was witnessed by other trusted constituents or whose identity was directly verified (e.g., confirming their GID using the protocol in the previous section).

Attackers can create false identities under which they can distribute digitally signed items. Items from untrusted constituents (not yet verified) do not offer non-repudiation of sender. Peers may block transmission of items from constituents that are not trusted either directly or via transitivity from trusted constituents.

Property 3 (Attribution) *DDP2P provides* repudiation of false attribution *when secret keys are not compromised, when either constituents are trusted only based on direct verification or any trusted constituents witness correctly about verified peers.*

Proof For an attacker A to attribute an item to another user B it needs to steal the secret key of B. In case A creates a false user B' with the same name and parameters as B, then A needs to convince other users to trust B', which contradicts the assumptions. □

In conclusion, a user can prove her innocence and any careful user can avoid believing false attributions.

Property 4 (Coherence) *The items disseminated using the DDP2P protocol offer* coherence.

Proof Items with GIDs and authentication based solely on secure digest values (e.g., grassroot organization) are immutable, each of them is stored independently, and the GIDs based on secure digest avoid conflicts. The same holds for items with GIDs based on public keys. Mutually exclusive items (e.g., vote by the same constituent for the same motion and witness from the same constituent for the same target) that are digitally signed and have different creation dates can be robustly and coherently compared by all participants (storing only the one with the newest date).

The fact that for exclusive items that are different but have identical date we store the ones with lexicographically the largest digest value, guarantees that all correct peers coherently store the same alternative of the two conflicting items. □

This coherence property guarantees that any connected peers will have identical items at quiescence if there is a path between them where those items are correctly accepted as of interest and broadcastable by all peers on the path (and under the assumption that synchronization between immediately connected peers is complete).

Self-interest Since the reference implementation is open source, programmers can easily change its behavior to fit their interests. Users are also provided with ample configuration options to specify their filters on received and sent data. The stronger these filters, the less paths will be available between users interested in the same items (needed for synchronization).

Given the mechanism used by DDP2P for synchronizing systematically any two immediately connected peers, we can state the provided version of the *soundness* property as:

Property 5 (Soundness) *Each DDP2P constituent eventually gets any item of information interesting to her, whenever there exists a chain of immediately connected peers interested in storing and broadcasting that type of information, and linking her to the constituent generating that item.*

Grassrootedness By construction, grassroot organizations enable the collaboration of people that are not known or supported by any central authority. In contrast authoritarian organizations only enable the coordination of citizens witnessed by its authority.

Property 6 (Verifiability) *The DDP2P protocol enables the verification of the identity of known constituents with reasonable effort.*

Proof Constituents with which this user is in contact can be queried by external means (e.g., using direct physical contact or electronic contact via externally-learned email or SMS addresses) whether they know the secret key for the GID of a constituent item with their name, address and email/SMS (see *Verification of Constituent Identity* in section *Protocol*). □

Relevance Collaborative filtering is enabled by the structure of the items, since each of them is voted (participants, neighborhoods, motions, justifications). Organizations are scored by the number of registered constituents, constituents by the witness stances, neighborhoods by witness stances and inhabitants, motions and justifications by the votes. Further collaborative filtering is ensured by the fact that users can separately block receipt and broadcasting of items. While these measures match state of the art, additional techniques may be needed in the future to counter future attacks.

Property 7 (Openness) *DDP2P provides the openness property to a social network (users do not lose*

personal data when somebody else's system fails, and new users can dynamically join at any moment).

Proof Each agent has its own copy of the whole data, therefore none of the items it generates or store can be lost due to the failure of any other peer. Any new agent starting the system starts synchronizing from time 0 and can therefore get all the items offered from the databases of its peers. □

Property 8 (Resilience from Censorship) *The DDP2P protocol provides resilience from censorship.*

Proof By construction, whenever a user directly broadcasts an item to a peer, that item can be registered by the destination and stored. There is no mechanism provided by DDP2P by which another constituent could remove that item. □

Resilience from Attacks While it is in general hard and rare to prove that any given technique would resist to unknown future attacks (and we do not prove it here), the proposed peer-to-peer architecture is particularly resilient in the sense that all agents have copies of all data of interest to them and an attack, to be successful, has to disrupt most of these peers simultaneously. For resilience to sybil attacks, see (Qin *et al.* 2013).

Conclusions

A framework, a peer-to-peer (P2P) communication protocol, and a system implementing them are proposed for a fully decentralized coordination. The properties of this system are studied for an argumentation application. With this framework, a constituency can be built in a complete bottom-up fashion. The eligibility is defined among participants by voting on each other according to principles that can be defined by anybody. Anybody can submit formal proposals (motions) to be debated and voted by the constituency. Any set of defined principles constitute an organization and any peer agreeing with these principles can join the corresponding organization. Based on transitivity of trust, each constituent has her own view of the status of the constituency and of the argumentation around motions.

The proposed P2P protocol enables synchronization of data provable to converge to a coherent state at quiescence given minimum connectivity. A key element is provided by a mechanism of referring to items using global identifiers (GIDs) defined in a way that avoids voluntary and involuntary collisions (by generating the GIDs as public keys or digest values). The protocol is based on a fully decentralized communication, where each peer can learn about other peers using existing connections or external addresses. Similarly, each peer can simultaneously be a supernode (helper for communication behind NATs and mobility), and can have a list of its own personal supernodes to enable roaming. Communication uses a combined pull-push mechanism where locally known items are segmented into groups based on their arrival time and on the capacity configured for individual messages given the current communication mechanism. Transfers only communicate GIDs for items as long as the peer does not identify these items as new for her and needed.

A reference implementation is provided as a P2P system for fully decentralized voting, where the above concepts and protocols were verified and validated. The studied properties of the new framework are *coherence, self-interest, soundness, verifiability, scalability, relevance, openness, resilience and repudiation of false attribution.* It also provides *equality of opportunity for initiative, management and voting.*

References

Buchegger, S.; Schiöberg, D.; Vu, L. H.; and Datta, A. 2009. PeerSoN: P2P social networking. In *WSNS*, 46–52.

Cohen, B. 2003. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, 68–72.

Douceur, J. 2002. The sybil attack. *Peer-to-peer Systems* 251–260.

Federation. Federated social web. `http://www.w3.org/2005/Incubator/federatedsocialweb/wiki/Main_Page`.

Fedoruk, A. ; Denzinger, J. 2006. A general framework for multi-agent search with individual and global goals: Stakeholder search. *ITSSA* 1(4):357–362.

Ford, B., and et al. 2005. Peer-to-peer communication across network address translators. In *USENIX Annual Technical Conference*, 179–192.

Gu, B.; Jarvenpaa, S.; et al. 2003. Are contributions to p2p technical forums private or public goods? In *Work. on Economics of P2P Systems.*

K. Raaflaub, J. O., and Wallace, R. 2007. *Origins of Democracy in Ancient Greece.* Univ. California Press.

Kagawa, T. 1936. *Brotherhood Economics.* Harper & Brothers.

Nieminen, K. 2003. Legal issues in p2p systems. *Peer to Peer and SPAM in the Internet* 115.

Qin, S.; Silaghi, M.; Matsui, T.; Yokoo, M.; and Hirayama, K. 2013. P2p decentralized population census. In *Workshop on Decentralized Coordination.*

Roberts, H. M. 1989. *Rules of Order.* Berkeley.

W3C. Ostatus community group. `www.w3.org/community/ostatus/`.

Xie, H., and Yang, Y. 2007. A measurement-based study of the skype p2p voip performance. In *IPTPS.*

Yahalom, R.; Klein, B.; and Beth, T. 1993. Trust relationships in secure systems-a distributed authentication perspective. In *Research in Security and Privacy*, 150–164.