Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 389 (2007) 528–549
Theoretical Computer Science
www.elsevier.com/locate/tcs

The complexity of fixed point models of trust in distributed networks

Karl Krukow[a], Andrew Twigg[b,*]

[a] *BRICS, University of Aarhus, Denmark*
[b] *Computer Laboratory, University of Cambridge, Cambridge, United Kingdom*

Abstract

In this paper we consider the complexity of some problems arising in a fixed point model of trust in large-scale distributed systems, based on the notion of trust structures introduced by Carbone, Nielsen and Sassone; a set of trust levels with two distinct partial orderings. In the trust model, a global trust state exists as the least fixed point of a collection of local policy functions of nodes in the network.

We first show that it is possible to efficiently compute a single component of the global trust state using a simple, robust and totally asynchronous distributed algorithm. We complement this with a lower bound which shows that, if the policies are unrestricted then communication and time linear in the number of distinct trust levels is required in the worst case.

We then consider the notion of distributed proof carrying requests previously introduced as a means of safely approximating the global trust state without computing its exact value. We present a new result that enables us to give a continuum of proof carrying protocols, the previously known protocol being one extreme of this. The theorem allows us to generate protocols that can prove all possible trust values (previously, it was only possible to prove trust values representing 'not too much bad behaviour'). However, we show that such a general protocol may not be efficient — it is NP-hard to construct an approximately minimal size proof in our model, and in the worst case the nodes must communicate almost as much data as if they were to compute the global trust state from scratch. The implications of our negative results are that it may be necessary to restrict the policy language in order to efficiently implement a fixed point model of trust in a distributed network.

Keywords: Distributed networks; Trust models; Fixed point computation; Proof carrying requests

1. Introduction

The vision of global computing includes large-scale distributed, ubiquitous and autonomous systems. How best to design a flexible security mechanism for a global computing system is not clear. The diversity and scale combined with the lack of any centralized authority means that traditional security mechanisms such as access control lists, are often too restrictive and complex to deploy [1]. The concept of trust management, introduced by Blaze et al. [2], was presented as a solution to the problems with authorization in large-scale distributed systems. Traditional trust management systems make security decisions based on policies, dealing with authorization by deciding the so-called compliance checking problem: given a *request* to perform a certain action, together with a set of *credentials*, does the request comply with the local *security policy*?

* Corresponding author.
E-mail address: andrew.twigg@cl.cam.ac.uk (A. Twigg).

0304-3975/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2007.09.007

While the traditional notion of trust management is well-understood, e.g. Mitchell et al. [3–5], and, to a large extent, captured concisely in a mathematical framework of Weeks [6]; a lot of the "broader" dynamic systems lack such foundation in formal methods (this point is illustrated by the wide range of related systems in the survey [7]). This lack prompted the development of a mathematical framework for trust [8], inspired by that of Weeks, but departing from Weeks by emphasizing the concept of *information* in contrast to *authorization*. The framework, which was introduced by Carbone et al. [8], discussed also by Nielsen et al. [9] and Krukow [10], is the focus of this paper. In this paper we present techniques for implementing this model in a large-scale distributed network, and consider some more fundamental problems of complexity associated with fixed point models of trust.

1.1. The trust structure framework

A trust model should be generic enough to be instantiated to support authorization in a variety of distributed computing systems. The trust structure framework [8] is a generic model, parameterized by a set X of possible *trust values* representing distinct levels or degrees of trust, relevant for a particular application. The framework is aimed at global computing environments, and is based on a domain-theoretic modeling of trust information. The goal is to provide a *unique global trust state* for every set $\mathcal{P}$ of principal identities, each principal $p \in \mathcal{P}$ defining a trust policy π_p which quantifies for any principal identity $q \in \mathcal{P}$ the level of trust that p has in q.

Trust structures. In the framework, trust is something which exists between *pairs of principals*; it is *quantified* and *asymmetric* in that we care of "how *much*" or "to what *degree*" principal p trusts principal q (which may not be to the same degree that q trusts p). Each application instance of the framework defines a so-called *trust structure*, $T = (X, \preceq, \sqsubseteq)$, which consists of a set X of *trust values*, together with two relations on X; the trust ordering ($\preceq$) and the information ordering ($\sqsubseteq$). The elements $s, t \in X$ express the levels of trust that are relevant for the particular instance, and $s \preceq t$ means that t denotes at least as high a trust level as s. In contrast, the information ordering introduces a notion of precision or information. The key idea is that the elements of X embody various degrees of uncertainty. One may think of assertion $x \sqsubseteq y$ as the statement that x can be refined into y, or that x approximates y.

Definition 1 (*Trust Structure*). A *trust structure* is a triple $T = (X, \preceq, \sqsubseteq)$, consisting of a set X of *trust values*, ordered by two binary relations: $\preceq \subseteq X \times X$ called the *trust ordering* of T, and $\sqsubseteq \subseteq X \times X$ called the *information ordering* of T. The trust ordering is a preorder on X, meaning that it is reflexive and transitive, and the information ordering makes $(X, \sqsubseteq)$ an ω-complete partial order with a least element, denoted $\bot_{\sqsubseteq}$. For any $\sqsubseteq$-ω-chain, $x_0 \sqsubseteq x_1 \sqsubseteq \cdots$, the least upper bound in $(X, \sqsubseteq)$ is denoted by

$$\bigsqcup_{i \in \omega} x_i.$$

In simple cases, the trust values are just symbolic, e.g. $\mathsf{unknown} \sqsubseteq \mathsf{low} \preceq \mathsf{high}$, but they may also have more internal structures. As a simple example of a trust structure, consider the so-called "MN" trust structure T_{MN} [10]. In this structure, trust values are pairs (m, n) of (extended) natural numbers, representing $m + n$ interactions with a principal; each interaction classified as either "good" or "bad". In a trust value (m, n), the first component, m, denotes the number of "good" interactions, and the second, the number of "bad" ones. The information ordering is given by: $(m, n) \sqsubseteq (m', n')$ only if one can refine (m, n) into (m', n') by adding zero or more good interactions, and, zero or more bad interactions, i.e., iff $m \leq m'$ and $n \leq n'$. In contrast, the trust ordering is given by: $(m, n) \preceq (m', n')$ only if $m \leq m'$ and $n \geq n'$. Nielsen et al. [11,8], have considered several additional examples of trust structures.

Trust policies. Given a fixed trust structure $T = (X, \preceq, \sqsubseteq)$, a *global trust state* of the system is a function $\mathsf{gts} : \mathcal{P} \to \mathcal{P} \to X$. The interpretation is that gts represents the trust state where p's trust in q (formalized as an element of X) is given by $\mathsf{gts}(p)(q)$. The goal of the framework is to uniquely define a global trust state, denoted $\overline{\mathsf{gts}}$. Each principal $p \in \mathcal{P}$ autonomously controls a trust policy, denoted π_p, which then determines p's trust within the unique global trust state, i.e. $\overline{\mathsf{gts}}(p)$.

Definition 2 (*Trust Policy*). Let $T = (X, \preceq, \sqsubseteq)$ be a trust structure. A *trust policy* in T, is a function $\pi : (\mathcal{P} \to \mathcal{P} \to X) \to \mathcal{P} \to X$, which is continuous with respect to the pointwise extension of $\sqsubseteq$.[1] This continuity property is called *information continuity*.

[1] We overload $\sqsubseteq$ (respectively $\preceq$) to denote the pointwise extension of $\sqsubseteq$ ($\preceq$) to the function space $X^{\mathcal{P}} = \mathcal{P} \to X$ as well as to $(X^{\mathcal{P}})^{\mathcal{P}} = \mathcal{P} \to \mathcal{P} \to X$.

In the simplest case, π_p could be a constant function, ignoring its first argument $\mathsf{gts} : \mathcal{P} \to \mathcal{P} \to X$. As an example, $\pi_p(\mathsf{gts}) = \lambda q.t_0$ (for some $t_0 \in X$) defines p's trust in any $q \in \mathcal{P}$ as the constant t_0. In general, policy π_p may refer to other policies (π_z, $z \in \mathcal{P}$), and the general interpretation of π_p is the following. *Given that* all principals assign trust values as specified in the global trust state $\mathsf{gts} : \mathcal{P} \to \mathcal{P} \to X$, *then* p assigns trust values as specified in function $\pi_p(\mathsf{gts}) : \mathcal{P} \to X$. For example, function $\pi_p(\mathsf{gts}) = \lambda q \in \mathcal{P}.(\mathsf{gts}(A)(q) \vee_{\preceq} \mathsf{gts}(B)(q)) \wedge_{\preceq} \mathsf{medium}$, represents a policy saying "for any $q \in \mathcal{P}$, the trust in q is the least upper bound in $(X, \preceq)$ of what A and B say, but no more than $\mathsf{medium} \in X$." Such *policy references* are very similar to the concept of *delegation*, known from traditional trust management.

Unique trust state. The collection of the trust policies of all principals, denoted $\Pi = (\pi_p : p \in \mathcal{P})$, thus "spins a global web of trust" in which the trust policies mutually refer to each other. Since trust policies Π may give rise to cyclic policy references, a crucial requirement is that the information ordering makes $(X, \sqsubseteq)$ a complete partial order (cpo) with a bottom element. Since all policies are information-continuous, there exists a unique information-continuous function $\Pi_\lambda = \langle \pi_p : p \in \mathcal{P} \rangle$, of type $X^{\mathcal{P}^{\mathcal{P}}} \to X^{\mathcal{P}^{\mathcal{P}}}$ with the property that $\mathtt{Proj}_p \circ \Pi_\lambda = \pi_p$ for all $p \in \mathcal{P}$, where $\mathtt{Proj}_p$ is the pth projection.[2] Since Π_λ is information-continuous and $(\mathcal{P} \to \mathcal{P} \to X, \sqsubseteq)$ is a cpo with a bottom element, Π_λ has a (unique) least fixed point [12] which we denote $\mathsf{lfp}_{\sqsubseteq} \Pi_\lambda$ (or simply $\mathsf{lfp}\, \Pi_\lambda$):

$$\mathsf{lfp}_{\sqsubseteq} \Pi_\lambda = \bigsqcup_{\sqsubseteq} \{\Pi_\lambda^i(\lambda p.\lambda q.\bot_{\sqsubseteq}) \mid i \in \mathbb{N}\}$$

(where $\bigsqcup_{\sqsubseteq}$ means the least upper bound wrt the ordering $\sqsubseteq$). Hence, for any collection of trust policies Π, we can define the unique global trust state induced by that collection, as $\overline{\mathsf{gts}} = \mathsf{lfp}\, \Pi_\lambda$, which has the type of global trust states, $\mathcal{P} \to \mathcal{P} \to X$. This unique trust state thus satisfies the following fixed point equation:

$$\begin{aligned} \forall p \in P.\quad m(p) &= \mathtt{Proj}_p(m) \\ &= \mathtt{Proj}_p(\Pi_\lambda(m)) \qquad (\text{since } \Pi_\lambda(m) = m) \\ &= \pi_p(m). \end{aligned}$$

Reading this from the left to the right, any function $m : \mathcal{P} \to \mathcal{P} \to X$ satisfying this equation is *consistent* with the policies $(\pi_p \mid p \in \mathcal{P})$, i.e. *any* fixed point of Π_λ is consistent with all policies π_p. Consider now two mutually referring functions π_p and π_q, given by $\pi_p = \lambda m.\mathtt{Proj}_q(m)$, and $\pi_q = \lambda m.\mathtt{Proj}_p(m)$. Intuitively, there is no information present in these functions; p delegates all trust questions to q and similarly q delegates to p. In this case, we would like the global trust state gts induced by the functions to take the value $\bot_{\sqsubseteq}$ on any entry $z \in \mathcal{P}$ for both p and q, i.e., for both $x = p$ and $x = q$ and for all $z \in \mathcal{P}$ we should have $\mathsf{gts}(x)(z) = \bot_{\sqsubseteq}$. This is exactly what is obtained by choosing the information-*least* fixed point of Π_λ. We summarize this as a definition.

Definition 3 (*Global Trust State*). Let $T = (X, \preceq, \sqsubseteq)$ be a trust structure, $\mathcal{P}$ a set of principal identities, and $\Pi = (\pi_p \mid p \in \mathcal{P})$ be a collection of trust policies in T, indexed by the principal identities. Let $\Pi_\lambda = \langle \pi_p \mid p \in \mathcal{P} \rangle$. The *global trust state induced by* Π, denoted $\overline{\mathsf{gts}}$, is given by

$$\overline{\mathsf{gts}} = \mathsf{lfp}_{\sqsubseteq} \Pi_\lambda.$$

1.2. The operational problem

Many interesting systems are instances of the trust structure framework [8,10], but one could argue against its usefulness as a basis for the actual construction of trust management systems. In order to make security decisions, each principal p will need to reason about its trust in others, that is, the values of $\overline{\mathsf{gts}}(p)$. While the framework does ensure the existence of a unique (mathematically well-founded) global trust state, it is not "operational" in the sense of providing a way for principals to actually *compute* the trust values. Furthermore, as we shall argue in the following, the standard way of computing least fixed points is inadequate in our scenario.

When the cpo $(X, \sqsubseteq)$ is of finite height h, the cpo $(\mathcal{P} \to \mathcal{P} \to X, \sqsubseteq)$ has height $|\mathcal{P}|^2 \cdot h$.[3] In this case, the least fixed point of Π_λ can, *in principle*, be computed by finding the first identity in the chain of approximants

[2] $\mathtt{Proj}_p$ is given by: for all $m : \mathcal{P} \to \mathcal{P} \to X$. $\mathtt{Proj}_p(m) = m(p)$.

[3] The height of a cpo is the size of its longest chain.

$(\lambda p.\lambda q.\bot_{\sqsubseteq}) \sqsubseteq \Pi_\lambda(\lambda p.\lambda q.\bot_{\sqsubseteq}) \sqsubseteq \Pi_\lambda^2(\lambda p.\lambda q.\bot_{\sqsubseteq}) \sqsubseteq \cdots \sqsubseteq \Pi_\lambda^{|\mathcal{P}|^2 \cdot h}(\lambda p.\lambda q.\bot_{\sqsubseteq})$ [12]. However, in the environment envisioned, such a computation is infeasible. The functions $(\pi_p : p \in \mathcal{P})$ defining Π_λ are distributed throughout the network, and, more importantly, even if the height h is finite, the number of principals $|\mathcal{P}|$, though finite, will be *very* large. Furthermore, even if resources were available to make this computation, we cannot assume that any central authority is present to perform it. Finally, since each principal p defines its trust policy π_p autonomously, an inherent problem with trying to compute the fixed point is the fact that p might decide to change its policy π_p to π'_p at any time. Such a policy update would be likely to invalidate data obtained from a fixed point computation done with global function Π_λ, i.e., one might not have time to compute $\mathsf{lfp}\,\Pi_\lambda$ before the policies have changed to Π'.

The above discussion indicates that exact computation of the fixed point is infeasible, and hence that the framework is not suitable as an operational model. Our motivation is to counter this by showing that the situation is not as hopeless as suggested. The rest of the paper presents a collection of techniques for *approximating* the idealized state $\mathsf{lfp}\,\Pi_\lambda$.

1.3. Contributions

We build on the formal fixed point model of trust presented in [8] by giving algorithmic techniques for efficient distributed computation of approximations to the idealized trust values. In addition we give several important negative results that suggest worthwhile future directions in this area.

The above details show that the notion of delegation, or policy reference, is crucial to fixed point models of trust, and in particular, the existence of delegation *cycles*. However, this introduces additional complexity in that the policies may *interact*. Since the policies are distributed throughout the network, it is important to understand the complexity of these interactions when one wishes to compute a portion of the global trust state. Until now, very little has been known about the complexity of these interactions, and what sorts of policies induce complex interactions.

We start by showing that although it may be infeasible to compute the global trust state, one can instead try to compute the so-called *local* fixed point values. We take the realistic view that there is some specific principal, wanting to reason about its trust value for some other principal. The basic idea is that instead of computing the entire state and *then* looking up the desired value, one may instead compute this value directly. We prove our results in an abstract model that removes most of the additional complexity associated with the trust model scenario. In the abstract model, there is a graph of n nodes where each has an associated function f_i, the collection of these functions gives a global function $F = (f_1, \ldots, f_n)$ and we wish to compute the ith component of the fixed point of F, denoted by f_i^*. This model naturally models computation in boolean networks and may be useful in many other contexts.

We prove a convergence result that enables us to apply a robust totally asynchronous distributed algorithm of Bertsekas [13] for local fixed point computation in this abstract model. This is developed in Section 2. We complement the above by proving a lower bound on the communication complexity of any distributed algorithm to compute a single component of the global trust state. Since our lower bound in proved in the abstract model, the technique used to establish it may be of independent interest.

We also develop the notion of 'proof carrying requests' by presenting a new result that gives proof carrying protocols that can be used for all trust values (previously, it was only known how to prove trust values representing 'not too much bad behaviour'). However, we show that, unfortunately, such a general proof carrying protocol may not be efficient — in some cases it is NP-hard to construct an approximately minimal size proof, and in the worst case the nodes must communicate at least as much data as if they were computing the global trust state from scratch.

The implication of our negative results is that it may be necessary to restrict the policy language in order to efficiently implement the trust model in a distributed network. We have been unable to obtain interesting results in this area but it remains an interesting area for future work.

1.4. Organization

The rest of the paper is organized as follows. In Section 2 we present a distributed algorithm and prove its correctness in a totally asynchronous network. In Section 3 we prove a lower bound on the complexity of computing least fixed points in networks when the policies are unrestricted. Section 4 presents the new proof carrying request techniques, and in Section 5 we use the lower bounds of Section 3 to show a negative result on the complexity of proof carrying request protocols.

2. Distributed computation of local least fixed points

In this section, we show how to compute the *local* fixed point value $\overline{\mathsf{gts}}(R)(q)$ for two fixed principals R and q, without computing the complete global trust state $\overline{\mathsf{gts}}$. The reason for computing local values is two-fold. First, we can benefit from distributing the computational and storage burdens, so that instead of centrally computing the complete state $\overline{\mathsf{gts}}$, node R will maintain "entry" $\overline{\mathsf{gts}}(R)(q)$ in the "distributed matrix" $\overline{\mathsf{gts}}$. Second, although the semantics of trust policies are functions of the type $(\mathcal{P} \to \mathcal{P} \to X) \to \mathcal{P} \to X$ which (due to policy referencing) in general may depend on the trust values of *all* principals, we expect that in practice, policies will not be written in this way. Instead, policies are likely to refer to a few known (and usually "trusted") principals. For fixed R and q, the set of principals that R's policy *actually* depends on in its entry for q, is often a significantly smaller subset of $\mathcal{P}$. For example, consider our policy from the previous section.

$$\pi_R(\mathsf{gts}) = \lambda q \in \mathcal{P}.(\mathsf{gts}(A)(q) \vee_{\preceq} \mathsf{gts}(B)(q)) \wedge_{\preceq} \mathsf{medium}.$$

This policy is independent of all entries of gts except for those of principals A and B. This means that in order to evaluate π_R with respect to some principal q, R needs only information from A and B.

We first compute (distributedly) a dependency graph which contains *only the dependencies relevant for the computation of* $\overline{\mathsf{gts}}(R)(q)$, thus excluding the set of principals that do not need to be involved in computation. We then proceed with the computation of $\overline{\mathsf{gts}}(R)(q)$ by showing that the conditions of a general algorithmic convergence theorem of Bertsekas [13] are satisfied, and hence we can appeal to known convergence results.

2.1. The abstract problem

We now describe the abstract model in which we shall prove our main results, and describe how the trust model problem is translated to this model. Although the results can be interpreted in the trust model setting, it is much easier to simply work in the abstract domain.

Let $(X, \sqsubseteq)$ be a cpo of finite height h and given some collection $F = (f_1, \ldots, f_n)$ of monotone functions $f_i : X^n \to X$, build a network G of processors where processor i has associated with it function f_i, and edges in the network represent the dependencies between the functions — the edge (i, j) is present iff function f_i depends on the variable x_j. It follows that the unique function $F : X^n \to X^n$ satisfying $F = \langle f_1, f_2, \ldots, f_n \rangle$ is also monotone and so has a unique least fixed point which we denote by $F^* = (f_1^*, \ldots, f_n^*) \in X^n$. Fix some special node $r \in \{1 \ldots n\}$, called the *root node*. Then the problem is for r to compute f_r^*.

We use an asynchronous communication model, assuming no known bound on the time it takes for a sent message to arrive. We assume that there exists some underlying protocol for sending messages between nodes with the following properties: communication is reliable in the sense that any message sent arrives exactly once to the destination node and that messages arrive in the order in which they are sent by a node (although messages from different nodes can be interleaved). More details on asynchronous models of communication can be found in [13].

We translate the trust structure setting into the abstract setting of the network model as follows. The n nodes of the network are labelled by pairs xy for $x, y \in \mathcal{P}$. The node xy will be responsible for computing approximations to the value $\overline{\mathsf{gts}}(x)(y)$, i.e., x's trust value for y. Since we are interested in a specific principal R's trust value for q, the *root node* is the node with label Rq. However, we only want to consider the nodes xy that are relevant for computing the value $\overline{\mathsf{gts}}(R)(q)$, and hence, not all labels appear in the network (i.e., $n \ll |\mathcal{P}|^2$).

The nodes that actually appear in the network (and their mutual dependencies) can be efficiently determined by a simple distributed spanning tree algorithm, assuming that each principal x knows the dependencies of its own policy π_x. For each node $i = xy$ in the network, the function f_i associated with that node is the entry for y in x's policy, i.e., the projection $\mathtt{Proj}_y \circ \pi_x$. The technical report [14] contains an asynchronous distributed algorithm to construct the dependency graph.

Note that, this translation might lead to a principal, say p, appearing several times in the dependency graph, e.g., as nodes py and pz for $y \neq z$. We shall think of these as distinct nodes in the graph, although a concrete implementation would have principal p playing the role of both nodes (e.g., both of the abstract operations of sending a message to py and sending a message to pz would be implemented as sending messages to principal p in the physical communication network). Note also, that the dependency graph is *not* modeling any network topology. Although the

nodes of the graph represent concrete nodes in a physical communication network, its edges do not represent any particular communication links.

From now on, we shall work in the abstract setting as it simplifies notation. Note that this translation might lead to a node z appearing several times in the dependency graph, e.g. with entries for principals w and y in π_z. We shall think of these as distinct nodes in the graph, although a concrete implementation would have node z playing the role of two nodes, z_w and z_y. Note also, that the (minimal) dependency graph is *not* modeling any network topology. Although the nodes of the graph represent concrete nodes in a physical communication network, its edges do not represent any communication links.

2.2. An asynchronous algorithm

Assume that the dependency graph has already been computed, and each node i knows two sets: $N^+(i)$, the set of variables (nodes) on which the function f_i depends, and $N^-(i)$, the set of nodes whose policy depends on i. We shall show that this gives a situation in which we can apply the existing work of Bertsekas for distributed computation of the least fixed point. Bertsekas has a class of algorithms, called totally asynchronous (TA) iterative fixed point algorithms, and a general theorem which gives conditions ensuring that a specific TA fixed point algorithm will converge to the desired result. In our case, "converge to" means that each principal $i \in \mathcal{P}$ will compute a sequence of values $\bot_{\sqsubseteq} = i.t_0 \sqsubseteq i.t_1 \sqsubseteq \cdots \sqsubseteq i.t_k = f_i^*$. The general theorem is called the "Asynchronous Convergence Theorem" (ACT), and we use this name to refer to Proposition 6.2.1 of [13]. The ACT applies in any scenario in which the Synchronous Convergence Condition (SCC) and the Box Condition (BC) are satisfied. Intuitively, the synchronous convergence condition states that if the algorithm is executed synchronously, then one obtains the desired result. In our case, this amounts to requiring that the "synchronous" sequence $\bot_{\sqsubseteq} \sqsubseteq F(\bot_{\sqsubseteq}) \sqsubseteq \cdots$ converges to the least fixed point F^*, which is true. Intuitively, the box condition requires that one can split the set of possible values appearing during synchronous computation into a product ("box") of sets of values that appear locally at each node in the asynchronous computation. As a consequence of $\sqsubseteq$-monotonicity of the policies, the conditions of the Asynchronous Convergence Theorem are satisfied (the following Proposition 7), and so, we can deploy a totally asynchronous distributed algorithm.

We now describe the algorithm and argue for its correctness. We will assume that each node i allocates variables $i.t_{cur}$ and $i.t_{old}$ of type X, which will later record the "current" value and the last computed value in X. Each node i has also an array, denoted by $i.m$. The array $i.m$ is of type X array, and will be indexed by the set $N^+(i)$. Initially, $i.t_{cur} = i.t_{old} = \bot_{\sqsubseteq}$, and the array is also initialized with $\bot_{\sqsubseteq}$. For any nodes i and $j \in N^+(i)$, when i receives a message from j (which is always a value $t \in X$), it stores this message in $i.m[j]$.

The algorithm. The algorithm is very simple — any node is always in one of two states: *sleep* or *wake*. All nodes start in the *wake* state, and if a node is in the *sleep* state, the reception of a message triggers a transition to the *wake* state.

In the *wake* state each node i asynchronously repeats $i.t_{cur} \leftarrow f_i(i.m)$. If there is no change in the value of $f_i(i.m)$ (compared to the last value $i.t_{old}$), it goes to the *sleep* state unless a message was received since $f_i(i.m)$ was computed. If $t_{old} \neq f_i(i.m)$ then $f_i(i.m)$ is sent to all the nodes in $N^-(i)$.

Concurrently with this we can run a termination detection algorithm, which will detect when all nodes are in the *sleep* state and no messages are in transit. Bertsekas has already addressed this problem with his termination detection algorithm [13], which directly applies, yielding only a constant overhead in the message complexity.

Asynchronous convergence theorem. We recall the definition of the Synchronous Convergence Condition (SCC) and the Box Condition (BC) (Section 6.2 [13]). Let X be any set, and $F : X^n \to X^n$ be any function with $F = \langle f_1, f_2, \ldots, f_n \rangle$.

Definition 4 (*SCC and BC*). Let $\{X(k)\}_{k=0}^{\infty}$ be a sequence of subsets $X(k) \subseteq X^n$ satisfying $\forall k.X(k+1) \subseteq X(k)$.

SCC. The sequence $\{X(k)\}_{k=0}^{\infty}$ satisfies the *Synchronous Convergence Condition* if

$$\forall x \in X(k).F(x) \in X(k+1)$$

and furthermore, if $\{y^k\}_{k\in\omega}$ is a sequence with $y^k \in X(k)$ for all k, then every limit point of $\{y^k\}_{k\in\omega}$ is a fixed point of F.

BC. The sequence $\{X(k)\}_{k=0}^{\infty}$ satisfies the *Box Condition* if for every k, there exist sets $X_i(k) \subseteq X$ such that

$$X(k) = \prod_{i=1}^{n} X_i(k).$$

We state a version of the Asynchronous Convergence Theorem that matches our notation. We need some preliminary terminology. Assume that before starting the asynchronous algorithm, the arrays of the nodes are initialized with a vector $\bar{x} \in X^n$, called the initial solution estimate. That is, for all nodes $i \in \{1, \ldots, n\}$, and all $j \in N^+(i)$ assume that $i.m[j] = \bar{x}_j$ and that $i.t_{old} = \bar{x}_i$.

A *limit point of the asynchronous algorithm* (with initial estimate $\bar{x}$) is a vector $\hat{x} \in X^n$ which can be written $\hat{x}_i = i.t_{cur}$, where there exists some state of the distributed system in which the algorithm has converged, and $i.t_{cur}$ is the current value of node i in this state (the algorithm has converged when all nodes are in the *sleep* state, and no messages are in transit).

Theorem 5 (*ACT [13]*). *Assume there exists a sequence of subsets $X(k) \subseteq X^n$ with $\forall k.X(k+1) \subseteq X(k)$, satisfying the SCC and the BC. Assume that the arrays of the nodes are initialized with $\bar{x} \in X(0)$, called the initial solution estimate. Then every limit point of the asynchronous algorithm is a fixed point of F.*

The ACT ensures that the algorithm converges to a fixed point of F under asynchronous computation. The algorithm computes $X(0), X(1), \ldots$ as a sequence of approximations ordered by the information ordering $\sqsubseteq$. Therefore, we set $X(0)$ to be any $\sqsubseteq$-approximation to the $\sqsubseteq$-least fixed point of F, such as the 'bottom' vector $(\bot_{\sqsubseteq}, \bot_{\sqsubseteq}, \ldots)$

2.3. Correctness

To prove the correctness of the asynchronous algorithm, we shall show that the assumption of the ACT is satisfied when all nodes initialize their trust values ($i.m$ and $i.t_{old}$) to $\bot_{\sqsubseteq}$, and that any limit point of the algorithm is the *least* fixed point of F. The following concept of an *information approximation* is central.

Definition 6 (*Information Approximation*). Let $F : X^n \to X^n$ be continuous. Say that a value $\bar{t} \in X^n$, is an *information approximation for F* if $\bar{t} \sqsubseteq F^*$ and $\bar{t} \sqsubseteq F(\bar{t})$.

The following Proposition 7 shows that we can indeed appeal to the ACT. In the following $(X, \sqsubseteq)$ is a finite height cpo, and $F : X^n \to X^n$ is continuous.

Proposition 7 (*Correctness*). *Let $\bar{t}$ be any information approximation for F. Assume that the arrays of the nodes are initialized with $\bar{t}$. Then there exists a decreasing sequence $\{X(k)\}$ of subsets of X^n with $\bar{t} \in X(0)$, satisfying the synchronous convergence condition and the box condition. Furthermore, any limit point of the asynchronous algorithm with initial estimate $\bar{t}$ is F^*.*

Proof. Define a sequence of subsets of X^n, $X(0) \supseteq X(1) \supseteq \cdots \supseteq X(k) \supseteq X(k+1) \supseteq \cdots$ by

$$X(k) = \{m \in X^n \mid F^k(\bar{t}) \sqsubseteq m \sqsubseteq F^*\}.$$

Note that $X(k+1) \subseteq X(k)$ follows from the fact that $F^k(\bar{t}) \sqsubseteq F^{k+1}(\bar{t})$ for any $k \in \mathbb{N}$, which, in turn, holds since $\bar{t}$ is an information approximation. For the synchronous convergence condition, assume that $m \in X(k)$ for some $k \in \mathbb{N}$. Since $F^k(\bar{t}) \sqsubseteq m \sqsubseteq F^*$, we get by monotonicity $F^{k+1}(\bar{t}) \sqsubseteq F(m) \sqsubseteq F(F^*) = F^*$. Let $(y_k)_{k\in\omega}$ be such that $y_k \in X(k)$ for every k.

Since $\bar{t}$ is an information approximation, we have $\bigsqcup_i F^i(\bar{t}) = F^*$. Since X is of finite height there exists $k_h \in \mathbb{N}$ so that for all $k' \geq k_h$, $X(k') = F^*$. Thus any such $(y_k)_{k\in\omega}$ converges to F^*. The box condition is also easy to show:

$$X(k) = \prod_{i=1}^{n} \{m(i) \in X \mid m \in X^n \text{ and } F^k(\bar{t}) \sqsubseteq m \sqsubseteq F^*\}. \quad \square$$

To prove the correctness of our algorithm, we simply invoke Proposition 7 in the case of the trivial information approximation $\bar{t} = \bot_{\sqsubseteq}^n$. The asynchronous convergence theorem ensures that the asynchronous algorithm converges towards the right values at all nodes, and, because of our assumption of finite height cpos, the distributed system will eventually reach a state which is stable. In this state, each node i will have computed f_i^*.

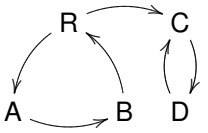


Fig. 1. Example dependency graph.

2.4. Communication requirements

Since any node sends values only when a change occurs, by monotonicity of f_i, node i will send at most $h \cdot |N^-(i)|$ messages, each of size $O(\log |X|)$ bits.[4] Node i will receive at most $h \cdot |N^+(i)|$ messages, each message (possibly) triggering a computation of f_i. Globally, the number of messages is $O(h \cdot |E|)$ each of bit size $O(\log |X|)$. Hence, the communication complexity of our algorithm is linear in the height of the lattice used by the policies. An important global invariant in this algorithm is that any value computed locally at a node (by the assignment $i.t_{cur} \leftarrow f_i(i.m)$) is a component in an information approximation for F. That is, it holds everywhere, at any time, that (1) $i.t_{cur} \sqsubseteq f_i^*$ and (2) $i.t_{cur} \sqsubseteq f_i(i.m)$. To see this, note that (1, 2) hold initially, and that both the properties are preserved by the update $i.t_{cur} \leftarrow f_i(i.m)$ whenever $i.m[y] \sqsubseteq f_y^*$ for all $y \in N^+(i)$ (which is always true). We state this fact as a lemma, as it becomes very useful in the next section where we consider fixed point approximation algorithms.

Lemma 8. *Any value $i.t_{cur} \in X$ computed by any node $i \in \{1, \ldots, n\}$, at any time in the algorithm by the statement $i.t_{cur} \leftarrow f_i(i.m)$, is a part of an information approximation for F, in the sense that $i.t_{cur} \sqsubseteq f_i^*$ and $i.t_{old} \sqsubseteq i.t_{cur}$.*

2.5. An example computation

In this subsection, we give a small example of a run of the asynchronous algorithm. Let us consider an example with 5 principals, named R, A, B, C and D. The example is meant to illustrate the situation where R wants to compute its trust value in a certain fixed subject S (which will not be involved in the computation). We will use the MN trust structure T_{MN} (Section 1) in the example.

Policies. The policies of the principals have the following entries for S.

$$\begin{aligned}
\pi_{\mathsf{R}} &= (\ulcorner \mathsf{A} \urcorner(S) \vee \ulcorner \mathsf{C} \urcorner(S)) \sqcup \mathtt{Loc}(S) \\
\pi_{\mathsf{A}} &= \ulcorner \mathsf{B} \urcorner(S) \sqcup \mathtt{Loc}(S) \\
\pi_{\mathsf{B}} &= \ulcorner \mathsf{R} \urcorner(S) \sqcup \mathtt{Loc}(S) \\
\pi_{\mathsf{C}} &= \ulcorner \mathsf{D} \urcorner(S) \sqcup \mathtt{Loc}(S) \\
\pi_{\mathsf{D}} &= \ulcorner \mathsf{C} \urcorner(S) \sqcup \mathtt{Loc}(S).
\end{aligned}$$

The construct $\ulcorner \cdot \urcorner$ is policy reference (as in Section 4.1), $\vee$ is $\preceq$-join and $\sqcup$ is $\sqsubseteq$-join. The construct $\mathtt{Loc}(S)$ is a special construct for the T_{MN} trust structure; it refers to the trust value derived from the local observations made by a principal about the subject in question (this construct is discussed also by Nielsen and Krukow [11]). For example, R's policy for the subject is to take the $\preceq$-join in T_{MN} of the values that A and C specify for the subject, and then the $\sqsubseteq$-join of this value and the trust value given by the local observations made by R about the subject.

It is not hard to see that both $(T_{MN}, \preceq)$ and $(T_{MN}, \sqsubseteq)$ are lattices, and that the joins are given by the following formulas; for any $(m, n), (m', n') \in T_{MN}$ we have:

$$\begin{aligned}
(m, n) \vee (m', n') &= (\max(m, m'), \min(n, n')) \\
(m, n) \sqcup (m', n') &= (\max(m, m'), \max(n, n')).
\end{aligned}$$

The dependency graph derived from the policies is given in Fig. 1.

Local data. We assume that the principals have the following local data, representing observations made about the subject.

[4] In fact, there will be *only* $O(h)$ *different* messages, each sent to all of $N^-(i)$. Consequently, a broadcast mechanism could implement the message delivery efficiently.

	R	A	B	C	D
$\mathtt{Loc}(S)$	$(0,0)$	$(1,5)$	$(3,0)$	$(2,5)$	$(4,6)$

The synchronous computation. Let us first illustrate the least fixed point of the policies by showing the sequence of computations corresponding to the "synchronous" iterations (i.e., $\bot_{\sqsubseteq}$, $\Pi_\lambda(\bot_{\sqsubseteq})$, $\Pi_\lambda(\Pi_\lambda(\bot_{\sqsubseteq}))$, ...). In the table below, column x of row $i+1$ is obtained by applying policy π_x to row i, e.g., the value $(3,5)$ in column A of row 2 is obtained by the following informal "calculation"

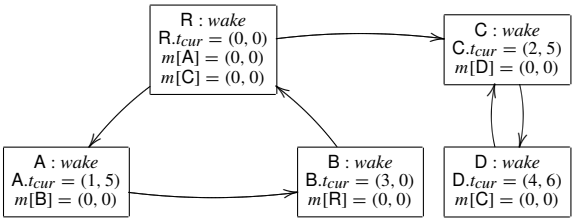
$$\ulcorner \mathsf{B} \urcorner(S) \sqcup \mathtt{Loc}(S) = (3,0) \sqcup (1,5) = (3,5).$$

It is easy to verify that the last row in the table below is the least fixed point of the policies (i.e., iterating round 6 will give the same row as iteration 5).

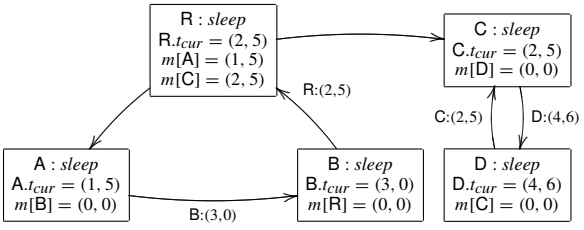
Iteration	R	A	B	C	D
0	$\bot_{\sqsubseteq}$	$\bot_{\sqsubseteq}$	$\bot_{\sqsubseteq}$	$\bot_{\sqsubseteq}$	$\bot_{\sqsubseteq}$
1	$(0,0)$	$(1,5)$	$(3,0)$	$(2,5)$	$(4,6)$
2	$(2,5)$	$(3,5)$	$(3,0)$	$(4,6)$	$(4,6)$
3	$(4,5)$	$(3,5)$	$(3,5)$	$(4,6)$	$(4,6)$
4	$(4,5)$	$(3,5)$	$(4,5)$	$(4,6)$	$(4,6)$
5	$(4,5)$	$(4,5)$	$(4,5)$	$(4,6)$	$(4,6)$

An asynchronous run. We now show a possible run of the asynchronous algorithm for the same set of policies as above. We illustrate the algorithm by showing the local states of the nodes in the network at various points in time. The nodes are denoted by boxes describing the local state in terms of values of arrays $i.m$, and the values of $i.t_{cur}$. Furthermore, messages that are in transit are visible on the "dependency" edges between the nodes. Note that messages "flow against" the direction of the arrowhead since arrows denote dependencies.

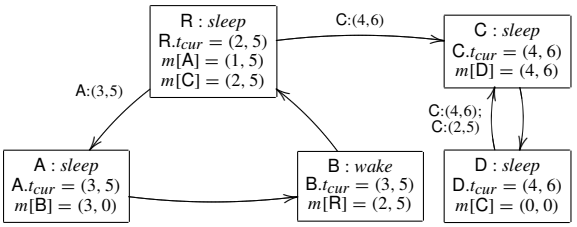
Network snapshot 1. We assume that the initial states of the nodes are given by the following. All nodes are *wake*, the arrays ($i.m$) are initialized to $\bot_{\sqsubseteq} = (0,0)$. Each node has $i.t_{cur} = \pi_i(i.m)$.



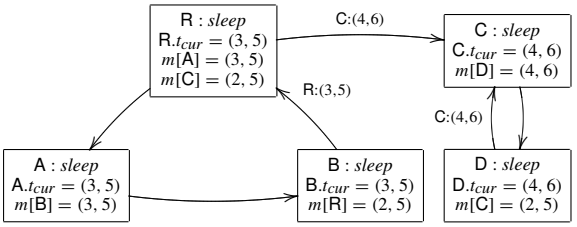
Network snapshot 2. Here R has received value $(1,5)$ from A and $(2,5)$ from C. Further values are in transit, e.g. value R : $(2,5)$ "on" edge B $\to$ R represents a message in transit from R to B.



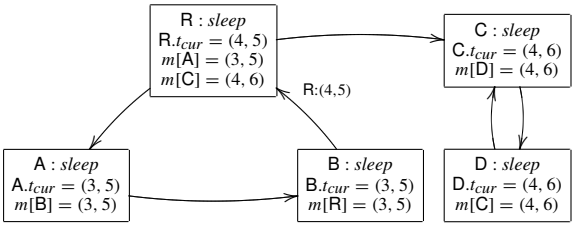
Network snapshot 3. Two messages are in transit on the (presumably slow) path from C to D (we are assuming a reliable network, so the first sent will also arrive first). B has just finished computing $\pi_{\mathsf{B}}(\mathsf{B}.m) = (3, 5)$, but has not yet sent this value.



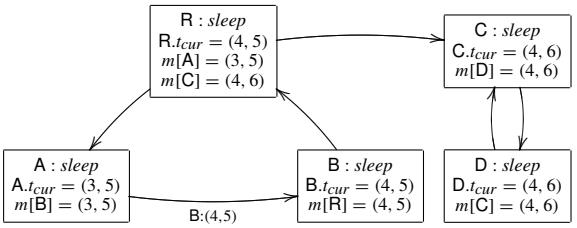
Network snapshot 4.



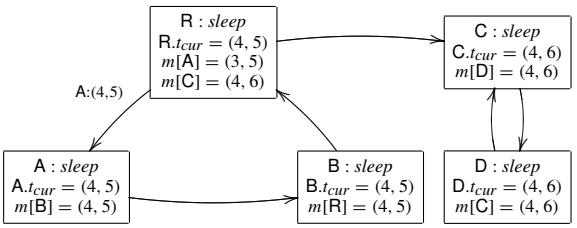
Network snapshot 5. Notice that the component consisting of C and D has converged. No more messages are exchanged between them for the remainder of the algorithm; this is in contrast to the globally synchronous iteration.



Network snapshot 6.



Network snapshot 7. When R receives the final value from A, the algorithm has converged.



3. Lower bounds

In this section we show a lower bound for the communication involved in the problem tackled in the previous section. The model is as described in Section 2.1, and in addition we assume that all processors have unlimited

computational power. The communication complexity of a protocol in a distributed network is the number of bits sent in total, over all edges of the network, in the worst case. For simplicity, we shall first consider the case where the cpo is the boolean lattice $X = (0 \sqsubseteq 1)$.

As stated in Section 2, the aim of the distributed algorithm is to compute local fixed points without computing the global state, if possible. Therefore in this section we consider the problem of computing a single component f_i^* of the lfp, rather than entire least fixed point F^*. Achieving good lower bounds for this appears to be a tricky problem. The algorithm of Section 2 uses $O(n^2)$ bits, and on the other hand, the trivial $\Omega(n)$ lower bound (by noting that some node must receive n bits to merely evaluate its function) does not capture the possible combinatorial nature of the *interactions* between the local policy functions.

3.1. Lower bound technique

We obtain a lower bound by considering a two-party version of the problem: partition the set of functions (nodes of the network) between two players, Alice and Bob. Alice knows all the functions in her partition and Bob knows all the functions in his partition. The number of bits that Alice and Bob must communicate for one of them to know the result is clearly a lower bound on the total communication incurred by any totally distributed algorithm.

Our result makes use of the set-disjointness problem: Alice and Bob each have a set, drawn from $\{1, \ldots, r\}$ and they must decide if their sets are disjoint. It is known that any protocol for this problem must communicate at least $\Omega(r)$ bits in the worst case, even if Alice and Bob are to know the result with probability at least $2/3$, i.e. a randomized protocol. A proof of this result can be found in [15], along with more details about the communication complexity model.

We can achieve a lower bound on our two-party problem by constructing a reduction from the set-disjointness problem: given two sets $P, Q \subseteq \{1, \ldots, r\}$, construct a set of functions $f_1 \ldots f_{n/2}$ for Alice using only knowledge of P and a set of functions $f_{n/2+1} \ldots f_n$ for Bob using only knowledge of Q such that knowing the least fixed point of the function $F = (f_1, \ldots, f_n)$ allows Alice or Bob to decide disjointness of P and Q (clearly once one player knows the result, the other can be informed with one extra bit of communication).

Lemma 9. *Given two sets $P, Q \subseteq \{1, \ldots, r\}$ for some number $r = \log n!$, there exists a pair of monotone boolean circuits $\langle f_1, \ldots, f_s \rangle$ and $\langle f_{s+1}, \ldots, f_{s+t} \rangle$ where $f_i : \{0,1\}^{s+t} \to \{0,1\}$ and $s, t = O(n)$. Letting $F = \langle f_1, \ldots, f_{s+t} \rangle$, then for some $i \in \{1, \ldots, s+t\}$, $f_i^* = 1$ iff P and Q are disjoint.*

Proof. The proof is by reduction from set-disjointness. Consider some permutation σ of $\{0, \ldots, n-1\}$. We can view σ as a set of size $\sum_{i=1}^{n} \lg i = \lg n! \approx n \lg n - n$ bits by partitioning it into blocks of length $\lg n, \lg(n-1), \ldots, 1$ bits where each block representing a binary expansion of some integer k has the meaning 'select the kth element not yet selected'. For example, the permutation $(0, 1, \ldots, n-1)$ corresponds to the empty set, and the permutation $(n-1, n-2, \ldots, 0)$ corresponds to the complement of the empty set. We will encode σ by revealing one bit at a time. We will use the notation $\sigma(i) \in \{0, \ldots, n-1\}$ to denote the ith element of the permutation, and $\sigma_i \in \{0,1\}^{\lg n - i}$ to denote the binary expansion of $\sigma(i)$. From now on we shall use σ, σ' to denote the two sets P, Q respectively.

The reduction proceeds as follows. Since the function $F = (f_1, \ldots, f_{s+t})$ is monotone, F^* exists and is unique. Hence we can simulate the network by computing $\mathbf{0}, F(\mathbf{0}), \ldots$ to find F^* and since F^* is unique, *any* algorithm must also reach this same final state. The idea is that after the kth iteration, the value of $F^{(k)}(\mathbf{0})$ will reveal disjointness of the sets $\bigcup_{i=0}^{k} \sigma_i$ and $\bigcup_{i=0}^{k} \sigma'_i$. We now show how such a network can be realised.

Define the symmetric k-threshold function as $Th_k(x_1, \ldots, x_n) = 1$ iff $x_1, \ldots, x_n$ contains a subset of size k. Note that with $\binom{n}{k}$ $\{\wedge, \vee\}$-gates we can construct the formula

$$Th_k(x_1, \ldots, x_n) = \underbrace{\bigvee_{i_1=1}^{n} \cdots \bigvee_{i_k=1}^{n}}_{i_1 < i_2 < \cdots < i_k} \left(\bigwedge_{j=1}^{k} x_{i_j} \right)$$

and note that $Th_0 = 1$. The permutation σ (and hence the set P) can be encoded into Alice's functions as follows. There are $s = 2n$ functions: set $f_i \equiv Th_{\sigma(i-1)}$ for $1 \le i \le n-1$ and $f_i \equiv Th_{\sigma(i-1)+1}$ for $n \le i \le 2n$.

Now we describe how to build Bob's functions, given the permutation σ' representing the set Q. The ith function f_{s+i} outputs 1 iff the sets σ_{i-1} and σ'_{i-1} are disjoint, and these sets are revealed by Alice on the ith iteration. Bob can

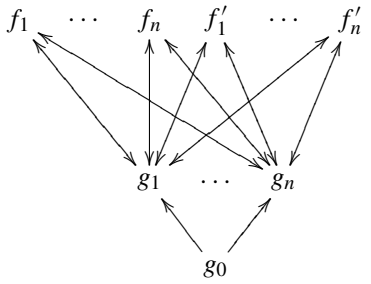


Fig. 2. The dependency graph for the lower bound construction.

compute a set of functions as follows: let S_i be the set of indices such that if $\sigma(i) \in S_i$ then $\sigma_i \cap \sigma'_i = \emptyset$. The functions can detect the value of $\sigma(i)$ by comparing the outputs of Alice's functions on the previous iteration with the current outputs, and seeing which bit has been raised (recall that we encode $\sigma(i-1) = j$ on the ith iteration by raising the jth nonzero bit of the outputs).

Let Bob's functions take as input $x_1 \dots x_n, y_1, \dots, y_n$, the current output of Alice's functions where (by construction) the second half of this string is the same as the first half on the previous iteration. Then Bob can detect which bit was raised by comparing the two halves. Since we are restricted to monotone functions, we cannot simply use a difference operator, instead the ith function can have a disjunction over the possible substrings of the input that would encode a $\sigma(i-1)$ that was also in the set S_{i-1}. We shall denote by $s_{j1} \dots s_{jk}$ the indices of the jth possible substring that would satisfy the condition on the kth iteration, i.e. the input should satisfy $x_{s_{j1}} \wedge \dots \wedge x_{s_{jk}} \wedge y_{s_{j1}} \wedge \dots \wedge y_{s_{j,k-1}}$. Bob's functions can then be written as:

$$
\begin{aligned}
f_{s+1} &= \bigvee_j x_{s_{j1}} \\
f_{s+2} &= \bigvee_j \left(x_{s_{j1}} \wedge x_{s_{j2}} \wedge y_{s_{j1}}\right) \\
f_{s+3} &= \bigvee_j \left(x_{s_{j1}} \wedge x_{s_{j2}} \wedge x_{s_{j3}} \wedge y_{s_{j1}} \wedge y_{s_{j2}}\right) \\
&\vdots \\
f_{s+n}.
\end{aligned}
$$

The circuits are connected together by connecting the outputs of Bob's functions to be the inputs of Alice's functions, and vice versa. This gives the dependency graph of Fig. 2.

Now, all the counter-examples to the disjointness of P, Q lie in the region strictly between $\mathbf{0}$ and $\mathbf{1}$. We now claim that Bob can compute disjointness by knowing f^*_j for some single component j. More precisely, we claim that $f^*_{s+1} \wedge \dots \wedge f^*_{s+n} = 1$ iff the sets P, Q are disjoint. In fact, this is quite easy to see by considering what happens as one applies the functions, starting from the all-zero vector $\mathbf{0}$: after k iterations of F, the outputs of Alice's functions encode the set σ_k and the output of Bob's functions is $\underbrace{1 \dots 1}_{k} 0 \dots 0$ iff the sets $\bigcup_{i=0}^{k} \sigma_i$ and $\bigcup_{i=0}^{k} \sigma'_i$ are disjoint. The claim then follows by considering the outputs at the $(s+t)$th iteration, and noting that after this many iterations there can be no changes in the outputs so it must also be the least fixed point. □

Since the conjunction $f^*_{s+1} \wedge \dots \wedge f^*_{s+n}$ can be computed by a single node say f_0, and since any monotone function has a monotone circuit computing it, the main result follows: any protocol that terminates with some node knowing f^*_0 with probability p can be used to solve the set-disjointness problem on sets of size $\Omega(n \log n)$ with probability p. Therefore any protocol that solves our problem, even with probability greater than $2/3$, must communicate at least $\Omega(n \log n)$ bits across the cut between Alice and Bob.

A two-party upper bound. In the two-party case, the obvious iterative algorithm is optimal. Suppose each party A, B has a monotone function $f_A, f_B : \{0,1\}^{2n} \to \{0,1\}^n$. Let $a_0 = b_0 = \mathbf{0}$. Party A starts by sending $a_1 = f_A(a_0 b_0)$ to B, then B computes $b_1 = f_B(a_0 b_0)$, and sends to A the set of indices of the bits that changed in b_1. This continues

until there is no change in either a_i or b_i, in which case $a_i b_i = F^*$. Since there can be no more than $2n$ iterations, this protocol communicates $O(n \log n)$ bits.

3.2. A lower bound for lattices of arbitrary height

We can now state the main result of this section by extending the boolean lower bound to functions on more general partial orders X of height h, as is the case with the trust structure model. Here the difference between the trivial lower bound and ours is significant: the trivial lower bound (for a processor to evaluate its function) is $\Omega(n \log h)$ bits and we now prove an $\Omega(nh \log n)$ lower bound. Thus, the communication must grow linearly with the lattice height, in the worst case.

The lower bound can be extended by considering how many ways the nodes can count by monotonically changing the bits on outputs to edges on the cut. This is illustrated by the following problem. We have hn balls (representing the maximum number of steps) and n bins (representing the outputs) each of capacity h (since we can assign h different values to each output). At each step, we throw a ball into an unfilled bin (this corresponds to the fact that we must change at least one output at each step). The question is how many different ways can we throw the balls into the bins? For our result we are only interested in the logarithm of this number, which is quite easy to bound as follows. An upper bound can be obtained by assuming that *every* ball has a choice of n bins (i.e. they do not fill up); then the order taken by the balls gives $\log n^{hn} = hn \log n$ bits of information. A lower bound can be obtained by assuming that the first h balls each have a choice of n bins, the second h only have $n - 1$ choices, and so on. This way, the order that the balls are assigned to bins gives $\log n^h (n-1)^h \cdots 1^h = h \log n! \approx h(n \log n - n)$ bits of information. But asymptotically these are equal, so the correct bound must be $\Theta(hn \log n)$ bits.

The relationship to the lower bound is as follows. Recall that the sets are encoded by considering the number of distinct ways of raising the outputs of Alice's functions. When the functions are boolean, there are exactly $n!$ ways of doing this, which gives the lower bound of $\log n! \approx n \log n$. With h different possible output values for each node, the above analogy gives the number of different ways that the outputs can be raised (while obeying monotonicity). It follows that, for partial orders of height h, the communication complexity of the two-party problem is $\Omega(nh \log n)$ bits, and this lower bound applies to any distributed algorithm.

3.3. Complexity of computing the dependencies

We have so far assumed that each node knows the other nodes that its function depends on. Here we look at how difficult this may be. If the function f_i depends on all its variables then building the dependency graph is easy. For arbitrary boolean functions, the following simple theorem shows that computing dependencies is not likely to be efficient. Define the problem DEPENDENCY: given a boolean function f on n variables $x_1, \ldots, x_n$ and an integer $1 \leq k \leq n$, does f depend on the variable x_k?

Theorem 10. *The problem DEPENDENCY is NP-complete.*

Proof. We first note that the problem is in NP since if f depends on x_k this can be verified by nondeterministically guessing an assignment $X = x_1, \ldots, x_n$ such that $f(x_1, \ldots, x_k, \ldots, x_n) \neq f(x_1, \ldots, \overline{x_k}, \ldots, x_n)$. We can prove NP-hardness by reduction from VALIDITY: a function f is valid iff $f(0, \ldots, 0) = 1$, and f depends on none of its variables. Deciding the validity of f is known to be NP-hard [16] and so deciding dependency is NP-complete. □

Remarks. We note that if the function f is monotone then we can solve the DEPENDENCY problem in polynomial time using the Quine–McCluskey algorithm [17]. Since we are considering systems containing only monotone policy functions, computing the dependencies is therefore not a problem.

4. Approximation techniques

In this section, we present two novel techniques for safe approximation of a component in the global trust state. Consider a situation in which a client principal p wants to access a resource controlled by server v. Assume that the access control policy of v is that, to allow access, its trust in p should be trustwise above some threshold $t_0 \in X$, i.e., the fixed point should satisfy $t_0 \preceq (\mathsf{lfp}\, \Pi_\lambda)(v)(p)$. The goal of the approximation techniques is to allow the server to (soundly) make its security decision without having to actually compute the exact fixed point value. Instead, the

server is able to efficiently compute a global trust state $\bar{p} : \mathcal{P} \to \mathcal{P} \to X$, which approximates the actual global trust state in the following sense. If the server was to grant some security property to the client in the trust state $\bar{p}$ then it would also grant the same property in the actual global trust state.

We begin by giving some preliminary definitions. By $x \prec C$ we mean that for all $y \in C$, $x \prec y$. Let $T = (X, \preceq, \sqsubseteq)$ be a trust structure, i.e. having an information ordering $(X, \sqsubseteq)$ with a bottom element $\bot_{\sqsubseteq}$ and a trust ordering $(X, \preceq)$ with a bottom element $\bot_{\preceq}$. Let $C = \{x_i \in X \mid i \in \mathbb{N}\}$ be a $\sqsubseteq$-chain.[5] The ordering $\preceq$ is $\sqsubseteq$-*continuous* if for any $x \in X$ we have (i) $x \preceq C$ implies $x \preceq \bigsqcup C$ and (ii) $C \preceq x$ implies $\bigsqcup C \preceq x$ (i.e. if an element is less than all the elements of a $\sqsubseteq$-chain then it is also less than its limit). We shall require continuity in the case of infinite height orderings, for example where the trust values are natural numbers.

4.1. Bounding "bad behaviour"

The first technique we present lets a client convince a server that, in the global trust state, the trust value it assigns to the client is above a certain trust value (wrt the trust ordering). The technique is based on the following proposition.

Proposition 11. *Let $(X, \preceq, \sqsubseteq)$ be a trust structure in which $\preceq$ is $\sqsubseteq$-continuous. Let $\bar{p} \in X^n$, and $F : X^n \to X^n$ be any function that is $\sqsubseteq$-continuous and $\preceq$-monotonic. If we have $\bar{p} \preceq (\lambda k \in \{1, \ldots, n\}.\bot_{\sqsubseteq})$ and $\bar{p} \preceq F(\bar{p})$, then $\bar{p} \preceq \mathsf{lfp}_{\sqsubseteq} F$.*

Proof. We have $\bar{p} \preceq \lambda k.\bot_{\sqsubseteq}$, which implies $F(\bar{p}) \preceq F(\lambda k.\bot_{\sqsubseteq})$ by $\preceq$-monotonicity. Since $\bar{p} \preceq F(\bar{p})$, transitivity implies that $\bar{p} \preceq F(\bar{p}) \preceq F(\lambda k.\bot_{\sqsubseteq})$. So again by $\preceq$-monotonicity of F and transitivity

$$\bar{p} \preceq F(\bar{p}) \preceq F^2(\bar{p}) \preceq F^2(\lambda k.\bot_{\sqsubseteq}).$$

Now since for all $i \geq 0$ we have $\bar{p} \preceq F^i(\lambda k.\bot_{\sqsubseteq})$, the fact that $\preceq$ is $\sqsubseteq$-continuous implies that

$$\bar{p} \preceq \bigsqcup_i F^i(\lambda k.\bot_{\sqsubseteq}) = \mathtt{lfp}_{\sqsubseteq} F. \quad \square$$

Note that the conclusion of the proposition is an assertion that is useful for authorization; if the server knows a $\bar{p} \in X^n$ that is sufficient to allow an authorization, and also knows that $\bar{p} \preceq \mathsf{lfp}_{\sqsubseteq} F$ then it should allow the authorization, since the global trust state of the system is higher in the trust ordering than the state $\bar{p}$. Since the server does not need to explicitly compute $\mathsf{lfp}_{\sqsubseteq} F$, this idea is the basis of an efficient protocol for a kind of "proof carrying" authorization (furthermore, it may be impossible to compute the actual trust state if the lattice has infinite height).

Consider for simplicity the "MN" trust structure T_{MN} from Section 1, which satisfies the information continuity requirement. Recall that, in this structure, trust values are pairs (m, n) of natural numbers, representing $m + n$ past interactions; m of which where classified 'good', and n, classified as 'bad'.

Suppose principal p wants to efficiently convince principal v, that v's trust value for p is a pair (m, n) with the property that n is less than some fixed bound $N \in \mathbb{N}$ (i.e., giving v an upper bound on the amount of recorded "bad behaviour" of p). Let us assume that v's trust policy π_v is monotonic, also with respect to $\preceq$, and that it depends on a large set S of principals. Assume also that it is sufficient that principals a and b in S have a reasonably "good" trust value for p, to ensure that v's trust value for p is not too "bad". An example policy with this property could be written in the language of Carbone et al. [8] as

$$\pi_v \equiv \lambda x : \mathcal{P}.(\ulcorner a \urcorner(x) \wedge \ulcorner b \urcorner(x)) \vee \bigwedge_{s \in S \setminus \{a,b\}} \ulcorner s \urcorner(x).$$

The construct $\ulcorner \cdot \urcorner$ represents *policy reference* or *delegation*, e.g., if a and x are principal identities then expression $\ulcorner a \urcorner(x)$ "evaluates" to the value that a's trust policy specifies for x. The construct $e \vee e'$ represents least upper bound in the trust ordering (intuitively, "trustwise maximum" of e and e'), and similarly $\wedge$ represents greatest lower bound ("trustwise minimum").[6] Thus, informally, the above policy says that any principal p should have "high trust" with

[5] A $\sqsubseteq$-chain is a subset of elements from a partial order, where the elements in the chain are totally ordered by $\sqsubseteq$.

[6] The example policy assumes that $(X, \preceq)$ is a lattice, meaning that for any $x, y \in X$ both $x \vee y$ and $x \wedge y$ exist. Furthermore operations $\vee$ and $\wedge$ must be continuous *also with respect to the information ordering*. In many trust structures this is often the case [8].

a and b, or, with *all of* $s \in S \setminus \{a, b\}$, for the v to assign "high trust" to p. Now, if p knows that it has previously performed well with a and b, and knows also that v depends on a and b in this way, it can engage in the following protocol.

Protocol. Principal p sends to v the "trust state"

$$t = [(v, p) \mapsto (0, N), (a, p) \mapsto (0, N_a), (b, p) \mapsto (0, N_b)]$$

which can be thought of as a "proof" stating that $(0, N) \preceq (\mathsf{lfp}\, \Pi_\lambda)(v)(p)$. Upon reception, v first extends t to a global trust state, which is the extension of t to a function $\bar{p}$ of type $\mathcal{P} \to \mathcal{P} \to T_{MN}$, given by

$$\bar{p} = \lambda x \in \mathcal{P} \lambda y \in \mathcal{P}. \begin{cases} (0, N) & \text{if } x = v \text{ and } y = p \\ (0, N_a) & \text{if } x = a \text{ and } y = p \\ (0, N_b) & \text{if } x = b \text{ and } y = p \\ (0, \infty) & \text{otherwise.} \end{cases}$$

To check the proof, principal v must verify that $\bar{p}$ satisfies the conditions of Proposition 11. First, v must check that $\bar{p}(x)(y) \preceq \bot_\sqsubseteq = (0, 0)$ for all x, y. But this holds trivially if $y \neq p$ or $x \neq v, a, b$ because then $\bar{p}(x)(y) = (0, \infty) = \bot_\preceq$. For the other few entries it is simply an order-theoretic comparison $\bar{p}(x)(y) \preceq (0, 0)$. Now v tries to verify that $\bar{p} \preceq \Pi_\lambda(\bar{p})$. To do this, v verifies that $(0, N) \preceq \pi_v(\bar{p})(p)$. If this holds then v sends the value t to a and b, and ask a and b to perform a similar verification (e.g. $(0, N_a) \preceq \pi_a(\bar{p})(p)$). Then a and b reply with 'yes' if this holds and 'no' otherwise. If both a and b reply 'yes', then v is sure that $\bar{p} \preceq \Pi(\bar{p})$: by the checks made by v, a and b, we have that $\bar{p}(x)(y) \preceq \Pi_\lambda(\bar{p})(x)(y)$ holds for pairs $(x, y) = (v, p), (a, p), (b, p)$, but for all other pairs it holds trivially since $\bar{p}$ is the $\preceq$-bottom on these. By Proposition 11, we have $\bar{p} \preceq \mathsf{lfp}\, \Pi_\lambda$, and so, v is *ensured* that its trust value for p is $\preceq$ greater than $(0, N)$.

We have illustrated the main idea of the protocol by way of an example, but the general technique for verifying a proof should be clear. In general, the proof $\bar{p}$ may include a larger number of principals, which would then have to be involved in the verification process.

Remarks. Our approximation protocol has very much the flavour of a proof carrying authorization: the requester (or prover) must provide a proof that its request should be granted. It is then the job of the service provider (or verifier) to check that the proof is correct. The strength of this protocol lies in replacing an entire fixed point computation with a few local checks made by the verifier, together with a few checks made by a subset of the principals that the verifier depends on. An interesting property of this protocol is that part of the information that the prover needs to supply should already be known to the prover; it should already know with whom it has performed well with in the past (e.g. in our example above, p could know the bounds N_a and N_b because of its previous interaction with a and b).

There are two important restrictions imposed by this approach. First, in order to construct its proof, the prover needs information about the verifier's trust policy and of the policies of those whom the verifier depends on. If policies are secret or not publicly disclosed, it is not clear that the verifier would be able to construct the required proof. Secondly, because of the requirement in Proposition 11 that $\bar{p} \preceq \bot_\sqsubseteq$, the protocol can usually only be used to prove that the trust in the client represents "not too much bad behaviour," and *not* properties guaranteeing "sufficiently good" behaviour. Later, we present a more general protocol that helps to remove some of these restrictions.

Notice that the protocol for exploiting Proposition 11 has a message complexity that is independent of the height of the cpo; in particular, it works also for infinite height cpos. In contrast, the algorithm for computing the least fixed point has message complexity linear in h in the worst case.

4.2. The complexity of constructing short proofs

Let the size of a proof in the previous protocol be the number of principals involved in the proof carrying request (and hence the verification protocol). Even if the prover p knows all the local policies, the following theorem shows that there is unlikely to be an efficient procedure to construct small proofs in all cases.

Theorem 12. *For a network of mbfs, it is NP-hard to approximate to within any constant factor the minimum size of an assignment* X *where* $X \preceq F^*$ *and* $X(v) = 1$.

Proof. We will consider an easier problem: imagine that there is an oracle that answers questions of the form $X \preceq F^*$, for any assignment X. We construct a very simple reduction from the following NP-complete problem:
MONOTONE MINIMUM SATISFYING ASSIGNMENT [16]: Given a monotone formula $f(x_1, \ldots, x_n)$ over the basis $\{\vee, \wedge\}$, a satisfying assignment is an assignment $(v_1, \ldots, v_n)$ such that $f(v_1, \ldots, v_n) = 1$. What is the minimum size of a satisfying assignment of f?

Monotone minimum satisfying assignment is NP-hard to approximate within any constant factor, by reduction from minimum hitting set. Now, given a monotone function g, build a network with nodes p,v and $v_1 \ldots v_n$ for the variables in f. Set $f_p = v$, $f_{v_i} = 1$ and $f_v = g$. Then p has a proof that $f_p^* = 1$ of size k, iff g has a satisfying assignment X of size k. Of course, v needs to check that $X <= F(X)$ and $X \preceq F^*$, but the latter can be done with a single call to the oracle.

Note that if X is a satisfying assignment of $g = f_v$ then $X \preceq F(X)$ is satisfied: let X' be the extended state where everything not in X gets 0. Then $F(X')(v) = 1$ since X' is a satisfying assignment for f_v, and $F(X')(v_i) = 1$ for all v_i since they are just constants, and $F(X')(p) = X'(p)$. Hence $F(X') \geq X$. Also, $X \preceq F^*$ is also trivial since there is only one fixed point and it is just the evaluation of the function $g(1, \ldots, 1)$. □

It is not difficulty to show that there are networks where $\Omega(n)$ bits are required, since we can construct a set of functions with 2^n different satisfying assignments: for each subset S of $\{1, \ldots, n\}$, S is a minimum satisfying assignment of

$$f_S = \bigwedge_{i \in S} x_i.$$

Hence $\Omega(n)$ bits are required to describe a satisfying assignment of at least one function f_S.

4.3. A proof carrying protocol using snapshots

In this section we present a new theorem that gives a different proof carrying protocol that requires more communication, but does not have the two restrictions of the previous protocol.

The approximation technique developed in this section is different to the protocol in the previous subsection. In particular, we do not require the prover (client) to provide any information. We show that if we can take a snapshot of the system as it approaches the true fixed point from below (and hence satisfies some properties) then we can use this to derive a suitable approximation for the correct trust value. Usefully, the snapshot can be obtained by essentially 'pausing' the asynchronous fixed point algorithm from Section 2.2 at some point. The verifiers (servers) are then able to make a collection of local checks on this snapshot, allowing them to infer that the fixed point value must be trustwise above the snapshot value. The technique is based on the following proposition.

Proposition 13. *Let $(X, \preceq, \sqsubseteq)$ be a trust structure in which $\preceq$ is $\sqsubseteq$-continuous. Let $\bar{t} \in X^n$, and $F : X^n \to X^n$ be any function that is $\sqsubseteq$-continuous and $\preceq$-monotonic. Assume that $\bar{t}$ is an information approximation for F. If $\bar{t} \preceq F(\bar{t})$ then $\bar{t} \preceq \mathsf{lfp}\, F$.*

Proof. Since $\bar{t}$ is an information approximation for F, we have by easy induction that for all $k \in \mathbb{N}$, $F^k(\bar{t}) \sqsubseteq F^{k+1}(\bar{t}) \sqsubseteq \mathsf{lfp}\, F$, and so by continuity of F, $\bigsqcup_{k \in \mathbb{N}} F^k(\bar{t}) = \mathsf{lfp}\, F$. Since $\bar{t} \preceq F(\bar{t})$, an easy induction gives $\bar{t} \preceq F^k(\bar{t})$ for all k. Then the information continuity of $\preceq$ implies that $\bar{t} \preceq \mathsf{lfp}\, F$. □

This proposition is very useful because, by Lemma 8, a global invariant in the asynchronous fixed point algorithm is that all values computed are information approximations for F. This means that we can combine the algorithm with a distributed protocol that checks the condition $\bar{t} \preceq F(\bar{t})$ in the above proposition.

Imagine that during the execution of the asynchronous algorithm, there is a point in time in which no messages are in transit, all nodes i have computed their function f_i, and sent the value $f_i(i.m)$ to all that depend on it. Thus we have a "consistent" state in the sense that for any node x and any node $y \in N^+(x)$ we have $x.m[y] = y.t_{cur}$. In particular if x and z both depend on y, then they agree on y's value: $x.m[y] = y.t_{cur} = z.m[y]$. In this ideal state, there is a consistent vector $\bar{t}$ which by Lemma 8, is an information approximation for F, i.e. $\bar{t}$ contains the values $\bar{t}_i = i.t_{cur}$ for nodes $i \in \{1, \ldots, n\}$. If the state of the system was frozen at this point and all nodes x simultaneously make the check $x.t_{cur} \preceq f_x(x.m)$, then vector $\bar{t}$ satisfies $\bar{t} \preceq F(\bar{t})$. Since $\bar{t}$ is an information approximation for F, by Proposition 13, the root node R knows that $\bar{t}_R \preceq \mathsf{lfp}\, F_R$, which is what we want.

Process: non-root nodes i

$||_{\{\texttt{A},\texttt{B},\texttt{C}\}}$

$\texttt{A}$: **receive** (`init`);

$||_{\{\texttt{A1},\texttt{A2}\}}$

$\texttt{A1}$: $||_{c\in i.S}\ c$: **send** (`init`) **to** c;

$\texttt{A2}$: [//wait until consistent state];

$i.t_{app} \leftarrow i.t_{cur}$;

$||_{j\in N^-(i)}\ j$: **send** (`copy`) **to** j;

$\texttt{B}$: $||_{\{\texttt{B1},\texttt{B2}\}}$

$\texttt{B1}$: $||_{k\in N^+(i)}$

k : **receive** (`copy`) **from** k;

$i.m_{app}[k] \leftarrow i.m[k]$;

$\texttt{B2}$: **join** {`B1`, `A2`} **then**

$i.b$: bool $\leftarrow (i.t_{app} \preceq f_i(i.m_{app}))$;

$\texttt{C}$: $||_{\{\texttt{C1},\texttt{C2}\}}$

$\texttt{C1}$: $||_{c\in i.S}$

c : **receive** ($i.b_c$: bool) **from** c;

$\texttt{C2}$: **join** {`C1`, `B2`} **then**

send $(i.b \wedge (\bigwedge_{c\in i.S} i.b_c))$ **to** $i.p$;

Fig. 3. Snapshot algorithm — Generic node behaviour.

We now describe how to obtain suitable snapshots from the distributed asynchronous fixed point algorithm (i.e. where no messages are in transit and each nodes have computed their function for this step). Assume that the asynchronous algorithm is running, and at some point the root node decides to run the approximation check, perhaps because it has computed a (possibly non-fixed point) value $R.t_{cur}$, which is sufficient to allow access. We assume that each node $i \in \mathcal{P}$ has additional variables $i.t_{app} : X$ and $i.m_{app} : X$ array, indexed by $N^+(i)$. The array will eventually store only consistent values. The algorithm, as usual, consists of a special process run by the root, and another similar process running at non-root nodes, given by Fig. 3.

Assume that we have a spanning tree T_R, rooted at R and each node knows the sets $N^+(i)$ and $N^-(i)$ as in Section 2. The root initiates the approximation algorithm. It starts by sending an init message to each of its children, stored in $R.S$ (Fig. 3, label A1). Now it waits until it is in a *locally consistent* state (A2), which means that, in the asynchronous algorithm, it has just computed $R.t_{cur} \leftarrow f_R(R.m)$, and (if necessary) has sent that value to each of $N^-(R)$. Once in such a state, R saves the value by doing $R.t_{app} \leftarrow R.t_{cur}$ — this value will become the value of R in the consistent vector we are seeking. R now sends a copy message to each node in $N^-(R)$ (A2). A node $y \in N^-(R)$ that receives a copy message from R will copy the last value received from R into its approximation array, i.e. $y.m_{app}[R] \leftarrow y.m[R]$ (B1). Since we are assuming a reliable network, the copied value is $R.t_{app}$, and so we are propagating consistent values. Root R now waits until each node $z \in N^+(R)$ has sent a copy message, and computes $R.t_{app} \preceq f_R(R.m_{app})$ (B2). Finally, the root waits for all children in the spanning tree to have replied with a boolean value, and if all of these are true and the check succeeded (C1, C2), then the root is ensured that $R.t_{app} \preceq (\mathsf{lfp}\ F)_R$. Non-root nodes i, once initiated, do almost the same. The only difference is that after the check has been made, and all children in the spanning tree have replied with a boolean, i sends value true to its parent $i.p$ only if all i's children sent true and i's own check succeeded.

Since there is a constant number of messages sent for each edge in G_R, the message complexity of the snapshot algorithm is $O(|E|)$ messages, each of size $O(1)$ bits.

A useful property of this algorithm is that it can be run concurrently with the asynchronous fixed point algorithm; there is no reason to stop! One may simply allocate a thread implementing the approximation check that runs concurrently with the asynchronous fixed point algorithm.

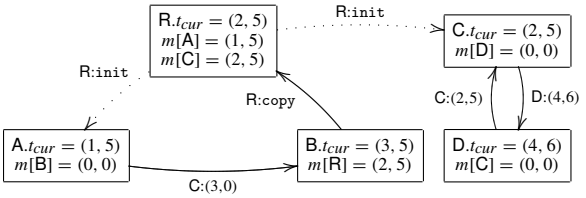
Note that, the style of this protocol is different from that of the previous section. In the previous protocol the client presents a "proof" t that the servers then verify. It is not clear how one could use Proposition 13 in this style. In

particular, if a client presented a "proof" t, then it is not clear how a distributed algorithm could check that $t \sqsubseteq \mathsf{lfp}\ F$ without already knowing $\mathsf{lfp}\ F$.

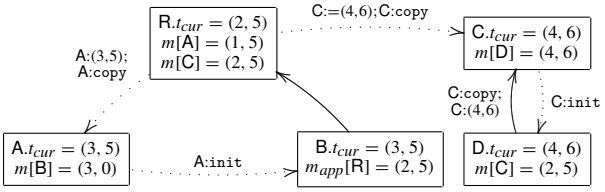
4.4. An example run

We illustrate the snapshot algorithm on the policies from Section 2.5. Again we illustrate the algorithm by a sequence of network states. We assume that the asynchronous algorithm has been running for some time (as in Section 2.5). Let us assume that R now wants to run the snapshot algorithm with respect to its current value, (2, 5).

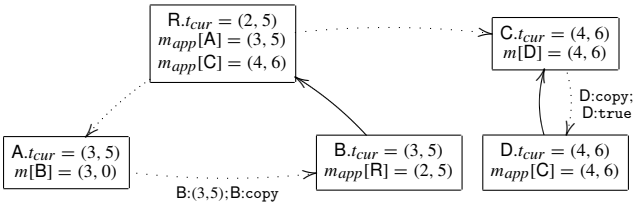
Network snapshot 1. R initiates the algorithm by sending `init` messages to A and C. Concurrently it sends a `copy` message to B to indicate that the last message received by B from R should be used for approximation. (Note that it is not a problem that B receives the `copy` message before it is initiated.)



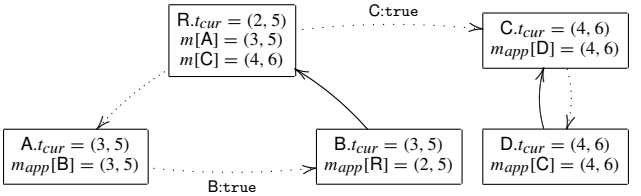
Network snapshot 2. Here C has received message D : (4, 6). It then receives the R : `init` message. C proceeds by sending its current value to R and D, followed by `copy` messages, and an `init` message (only) to D. A behaves similarly.



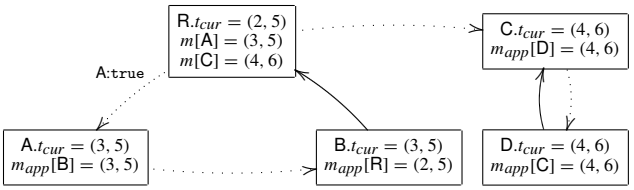
Network snapshot 3. Here D has asserted that $\mathsf{D}.t_{cur} \preceq \pi_{\mathsf{D}}(\mathsf{D}.m_{app})$ is true. Since it has no children in the spanning tree, it immediately sends value `true` to its (spanning tree) parent C (illustrated by the dotted edges).



Network snapshot 4. B has made its assertion, and similarly, once C receives `true` from D, it makes assertion $\mathsf{C}.t_{cur} \preceq \pi_{\mathsf{C}}(\mathsf{C}.m_{app})$, and sends `true` to R.



Network snapshot 5. Finally, A makes its assertion, and once R receives value `true`, R knows that $(2,5) \preceq \mathsf{lfp}(\Pi_\lambda)(\mathsf{R})(\mathsf{S})$ (recall that S is the subject of the trust computation).



4.5. Dual propositions and generalization

Note that both the propositions in this section have "dual" versions.

Proposition 14. *Let $(X, \preceq, \sqsubseteq)$ be a trust structure in which $\preceq$ is $\sqsubseteq$-continuous. Let $\bar{p} \in X^n$, and $F : X^n \to X^n$ be any function that is $\sqsubseteq$-continuous and $\preceq$-monotonic. If $\bot_\sqsubseteq \preceq \bar{p}$ and $F(\bar{p}) \preceq \bar{p}$ then $\mathsf{lfp}\, F \preceq \bar{p}$.*

The dual of Proposition 13 is the following.

Proposition 15. *Let $(X, \preceq, \sqsubseteq)$ be a trust structure in which $\preceq$ is $\sqsubseteq$-continuous. Let $\bar{t} \in X^n$, and $F : X^n \to X^n$ be any function that is $\sqsubseteq$-continuous and $\preceq$-monotonic. Assume that $\bar{t}$ is an information approximation for F. If $F(\bar{t}) \preceq \bar{t}$ then $\mathsf{lfp}\, F \preceq \bar{t}$.*

We can deploy similar algorithms for the duals. At first sight Proposition 14 does not seem as useful as its dual. The conclusion $\mathsf{lfp}\, F \preceq \bar{p}$ can usually only be used to *deny* a request, and a prover in the protocol for Proposition 14 would probably not be interested in supplying information which would help in refuting its request. However, this is not always so. For example, suppose one is using trust structures conveying probabilistic information (e.g. [18, 11]), and that $\bar{p} \preceq \bar{p}'$ expresses (informally) that, when interacting with a certain principal, the probability of a specific outcome given $\bar{p}$, is lower than the probability of that outcome given $\bar{p}'$. In this case, an assertion of the form $\mathsf{lfp}\, F \preceq \bar{p}$, can convince the verifier that when interacting with the prover, the probability of a "bad" outcome is *below* a certain threshold.

Essentially, we can use the same algorithm as that of Section 4.3 for exploiting Proposition 15. Servers can incorporate the check $F(\bar{t}) \preceq \bar{t}$ together with the dual check $\bar{t} \preceq F(\bar{t})$.

4.6. A more general class of proof carrying protocols

Interestingly, it turns out that the two propositions of this section are actually instances of a more general theorem, which gives rise to a generalized approximation protocol, that can be seen as a combination of the two techniques presented in this section.

Proposition 16. *Let $(X, \preceq, \sqsubseteq)$ be a trust structure in which $\preceq$ is $\sqsubseteq$-continuous. Let $\bar{p} \in X^n$, and $F : X^n \to X^n$ be any function that is $\sqsubseteq$-continuous and $\preceq$-monotonic. Assume that $\bar{p}$ satisfies $\bar{p} \preceq F(\bar{p})$. If there exists an information approximation $\bar{t} \in X^n$ for F, with property that $\bar{p} \preceq \bar{t}$, then $\bar{p} \preceq \mathsf{lfp}\, F$.*

Proof. The proof of Proposition 16 is similar to that of Proposition 11. We use the diagram:

$$\begin{array}{ccccccccc} \bar{p} & \preceq & F(\bar{p}) & \preceq & \cdots & \preceq & F^i(\bar{p}) & \preceq & \cdots \\ \wedge & & \wedge & & & & \wedge & & \cdots \\ \bar{t} & \sqsubseteq & F(\bar{t}) & \sqsubseteq & \cdots & \sqsubseteq & F^i(\bar{t}) & \sqsubseteq & \cdots \end{array}$$

By the continuity of $\preceq$ we have $\bar{p} \preceq \bigsqcup_i F^i(\bar{t})$. □

Note that one obtains Proposition 11 with the trivial information approximation $\bar{t} = \bot_{\sqsubseteq}$, and Proposition 13 by taking the proof to be the approximation, i.e. $\bar{p} = \bar{t}$.

In fact, this proposition can be used for a protocol, which can be seen as a merger of the ideas of proof carrying authorization and the snapshot protocol. The prover p sends a proof $\bar{p}$ to the verifier v. The condition $\bar{p} \preceq \Pi(\bar{p})$ can be checked in the same manner as in the proof carrying authorization protocol. Now v needs to assert the existence of an information approximation $\bar{t}$, with $\bar{p} \preceq \bar{t}$. This can be done by running the asynchronous algorithm until property $\bar{t}_i \preceq \bar{p}_i$ is satisfied locally at each node i, which can be checked in a manner similar to that in the snapshot algorithm for Proposition 13. This protocol is analogous to that in Section 4.1 without the restriction $\bar{p} \preceq \bot_{\sqsubseteq}$, but requiring more work for the verifiers to check the proof (in the worst case they must compute their local fixed point values).

We note finally that the $\sqsubseteq$-continuity property, required of $\preceq$ in our propositions, is satisfied for all interesting trust structures we are aware of: Theorem 3 of Carbone et al. [8] implies that the information continuity condition is satisfied for all interval-constructed structures. Furthermore, their Theorem 1 ensures that interval-constructed structures are complete lattices with respect to $\preceq$ (thus ensuring the existence of $\bot_{\preceq}$). Several natural examples of non-interval domains can also be seen to have the required properties [10]. The requirement that all policies π_p are monotonic also with respect to $\preceq$ is not unrealistic. Intuitively, it amounts to saying that if everyone raises their trust levels in everyone, then policies should not assign lower trust levels to anyone.

5. Lower bounds for proof carrying requests

In the previous section we presented two simple proof carrying request protocols that allow a prover p to convince a verifier v of a property of the global trust state, i.e. a component of the least fixed point corresponding to v's trust in p. We now consider the complexity of such a proof carrying request model.

We will assume the abstract model used for the lower bounds in Section 3, with the following addition. There is a prover p who knows all the policies $f_1, \ldots, f_n$, and some verifier v who wishes to be convinced of f_v^*. Crucially, v does not trust what p says. One thing p can do is to send the entire state $f_1^*, \ldots, f_n^*$ to v (with the implicit claim that this is indeed the lfp), and then v can ask the other nodes to check that this is indeed a fixed point, as in the information approximation protocol. But this only convinces v that $f_1^*, \ldots, f_n^*$ is a component of some fixed point; how is v to know that this is the least fixed point?

Clearly two desirable properties of any such proof carrying protocol are that small proofs can be efficiently generated by the prover, and the verifier can efficiently verify or reject such a proof. Whether such a protocol exists depends on our definitions of 'small' and 'efficient', and in addition whether we allow the verifier to be convinced with some small probability of error. In this section we use the results of Section 3 to show that any proof carrying request must require communication linear in the height of the partial order for the verifier to verify it.

5.1. Nondeterministic communication complexity

To be able to utilise our results from Section 3, we shall need some terminology from nondeterministic communication complexity [15]. Just as NP refers to the languages solvable in nondeterministic polynomial time, there is an analogue in communication problems. For this purpose, we consider communication complexity of polylogarithmic in n to be 'efficient', as opposed to polynomial in the case of computational problems. We can then define a set of analogous complexity classes.

In particular, the class NP^{cc} is the set of functions where, for two players Alice and Bob, Bob can guess Alice's input x and they can then efficiently verify that $f(x, y) = 1$. Similarly, the functions in $\mathrm{co} - \mathrm{NP}^{cc}$ are those where a guess that $f(x, y) = 0$ is efficiently verifiable. The following definition relates nondeterministic complexity with the communication required in a proof and is crucial.

Definition 17 (*Nondeterministic Commmunication Complexity [15]*). The nondeterministic communication complexity $N(f)$ of a boolean function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ is the total amount of communication involved in proving and verifying $f(x, y)$ in the most efficient proof system, where the verifier is to be convinced with certainty.

We shall also make use of the following known result.

Lemma 18 (*[15]*). $DISJ \in \mathrm{co} - \mathrm{NP}^{cc}$ *and* $DISJ \notin \mathrm{NP}^{cc}$.

The lemma implies that there exist proofs of non-disjointness for two sets which require small total communication (p sends to v, say Alice, an element $x \in P \cap Q$ then Alice asks Bob if $x \in Q$), but that there are no small proofs of disjointness. More details about communication complexity can be found in the book [15].

5.2. Proof carrying requests for the least fixed point

Recall that the proof of the lower bound in Section 3 for computing a component of the least fixed point used a reduction from the set-disjointness function. By the above lemmas it immediately follows that for partial orders of height h, any proof of f_v^* must use at least $\Omega(hn \log n)$ bits of communication. Hence the proof itself must be at least this large.

Recall that in the lower bound, the two sets are disjoint iff the least fixed point has $g_0^* = \top$. Since the disjointness function is not in NP^{cc}, proving that some node v has $f_v^* = \top$ is not efficient, but since it is in $\mathrm{co} - \mathrm{NP}^{cc}$ there may exist short proofs that $f_v^* \sqsubseteq \top$.

Recall the approximation theorem of the previous section: if $t \preceq \bot_{\sqsubseteq}$ then there exist proofs of size almost independent (logarithmic) in the height h of the partial order. However, for arbitrary t, the above theorem shows that some proofs must use communication at least linear in h, in the worst case. This appears to suggest an interesting tradeoff between the size of a proof and the height in the partial order of the state being proved. Intuitively, this is because v cannot be efficiently convinced that there are no fixed points smaller than some given one.

Note that, although the original lower bound of Section 3 applies to both randomized and deterministic computation, the proof carrying result only applies to deterministic computation (and the algorithms we presented were also deterministic). It is interesting to ask whether we can produce an efficient randomized proof carrying request protocol.

6. Discussion

We have shown that it is possible to compute local fixed points in the trust structure framework using a totally asynchronous distributed algorithm, and proved its correctness. We also extended the notion of proof carrying requests by giving a theorem that can be parameterized to give various different protocols, each with different restrictions on the range of trust values they can 'prove' and with differing complexities.

Our negative results show that this range of complexities is inherent to any proof carrying request protocol. The original proof carrying protocol presented in [8] requires an amount of communication almost independent of the number of trust levels but can only be used to prove a limited range of trust values. On the other hand, we show that in the worst case, a general proof carrying protocol may require approximately as much communication as computing the trust value 'from scratch.'

The implication of our negative results is that it may be necessary to restrict the policy language in order to efficiently implement the trust model in a distributed network. We have been unable to obtain interesting results in this area but it remains an interesting area for future work.

Finally, we presented the distributed algorithm and the lower bounds in a general model that may allow them to be applied to other problems involving fixed points in a distributed network of policies, for example policy-based routing schemes [19].

Acknowledgments

We thank everyone in the SECURE project consortium for their cooperation; Mogens Nielsen in particular, for the initial developments of 'proof carrying authorization'. Parts of this paper appeared in a preliminary form in the 25th International Conference of Distributed Computing Systems (ICDCS). We also wish to thank the referees for their many helpful comments.

References

[1] M. Blaze, J. Feigenbaum, J. Ioannidis, A.D. Keromytis, The role of trust management in distributed systems security, in: J. Vitek, C.D. Jensen (Eds.), Secure Internet Programming: Security Issues for Mobile and Distributed Objects, in: Lecture Notes in Computer Science, vol. 1603, Springer, 1999, pp. 185–210.

[2] M. Blaze, J. Feigenbaum, J. Lacy, Decentralized trust management, in: Proceedings from the 17th Symposium on Security and Privacy, IEEE Computer Society Press, 1996, pp. 164–173.

[3] A. Chander, D. Dean, J. Mitchell, Reconstructing trust management, Journal of Computer Security 12 (1) (2004) 131–164.
[4] N. Li, J. Mitchell, A role-based trust-management framework, DISCEX III, in: Proceedings from DARPA Information Survivability Conference and Exposition, vol. 1, IEEE Computer Society Press, 2003, pp. 201–213.
[5] N. Li, J. Mitchell, Understanding SPKI/SDSI using first-order logic, in: Proceedings from the 16th IEEE Computer Security Foundations Workshop, CSFW'03, IEEE Computer Society Press, 2003, pp. 89–103.
[6] S. Weeks, Understanding trust management systems, in: Proceedings from the 2001 IEEE Symposium on Security and Privacy, IEEE Computer Society Press, 2001, pp. 94–106.
[7] A. Jøsang, R. Ismail, C. Boyd, A survey of trust and reputation for online service provision, Decision Support Systems (2004) (in press). Preprint available online: http://security.dstc.edu.au/staff/ajosang.
[8] M. Carbone, M. Nielsen, V. Sassone, A formal model for trust in dynamic networks, in: Proceedings from Software Engineering and Formal Methods, SEFM'03, IEEE Computer Society Press, 2003, p. 54.
[9] M. Nielsen, K. Krukow, Towards a formal notion of trust, in: Proceedings from the 5th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming, PPDP'03, ACM Press, 2003, pp. 4–7.
[10] K. Krukow, On foundations for dynamic trust management, Unpublished Ph.D. Progress Report, 2004. Available online: http://www.brics.dk/~krukow.
[11] M. Nielsen, K. Krukow, On the formal modelling of trust in reputation-based systems, in: J. Karhumäki, H. Maurer, G. Paun, G. Rozenberg (Eds.), Theory Is Forever: Essays Dedicated to Arto Salomaa, in: Lecture Notes in Computer Science, vol. 3113, Springer Verlag, 2004, pp. 192–204.
[12] G. Winskel, Formal Semantics of Programming Languages: An Introduction, Foundations of Computing, The MIT Press, Massachusetts Institute of Technology, Cambridge, MA, 1993.
[13] D.P. Bertsekas, J.N. Tsitsiklis, Parallel and Distributed Computation: Numerical Methods, Prentice-Hall International Editions, Prentice-Hall, Inc., 1989.
[14] K. Krukow, A. Twigg, Distributed approximation of fixed-points in trust structures, Tech. Rep. RS-05-6, BRICS, University of Aarhus. (Jan. 2005) Available online: http://www.brics.dk/RS/05/6.
[15] E. Kushilevitz, N. Nisan, Communication Complexity, Cambridge University Press, New York, NY, USA, 1997.
[16] M.R. Garey, D.S. Johnson, Computers and Intractability; A Guide to the Theory of NP-Completeness, W.H. Freeman & Co., New York, NY, USA, 1990.
[17] E. McCluskey, Minimisation of boolean functions, Bell System Technical Journal 35 (1956) Tech. Rep..
[18] V. Cahill, E. Gray, et al., Using trust for secure collaboration in uncertain environments, IEEE Pervasive Computing 2 (3) (2003) 52–61.
[19] T.G. Griffin, F.B. Shepherd, G. Wilfong, The stable paths problem and interdomain routing, IEEE/ACM Transactions on Network 10 (2) (2002) 232–243.