An Alloy Verification Model for Consensus-Based Auction Protocols

Saber Mirzaei
smirzaei@cs.bu.edu
Computer Science Department
Boston University, MA

Flavio Esposito
fesposito@exegy.com
Exegy Inc.
St. Louis, MO

***Abstract*— Max Consensus-based Auction (MCA) protocols are an elegant approach to establish conflict-free distributed allocations in a wide range of network utility maximization problems. A set of agents independently bid on a set of items, and exchange their bids with their first hop-neighbors for a distributed (max-consensus) winner determination. MCA protocols have been proposed, $e.g.$, to solve the task allocation problem for a fleet of unmanned aerial vehicles, in smart grids, or in distributed virtual network management applications. Misconfigured or malicious agents participating in a MCA or an incorrect combination of policy instantiations can lead to oscillations of the protocol, causing, $e.g.$, Service Level Agreement (SLA) violations.**

In this paper we propose a formal, machine-readable, Max-Consensus Auction model encoded in the Alloy lightweight modeling language. The model consists of a network of agents applying the MCA mechanisms instantiated with potentially different policies, and a set of predicates to analyze its convergence properties. We were able to verify that even when all agents follow the protocol, MCA is not resilient against rebidding attacks, and that the protocol fails (to achieve a conflict-free resource allocation) for some specific combinations of policies. Our model can be used to verify, with a "push-button" analysis, the convergence of the MCA mechanism to a conflict-free allocation under a wide range of policy instantiations.

I. Introduction

Resource allocation problems are ubiquitous in distributed systems. The Max Consensus-based Auction (MCA) protocol is a recent approach that allows a set of communicating agents to rapidly obtain a conflict-free (distributed) allocation of a set of items, given a common network utility maximization goal. Without calling it MCA, recent work [8], [10] demonstrated how max-cosnensus auction protocols provide desirable performance guarantees with respect to the optimal network utility. The MCA protocol consists of two mechanisms: a bidding mechanism, where agents independently bid on a single or on multiple items, and an agreement (or consensus) mechanism, where agents exchange their bids for a distributed winner determination.

The use of MCA protocols was proposed to solve resource allocation problems across several disciplines. To our knowledge, its first use appeared to solve the distributed task assignment problem [8], where a fleet of unmanned aerial vehicles bid to assign a set of tasks (geo-locations to be covered.) MCA protocols were also proposed for distributed virtual network management applications [10], where federated infrastructure providers bid to host virtual nodes and virtual links on their physical network, in attempt to embed a wide-area cloud service. More recently, MCA protocols have been also proposed to solve the economic dispatch problem in a distributed fashion, $i.e.$, the problem of allocating power generation tasks among available units in a smart-grid [5].

Each (invariant) mechanism of the MCA protocol may be instantiated with different policies. An MCA policy is a variant aspect of the bidding or the agreement mechanism, and represents an high-level application goal. Examples of policies for the bidding mechanism are the (private) utility function used to generate bids, or the number of items on which agents simultaneously bid on, in each auction round. Note that MCA does not require a centralized auctioneer [8], [10]. [1]

Earlier work on protocols verification established how certain combinations of policy instantiations may lead to incorrect behaviors of a protocol [3], [26]. Similarly, in this paper we analyze the convergence properties of the MCA protocol under various settings using a lightweight, machine-readable, Alloy [16] verification model. Our aim is to show how certain combinations of MCA policies, obtained by design, resulting from misconfigured or malicious agents, may break the convergence of the MCA protocol causing the application to fail and inducing $e.g.$, Service Level Agreement (SLA) violations, energy inefficiencies, or the loss of expensive unmanned vehicles (whose software may fail under an MCA instability.) By MCA convergence, we mean the attainment of a distributed conflict-free assignment of the items on auction.

In particular, we present the following contributions: (i) we identify the common mechanisms of several existing max-consensus auction protocols, renaming MCA such unifying set of mechanisms, and we separate them from their policies in our Alloy model. We then verify the impact of some of the policy combinations on correctness of the protocol. (ii) We describe the Max-Consensus Auction mechanism, and some applications to motivate its versatility in Section II. As a case study, we dissect one particular application: the distributed virtual network mapping problem (defined in Section II), $i.e.$, the NP-Hard problem of assigning, or mapping, constrained virtual nodes and virtual links (items) to physical nodes (agents) and loop-free physical paths, belonging to multiple

[1] Variation of policies may induce different behavior. For example, second price auctions on a *single* item are known to have the strong property of being truthful in dominant strategies, $i.e.$, the auction maximizes the revenue of the bidders who do not have incentives to lie about their true (utility) valuation of the item. In the MCA settings however, truthful strategies may not work as there is uncertainty on whether more items are to be assigned in the future; bidders may have incentives to preserve resources for stronger future bids.

federated infrastructure providers. (iii) In Section III we overview the basic concepts of the Alloy Modeling Language and the Alloy Analyzer in context with our model, described in Section IV, and available at [1]. The model consists of a network of agents together with the set of rules used to asynchronously resolve conflicts, and a set of predicates to analyze the convergence property, when agents are instantiated with different MCA policies. (iv) In Section V we present the analysis of the convergence properties of the MCA protocol. In particular, we present a set of counter-examples to show how the MCA protocol fails to reach a conflict-free assignment of the items on auction, for a particular combination of policy instantiations (Section V.) We finally discuss some relevant work in Section VI and conclude our paper in Section VII.

Once released, both our static and dynamic models will serve as a baseline tool for a deeper investigation of the MCA convergence properties, when the bidding agents are instantiated with possibly conflicting policies.

II. The Max-Consensus Auction Protocol

In this section, we first introduce the Max Consensus-based Auction mechanisms, and then we describe few motivating applications on which such a protocol may be, or was already applied, with particular attention to the virtual network mapping application, that we use as a case study for the rest of the paper.

A. The Mechanisms

Consider a set $\mathcal{I}$ of independent agents (or nodes), that need to allocate in a distributed fashion a set $\mathcal{J}$ of items. Each agent is associated with a private utility $\mathbf{u}_i \in \mathbb{R}_+^{|\mathcal{J}|}$, that represents the benefit (or cost) of hosting an element of $\mathcal{J}$. As in [8] and [10], we assume that agents cooperate to reach a Pareto optimal solution: $\sum_{i \in \mathcal{I}} u_i$. A Max-Consensus Auction protocol consists of two independent mechanisms: (i) a bidding mechanism, where agents independently bid on the items in $\mathcal{J}$, and (ii) an agreement mechanism, where bids are exchanged with the logical neighbors for a distributed winner determination. In particular, an asynchronous agreement is sought on the *maximum* bid on each item to be assigned.

During the bidding phase, using their (private) utility function $\mathbf{u}$, each agent independently assigns bid values on a subset of $\mathcal{J}$. Each agent constructs a vector $\mathbf{b}$, where $\mathbf{b}_{ij}$ is the bid of agent i on item j. The utility function $\mathbf{u}_i$, used to generate the bids, may depend also on previous bids. Agents have a limited budget, $i.e.$ the capacity of physical node to host virtual nodes, and their bids on current items depend on how many items they have won in the past. Formally, the max-consensus on a set of items is defined as follows [19]:

Definition 1: (*max-consensus*). Given a network of agents G, composed by a set of agents $\mathcal{I}$, an initial bid vector of nodes $\mathbf{b}(0) \triangleq (\mathbf{b}_1(0), \ldots, \mathbf{b}_{|\mathcal{I}|}(0))^{|\mathcal{J}|}$ [2], a set of neighbors $\mathcal{N}_i \; \forall i \in \mathcal{I}$, and the consensus algorithm for the communication instance $t+1$:

$$\mathbf{b}_i(t+1) = \max_{j \in \mathcal{N}_i \cup \{i\}} \{\mathbf{b}_j(t)\} \;\; \forall i \in \mathcal{I}, \tag{1}$$

[2]The notation implies that each vector $\mathbf{b}_i(0)$ has size $|\mathcal{J}|$.

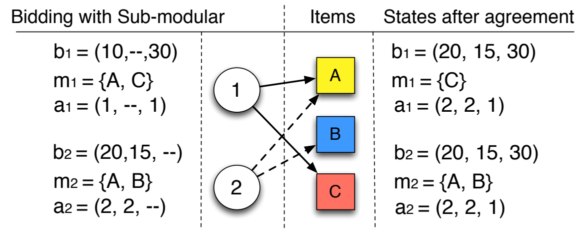

Fig. 1. Two agents (1,2) independently bid on three items (A,B,C), and exchange their bid and allocation vectors applying a distributed max-consensus auction protocol.

Max-consensus on the bids among the agents is said to be achieved with convergence time τ, if $\exists \; \tau \in \mathbb{N}$ such that $\forall \; t \geq \tau$ and $\forall \; i, i' \in \mathcal{I}$,

$$\mathbf{b}_i(t) = \mathbf{b}_{i'}(t) = \max\{\mathbf{b}_1(0), \ldots, \mathbf{b}_{|\mathcal{I}|}(0)\}, \tag{2}$$

where $\max\{\cdot\}$ is the component-wise maximum.

During the bidding phase, agents also save the identity of the items in a bundle vector $\mathbf{m}_i \in \mathcal{J}^{T_i}$, where T_i is the target number of items that can be assigned to agent i, and a vector of time stamps $\mathbf{t}$, $i.e.$, the time at which the bid was generated. The bid generation time-stamps are used in the agreement phase to resolve assignment conflicts in an asynchronous fashion; when transmitted among agents, bids can in fact arrive out of order. After the bidding phase, each physical node exchanges the bids with its neighbors, updating an assignment vector $\mathbf{a}_i \in \mathcal{I}^{|\mathcal{J}|}$ with the latest information on the current assignment of all items.

***Example** 1: Consider Figure 1: agents 1 and 2 independently bid on three items (A, B, C). Agent 1 assigns a value of 10 on item A, and a value of 30 on item C, $b_1 = (10, -, 30)$; then agent 1 stores the identity of items A and C in its bundle vector, $m_1 = (A, C)$, and assigns itself as a winner for both items, $i.e.$, sets the allocation vector a_1 with its own identifier for items A and C. The bidding phase of agent 2 is similar. After bidding, the agents exchange their bids and allocation vectors. Agent 1 learns that there is a higher bid for item A, and stores such higher bid in its bid vector, and the identity of the overbidding agent 2 in its allocation vector (see Figure 1, right column). The protocol has reached consensus. An additional agent 3, connected to agent 1 but not agent 2, would receive the maximum bid so far on each item, as well as the latest allocation vector* $\mathbf{a} = \mathbf{a}_1 = \mathbf{a}_2$.

In Example (1), an agreement is found after the first bidding phase. In general, for more elaborate (non-fully connected) networks of agents, the mechanism iterates over multiple node bidding and agreement (consensus) phases. Note how a successful distributed allocation needs to be conflict-free, $i.e.$, the protocol can only assign each item to a single agent, while agents may win multiple items.

***Remark** 1:* **(no-rebidding allowed on lost items).** A necessary condition to reach an agreement in an MCA protocol is that agents do not bid again on items on which they were overbid in previous auction rounds.

***Remark** 2:* **(rebidding is convenient on items subsequent to an outbid.)** Note how bids generated subsequently to an outbid item are outdated, because they were generated assuming a lower budget. Hence, agents may rebid assigning a higher utility to the items already in their bundle, but subsequent to an outbid item.

Remark 3: **(sub-modularity of the bidding function).** Assigning a set of items to a set of agents is equivalent to a Set Packing Problem, which is NP-hard [22]. Earlier work [8], [10] has shown that if agents generate their bids using a sub-modular function $\mathbf{u}$, then the network utility cannot be arbitrarily low. In particular, in [10] it has been shown that the allocation resulting from the MCA protocol has an approximation ratio of $(1 - \frac{1}{e})$ with respect to the optimal network utility $\sum_i \mathbf{u}_i$.

In the context of the MCA protocol, a sub-modular function is defined as follows:

Definition 2: (*sub-modular function.*) The marginal utility function $u(j, \mathbf{m})$ obtained by adding an item j to an existing bundle $\mathbf{m}$, is sub-modular if and only if

$$u(j, \mathbf{m}') \geq u(j, \mathbf{m}) \; \forall \, \mathbf{m}' \subset \mathbf{m}. \quad (3)$$

If an agent uses a sub-modular utility function, a value of a particular item j cannot increase because of the presence of other items in the bundle $\mathbf{m_i}$ (the data structure keeping track of the items currently assigned to bidder i). This implies that *bids on subsequent items cannot increase*. An example of sub-modular utility function is the residual capacity of a physical node bidding to host virtual nodes; the residual (CPU) capacity can in fact only decrease as virtual nodes to be supported are added to the bundle vector $\mathbf{m}$.

B. MCA Case Study: the Virtual Network Mapping Problem

MCA protocols have been used across a wide range of networked applications, (see, *e.g.*, [5], [8], [10]). In this subsection, we define the virtual network mapping problem, the application that we have chosen as a case study for our MCA Alloy model.

Given a virtual network (VN) $H = (V_H, E_H, C_H)$ and a physical network $G = (V_G, E_G, C_G)$, where V_H and V_G are the sets of virtual and physical nodes, respectively, and E_H and E_G are the set of virtual and physical links, respectively. Each node or link e (physical or virtual) is associated with a capacity constraint $C(e)$. [3] The virtual network mapping is the problem of finding *at least* a mapping of H onto a subset of G, such that each virtual node is mapped onto exactly one physical node, and each virtual link is mapped onto at least one loop-free physical path p while maximizing some utility or minimizing a cost function. Formally, the mapping is a function $\mathcal{M} : H \rightarrow (V_G, \mathcal{P})$ where $\mathcal{P}$ denotes the set of all loop-free paths in G. $\mathcal{M}$ is called a *valid mapping* if all constraints of H are satisfied, and for every $l^H = (s^H, r^H) \in E_H$, exists at least one physical loop-free path $p : (s^G, \ldots, r^G) \in \mathcal{P}$ where s^H is mapped to s^G and r^H is mapped to r^G. The MCA protocol is not necessarily applied to virtual links, as physical nodes (infrastructure provider processes) can merely bid to host virtual nodes, and later run k-shortest path to map the virtual links.

Remark 4: In the rest of the paper, our notation refers to the VN mapping problem, but our verification results are independent from the application running on top of the MCA protocol. Adapting to other MCA applications merely requires a change of variable names.

III. Alloy Overview

In this section we describe how the Alloy [16] language and analyzer work, and we give few examples of primitives that we used to model the MCA protocol.

What is Alloy and how does it work? The term "Alloy" refers to both a formal language and an automated analyzer. The *Alloy Analyzer* translates the user models into SATs, *i.e.*, boolean satisfiability problems. The Alloy language is a declarative specification language for modeling complex structures and behaviors in a system. Alloy is based on first order logic, and is designed for model *enumeration*. The Alloy language is is based on the notions of relations and sets. For example, a physical link can be modeled with a relation between two members of a physical node set. A relation is a particular set whose members are tuples with a specific arity.

To verify the satisfiability of the SAT representing the model, the Alloy Analyzer uses a constraint solver [24]. Checking satisfiability of a large SAT instance may be intractable (or in general, time consuming); In general, checking the satisfiability of the translated SAT instance is exponential (unless $P = NP$) [9], however, the scope of the Alloy Analyzer can be customized and limited to ensure termination of the checking process in a timely fashion.

How can we build a model using the Alloy language? The Alloy language is *lightweight*, and shares standard features and elements with most programming languages, *e.g.*, modules and functions; some features have been instead introduced by Alloy, such as the concepts of *signature* or *scope* [16]. A *signature* declaration is denoted by the keyword `sig`, and models the sets of elements of the system. For example when the MCA protocol is applied to solve a virtual network mapping problem, a basic signature for a physical node with a given hosting CPU capacity and some capacitated connections with other physical nodes can be modeled as follows:

```
sig pnode{
    pcp: one Int, // Physical Cpu Capacity
    id: one Int, // ID of pnode
    pconnections: Int -> pnode // set of Connections
}
```

Signatures may contain some properties that model relations between elements. For example, the signature `pnode` has three relations, two binary (`pcp` and `id`), as they relate two signatures, and one ternary relation. The Alloy language also allows us to express constraints on sets and relations. Such constraints are defined with *constraint paragraphs*, labeled by the keyword `fun`, as in function, *i.e.*, a reusable expression that always outputs a relation, `pred`, as in predicate, whose output is always a boolean, and `fact` *i.e.*, a constraint valid for any instance of the model.

For example, to impose the constraint of non negative physical links capacity, in our model we define a fact `positiveCap` as follows:

[3] Each $C(e)$ could be a vector $(C_1(e), \ldots, C_\gamma(e))$ containing different types of constraints, *e.g.* physical geo-location, delay, or jitter.

```
fact positiveCap{
    all n:pnode | (n.pconnections).pnode >= 0
}
```

where the operator "." is the inner join in relational algebra.

To check that the model satisfies specific properties, the Alloy language supports *assertions*. Assertions are labeled with the keyword `assert`. For example, to assert that two disjoint agents (physical nodes) `n1` and `n2` have non equivalent identifiers, we use the following assertion:

```
assert uniqueID{
    all disj n1, n2: pnode | n1.id != n2.id
}
```

Note that assertions are not enforced rules, but merely properties that we are interested in verifying.

The Alloy language also supports *commands*, *i.e.*, calls to the Alloy Analyzer. For example, to verify whether an assertion holds on a previously defined model, within a user-defined scope, we use the command `check`. The command `run` instead instructs the Alloy Analyzer to find a satisfying instance of the SAT of the model. To check if the assertion `uniqueID` holds in all instances of a model scope containing up to three physical nodes, we run the following Alloy command: `check uniqueID for 3`.

IV. The Alloy Model for Consensus-based Virtual Network Mapping

In this section we overview our MCA Alloy model, applied to the virtual network mapping problem (defined in Section II-B.) Our Alloy code is logically divided into a static and a dynamic sub-model: the static sub-model refers to the underlying hosting physical network, and the virtual nodes to be mapped with the max-consensus based auction protocol, while the dynamic sub-model captures the state transitions.

Static Model. A simplified version of the signature `pnode` was explained in Section III. We extend this signature to include some biding policies and other ternary relations:

```
sig pnode{
    pcp: one Int,
    pid: one Int,
    initBids: vnode->Int,
    initBidTimes: vnode->Int,
    pconnections: some pnode,
    p_T: one Int,
    p_u: one utility,
    p_RO: one release_outbid
    // add your policy here
}
```

In this signature, `initBids` models the initial values that an agent (physical node) assigns when bidding on a subset of items (a virtual node is modeled as `vnode`), and the ternary relation `initBidTimes` models the corresponding bidding time on the virtual nodes, used for the MCA asynchronous conflict resolution mechanism. The bidding policies are modeled using the binary relations *e.g.*, `p_T`, `p_u`, and `p_RO`. `p_T` models the target capacity of an agent (`pnode`) imposing a limit on the number of items (`vnodes`) that an agent can bid on. `p_u` models the utility function, that can be sub-modular or not. [4]

[4]In the definition of relation `pcp`, the keyword `one` refers to each physical node being in relation with exactly `one` integer, *e.g.*, the physical capacity. Similarly, the relation `pconnections` models the fact that in a network, each node is connected to `some` other nodes.

The virtual network mapping problem maps constrained virtual networks on a constrained physical network, eventually owned by multiple, federated infrastructure providers. Our model can be extended to capture any constraints in the form of an Alloy `fact`. As a representative example, we show in this paper how to model the `fact` that physical nodes can bid on virtual nodes only if they have enough physical capacity to host them:

```
fact pcapacity{
     all p: pnode | (sum vnode.(p.initBids) ) <= p.pcp
}
```

In Alloy relations are modeled by ordered tuples; this means that unordered relations must be explicit, *e.g*, our `pconnectivity` fact shows how an undirected link has to be modeled using two (directed) relations:

```
fact pconnectivity{
   all disj pn1,pn2:pnode | (pn1.pid != pn2.pid) and
   (pn1 in pn2.pconnections <=> pn2 in pn1.pconnections)
 }
```

Our static model includes several other facts that regulate basic networking properties. The full Allow model code can be downloaded at [1].

Dynamic Model. The dynamic behavior of the network is modeled as a *transition system*, and the sequence of state changes is regulated by the MCA protocol. Network states are captured in our model using the following signature:

```
sig netState {
    bidVectors: some bidVector,
    time: one Int,
    buffMsgs: set message }
```

The state of the physical network is updated as bid messages are exchanged among agents (or physical nodes.) The `bidVectors` relation contains the current view of each agent, *i.e.*, the vectors $\mathbf{a}$, $\mathbf{b}$, $\mathbf{t}$, and $\mathbf{m}$ (defined in Section II-A and depicted in Figure 1.) The relation `time` models the time generation of each state, while the set of unprocessed messages is modeled with the `buffMsgs` relation. This relation captures the correspondence between states and the buffer of messages in transit. The signature message is modeled as follows:

```
sig message{
    msgSender: one pnode,
    msgReceiver: one pnode,
    msgWinners: vnode->(pnode + NULL),
    msgBids: vnode->Int,
    msgBidTimes: vnode->Int
}
```

Aside from defining the sender and the receiver physical node, the bid `message` signature contains: the view of the sender about the maximum bid known so far on every virtual node (`msgBids`), their winners (`msgWinners`), and the time at which the highest bids were generated (`msgBidTimes`.) Note how, when a message is being processed, these relations are used to update the states $\mathbf{a}$, $\mathbf{b}$, $\mathbf{t}$, and the bundle vector $\mathbf{m}$ for each physical node.

The core of the MCA protocol is modeled by some constraint paragraphs. In particular, the Alloy fact `stateTransition` models the sequence of message processing, and the transitions from state s to s':

```
fact stateTransition{
     all s: netState, s': s.next | one m:message |
     messageProcessing[s, s', m]
}
```

Using the built-in library `ordering`, we can model the states of the system as an ordered sequence which keyword `next` in `s.next` represents the state subsequent to `s` in the transition.

Abstractions Efficiency. The model we have presented so far, contains integer variables and ternary relations; see *e.g.*, the three signatures `pnode`, `bidVector` and `message`. Ternary relations and integers were introduced for the sake of explaining our model, but lead to inefficiencies of the Alloy Analyzer. Using such elements, our model containing the conflict resolution table of the asynchronous MCA protocol generated over $259K$ SAT clauses, for a scope as limited as 3 physical nodes and 2 virtual nodes.

We obtained a more efficient model by (i) replacing each ternary relation with two binary relations, and by (ii) defining our own *values* —combinations of signatures and facts— instead of using integers —predefined and more complex abstractions in Alloy. As an example of signature introduced to reduce the complexity of the ternary abstractions, we show `bidTriple`:

```
sig bidTriple{
    bid_v: one vnode,
    bid_b: one Int,
    bid_t: one Int,
    bid_w: one (pnode + NULL)
}
```

To avoid using the Alloy's predefined integers (signature `Int`) we model natural numbers with the signature `value`:

```
sig value{
    succ: set value,
    pre: set value
}
```

Each instance of the signature `value` only models relations between numbers. Using the two relations `succ` and `pre` we model binary operators $<$, $\leq$, $>$ and $\geq$, respectively, using the binary predicates `valL[,]`, `valLE[,]`, `valG[,]` and `valGE[,]`. For two instances `v1` and `v2` of the signature `value`, we model the inequality `v1` $\leq$ `v2` with the predicate `valLE[v1, v2]` (which in our Alloy model is equivalent to `v1 in v2.pre`).

Using these more efficient abstractions, for the same scope, we were able to reduce the number of SAT clauses from circa $259K$ to circa $190K$, reducing the running time of our consensus assertion from circa a day to less than two hours. [5]

V. Using Alloy to Analyze MCA Convergence

Our model enables the study of the convergence properties of the MCA protocol. In this section we first introduce the assertion for checking such convergence property, and then we show how specific combinations of the MCA policy instantiations may or may not lead to convergence.

Checking the convergence property in Alloy means checking whether or not the consensus assertion holds. All the agents (physical nodes) need to reach an agreement on (i) the assignment vector, containing the identity of the winner agents (virtual nodes), (ii) and the bid vector. The assertion consensus is coded in Alloy as follows [6]:

```
assert consensus{
   (#(netState) >= val) implies consensusPred[]
}
pred consensusPred{
   some s: netState | all disj bv1,bv2: s.bidVectors |
    (
        (bv1.winners    = bv2.winners) and
        (bv1.winnerBids = bv2.winnerBids)
    )
}
```

Utility	Iteration 1	Iteration 2	Iteration 3
Sub-modular	$b_1=\{20,10\}, m_1=\{A,C\}$ $b_2=\{20,10\}, m_2=\{C,A\}$	Agreement $b_1=\{20\}, m_1=\{A\}$ $b_2=\{20\}, m_2=\{C\}$	
Non Sub-modular	$b_1=\{10,30\}, m_1=\{A,C\}$ $b_2=\{10,20\}, m_2=\{C,A\}$	Both PNs outbid on first VN $b_1=\{\}, m_1=\{\}$ $b_2=\{\}, m_2=\{\}$	identical to iteration 1 $b_1=\{10,30\}, m_1=\{A,C\}$ $b_2=\{10,20\}, m_2=\{C,A\}$

Fig. 2. The policy of releasing outbid items, combined with the non sub-modularity policy lead to instability of the MCA: with non sub-modular utility, after the first round both agents have been outbid on the first item, and their bids on the second item have been invalidated. Even bids subsequent to an outbid (see Remark 2) are released seeking a Pareto optimality.

From the consensus literature [19], and from previous studies on the MCA [8], [10], we know that the number of messages required to reach consensus is upper bounded by $D \cdot |V_H|$ where $|V_H|$ is the size of the item set, and D is the diameter of the network of agents. Intuitively, this is because the maximum bid for each item, only has to traverse the network of agents once. We use this bound to set our `val` parameter in the consensus assertion: after `val` number of messages is being processed, a max-consensus on the bid has to be achieved.

Result 1: We checked the assertion `consensus` over several scopes, for a key representative combinations of policies. *We found that MCA always reaches consensus, except when the utility function policy* `p_u` *is set to non sub-modular, and the agents release (and rebid) all subsequent items to an outbid item* $i.e.$*, the* `p_RO` *policy is set to true.*

To understand why the MCA protocol fails for this combination of policies, consider the scenario in Figure 2 (first row): the agent's bids do not increase as items are added to the bundle, as bids have been generated using a sub-modular function. After exchanging the bids, item C is won by agent 2, and item A is won by agent 1. When instead MCA uses a non sub-modular function (as in Figure 2 second raw), bids can increase as items are added to the bundle, and releasing items (subsequent to an outbid node to refresh their bids) causes oscillations, and hence the MCA failure to reach a conflict-free assignment.

Result 2: We also tested the consensus property under circumstances of protocol misbehavior or misconfiguration. In particular, we removed from our model the necessary condition discussed in *Remark 1*, allowing physical nodes to re-bid after they were outbid on a virtual node and, as expected, we found instances in which, consensus (a conflict-free assignment) is not reached. A consequence of this sanity-check for our model is that the MCA protocol is not resilient to *rebidding attacks*, $i.e.$, malicious agents

[5] Our experiments were carried out on a Linux machine running Intel core i3 CPU at 1.4GHz and 4 GB of memory.

[6] The model presented in this paper is a simplified version of the full model described at [1].

can perform a denial of service attack by rebidding even on outbid items. [7]

VI. Related Work

Protocol Verification with Alloy. Theorem proving and model checking tools have been widely used to analyze and verify (distributed) algorithms and protocols [4], [13], [26], [23] for a wide range of (networked) applications, as they allow with minimal implementation efforts to verify complicated properties. The attention towards *lightweight* model-finding tools [14], [16], as an alternative to model checking tools [25], [11] has only recently increased, due to the ease of use, and to the automation that they have introduced. We only cite a representative set of references to define our work in context. In [26] and [23], the authors study with an Alloy model Chord [15], a peer-to-peer distributed hash table protocol.

Alloy has been applied to model and study the properties of other protocols as well [3], [7], [21]. In [3] for example, Alloy is used to analyze the properties of the Stable Path Problem (SPP), and to verify sufficient conditions on SPP instances.

Verifying Correctness of Networking Mechanisms. Recent work has been also carried out to verify the correct behavior of many networking mechanisms. For example, there has been interest in verifying that network forwarding rules match the intent specified by the administrator [17], [18], or even in building tools to debug the network forwarding plane in the context of Software-Defined Networks [12].

Approaches that verify the correct behavior of the routing [20] or the forwarding [2], [6] mechanisms have also been investigated. The authors in [20] for example, propose via SAT instances to statically analyze the router configurations of the data plane, to check isolation errors and network disconnections caused by misconfigurations.

Similar to all these approaches, our work also aims to verify the correctness of a network mechanism, but our focus is on the Max Consensus Auction; in particular, on the virtual network mapping, a management application that infrastructure providers use *during* the creation of a virtual network, not after a (virtual) network has been instantiated.

VII. Conclusions

Max Consensus-based Auction protocols are a recent solution that allows a set of communicating agents to obtain a conflict-free (distributed) allocation of a set of items, given a common network utility maximization goal. We extracted the common mechanisms of such protocols, renaming them MCA: a bidding mechanism, where agents independently bid on a single or on multiple items, and an agreement (consensus) mechanism, where agents exchange their bids for a distributed winner determination. Each MCA mechanism can be instantiated with a wide-range of policies that lead to different behaviors and protocol properties.

In this paper, we used the Alloy Language to model the MCA protocol, and verify its convergence properties under a range of different policies. Our MCA model is application agnostic, but we described our result in context of the virtual network mapping problem. With our model, we were able to show how given combination of MCA policies lead to instability (oscillations) $i.e.$, no convergence to a conflict free assignment is guaranteed, and that MCA is not immune to denial of service attacks as *rebidding attack*. Our released Alloy model can be used to verify the correctness of the MCA protocol, for a wide range of policies and applications, or extended to include property-checking features for large instance of the model.

[7] A thorough analysis on how to design and implement solutions to detect or prevent malicious MCA agents is left as an open research question. However, singular malicious user behavior can be isolated by requiring every agent to sign their messages before broadcasting, using a unique ID. By keeping track of the bidding history of their first hop neighborhood, agents could then detect rebidding attacks (condition in Remark 1), ignoring subsequent invalid bid messages.

References

[1] MCA Alloy model code. `http://csr.bu.edu/alloy/`.
[2] Al-Shaer, E. and Al-Haj, S. Flowchecker: Configuration analysis and verification of federated openflow infrastructures. In *Proc. of SafeConfig*, pages 37–44, New York, NY, USA, 2010. ACM.
[3] M. Arye, R. Harrison, R. Wang, P. Zave, and J. Rexford. Toward a lightweight model of BGP safety. *Proc. of WRiPE*, 2011.
[4] K. Bhargavan, D. Obradovic, and C. A. Gunter. Formal verification of standards for distance vector routing protocols. *Journal of the ACM (JACM)*, 49(4):538–576, 2002.
[5] Binetti, G. *et al.* A distributed auction-based algorithm for the nonconvex economic dispatch problem. *Industrial Informatics, IEEE Trans. on*, 10(2):1124–1132, May 2014.
[6] Canini, M. *et al.* A NICE Way to Test Openflow Applications. In *Proc. of NSDI*, pages 10–10, Berkeley, CA, USA, 2012.
[7] C. Chen, P. Grisham, S. Khurshid, and D. Perry. Design and validation of a general security model with the alloy analyzer. In *Proc. of the ACM SIGSOFT*, pages 38–47, 2006.
[8] Choi, Han-Lim *et al.* Consensus-based decentralized auctions for robust task allocation. *IEEE Trans. on Robotics*, Aug 2009.
[9] S. A. Cook. The complexity of theorem-proving procedures. In *Proc. of ACM Theory of comp.*, pages 151–158, 1971.
[10] F. Esposito, D. Di Paola, and I. Matta. On distributed virtual network embedding with guarantees. *ACM/IEEE Transactions on Networking. Accepted (to appear)*, Nov 2014.
[11] Groote, J. F. *et al.* *The formal specification language mCRL2*.
[12] Handigol, Nikhil *et al.* Where is the debugger for my software-defined network? HotSDN '12, pages 55–60, 2012.
[13] K. Havelund and N. Shankar. Experiments in theorem proving and model checking for protocol verification. In *FME'96*, pages 662–681. Springer, 1996.
[14] G. Holzmann. *Spin Model Checker, the: Primer and Reference Manual*. Addison-Wesley Professional, first edition, 2003.
[15] I. Stoica *et al.* Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM01*, pages 27–31.
[16] D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2006.
[17] Kazemian, Kazemian *et al.* Real time network policy checking using header space analysis. In *Proc. of the 10th USENIX*, nsdi'13, pages 99–112, Berkeley, CA, USA, 2013.
[18] Khurshid, Ahmed *et al.* Veriflow: Verifying network-wide invariants in real time. In *Proc. of HotSDN '12*, pages 49–54, NY, USA, 2012.
[19] N. A. Lynch. *Distributed algorithms*. Morgan Kaufmann, 1996.
[20] Mai, Haohui *et al.* Debugging the data plane with anteater. In *Proc. of the ACM SIGCOMM '11*, pages 290–301, NY, USA, 2011. ACM.
[21] S. Mirzaei, S. Bahargam, R. Skowyra, Kfoury, a., and A. Bestavros. Using alloy to formally model and reason about an openflow network switch. *CS Dept., Boston University, Tech. Rep. 2013-007*, 2013.
[22] R.M. Karp. Complexity of Computer Computations. In *Reducibility Among Combinatorial Problems, Miller and Thatcher.*, 1972.
[23] H. Sadeghian, A. Samadi, and H. Haghighi. Formal analysis of pure-join model of chord using alloy. In *ICSESS*, May 2013.
[24] E. Torlak and D. Jackson. Kodkod: A relational model finder. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 632–647. Springer, 2007.
[25] S. Yovine. Kronos: A verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer*, pages 123–133, 1997.
[26] P. Zave. Lightweight verification of network protocols: The case of chord. 158, 2009.