2014-07-15

An alloy verification model for consensus-based auction protocols

Mirzaei, Saber

Computer Science Department, Boston University

http://hdl.handle.net/2144/20812

Boston University
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. The use of MCA protocols was 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 instantiation of policies 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 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 of 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. Recent work [8], [9] demonstrated how MCA 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 [9], 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 bid on the items, or the number of items on which agents simultaneously bid on, in each auction round. 1 Note that MCA does not require a centralized auctioneer [8].

Earlier work [3], [23] on protocols verification established how certain combinations of policy instantiations may lead to incorrect behaviors of a protocol. Similarly, in this paper we analyze the convergence properties of the MCA protocol, under various settings, using a lightweight, machine-readable, Alloy [13] 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. By MCA convergence, we mean the attainement of a distributed conflict-free assignment of the items on auction.

In particular, we present the following contributions: In Section II we describe the Max-Consensus Auction mechanism, and some applications to motivate its versatility. 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 (items) to physical nodes (agents) belonging to multiple federated infrastructure providers. We discuss related work in Section VI and in Section III we overview some basic concepts of the Alloy Modeling Language and the Alloy Analyzer in contest of our model, described in Sections IV. 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. Using our model, available at [1], we present

1Variation 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 [16], 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 our setting 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.
a set of counter examples to show how MCA fails to reach a conflict-free assignment of the items on auction, for a particular combination of policy instantiations (Section V). Our work serves as a baseline for a deeper investigation of the MCA convergence properties when the bidding agents are instantiated with possibly conflicting policies.

II. THE MAX-CONSSENSUS AUCTION PROTOCOL

In this section, we first introduce the max consensus-based auction mechanism, and then we describe few motivating applications on which such 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 Max-Consensus Allocation Mechanism

Consider a set $I$ of independent agents (or nodes), that need to allocate in a distributed fashion a set $J$ of items. Each agent is associated with a private utility $u_i \in \mathbb{R}_+$, that represents the benefit (or cost) of hosting an element of $J$. As in [8], [9] we assume that agents cooperate to reach a Pareto optimality: $\sum_{j \in J} u_j$. A Max-Consensus Auction consists of two independent mechanisms (or phases): (i) a bidding mechanism, where agents independently bid on the items in $J$, 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 $u$, each agent independently assigns values on a disjoint subset of $J$, in a form of a bid. Each agent constructs a vector $b$, where $b_{ij}$ is the bid of agent $i$ on item $j$. The utility function $u_i$, used to generate the bids, may depend also on previous bids. Agents have a limited budget, i.e., physical node capacity to host virtual nodes, and their bids on current items depend on how many item they have won in the past. Formally, the max-consensus on a set of items is defined as follows [17]:

Definition 1: (max-consensus). Given a network of agents $G$ composed by a set of agents $V_G$, an initial bid vector of nodes $b(0) \equiv (b_{1}(0), \ldots, b_{|V_G|}(0))^T$, a set of neighbors $N_i \forall i \in V_G$, and the consensus algorithm for the communication instance $t+1$:

$$b_i(t+1) = \max_{j \in N_i \cup \{i\}} \{b_j(t)\} \quad \forall i \in V_G, \quad (1)$$

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

$$b_i(t) = b_{i'}(t) = \max\{b_1(0), \ldots, b_{|V_G|}(0)\}, \quad (2)$$

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

During the bidding phase, agents also save the identity of the items in a bundle vector $m_i \in J^{|V_i|}$, where $|V_i|$ is the target number of items that can be assigned on agent $i$, and a vector of time stamps $t$, saving 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 $a_i \in I^{\mid J}$ with the latest information on the current assignment of all items, for a distributed auction winner determination.

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 converged to an agreement (consensus) on the maximum bid for each item. 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 $a_i = a_1 = a_2$.

In Example (1), an agreement is found after the first bidding phase. In general, for more elaborate networks formed by multiple agents, the mechanism iterates over multiple node bidding and agreement (consensus) phases asynchronously, that is, the second bidding phase can start even if the first agreement phase is not terminated. Agents act upon messages received at different times during each bidding phase and each consensus phase; therefore, each individual phase is also asynchronous. 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 generated assuming a lower budget that accounted for the outbid item. 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 [19]. Earlier work [8], [9] has shown that if agents generate their bids using a sub-modular function \( u \), then the network utility cannot be arbitrarily low. In particular, in [9] the authors showed that the allocation resulting from the MCA protocol has an approximation ratio of \( (1 - \frac{1}{e}) \) with respect to the optimal network utility \( \sum u_i \).

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

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

\[
u(j, m') \geq u(j, m) \forall m' \subset m.
\]

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 \( m_i \) (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 submodular utility function is the residual capacity of a physical node bidding to host virtual nodes; the residual capacity can only decrease as virtual nodes are added to the bundle vector \( m \).

**B. MCA Case Study: the Virtual Network Mapping Problem**

MCA protocols have been used across a wide range of networked applications, (see, e.g., [8], [9], [5]). 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 \( H = (V_H, E_H, C_H) \) and a physical network \( G = (V_G, E_G, C_G) \), where \( V \) is a set of nodes, \( E \) is a set of links, and each node or link \( e \in V \cup E \) is associated with a capacity constraint \( C(e) \), the virtual network mapping is the problem of finding 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 a loop-free physical path \( p \) while maximizing some utility or minimizing a cost function. The mapping is a function \( M : H \rightarrow (V_G, P) \) where \( P \) denotes the set of all loop-free paths in \( G \). \( M \) is called a valid mapping if all constraints of \( H \) are satisfied, and for every \( i^H = (s^H, r^H) \in E_H \), exists at least one physical loop-free path \( p = (s^G, \ldots, r^G) \in P \) 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 thereby bid host virtual nodes, and later run \( k \)-shortest path to map the virtual links.

**Remark 4:** In the rest of the paper, our notations refers to the virtual network mapping problem, but our verification results are independent from the application running on top of the MCA protocol.

**III. ALLOY OVERVIEW**

In this section we describe how the Alloy [13] 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 (distributed) system. Alloy is based on first order logic, and is designed for model enumeration (also known as model checking). The Alloy language is simple, expressive, and it 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, e.g., a physical link relation has arity two —the two physical nodes.

To verify the satisfiability of the SAT representing the model, the Alloy Analyzer uses a constraint solver [21]. The SAT satisfiability is checked by attempting all possible boolean combinations of assignments, for all variables composing the SAT instance. The size of the SAT instance is determined by the number of sets and by the number and type of relations composing the model. Checking satisfiability of a large SAT instance may be time consuming; however, the scope of the Alloy Analyzer can be 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 are instead unique to Alloy, such as the concepts of signature or scope. A signature declaration is denoted by the keyword sig, and models the sets of elements of the system, or the protocol, under verification. For example, when an MCA protocol is applied to solve a virtual network mapping problem, a basic signature for a bidding physical node \( pnode \) with a given hosting CPU capacity \( cp \) and some capacitated connections with other physical nodes can be modeled as follows:

```alloy
sig pnode{
    pcp: one Int,
    id: one Int,
    pconnections: Int -> pnode
}
```

Signatures can define fields to model relations. For example, the signature \( pnode \) has three relations, two binary (\( pcp \) and \( id \), as they relate two signatures, and one ternary relation. With the Alloy language is possible to express also constraints on sets and relations. Such constraints are defined with constraint paragraphs, labeled by the keywords fact, fun, as in function, and pred, as in predicate. For example, to impose the constraint of non negative physical links capacity, in our model we define a fact \( \text{positiveCap} \) as follows:

```alloy
fact positiveCap{
    all n:pnode | Int.(n.pconnections) >= 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 for every relation that involves two disjoint agents (physical nodes) n1 and n2, such agents have non-equivalent identifiers, we use the following:

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

Note that assertions are not enforces 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 use the command: check uniqueID for 3.

IV. THE ALLOY MODEL FOR CONSENSUS-BASED VIRTUAL NETWORK MAPPING

In this section we overview our MCE Alloy model, applied to the context of the virtual network mapping problem (see 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 other MCA bidding policies, and few other ternary relations. In particular, initBids models the initial values that an agent (physical node) assigns when bidding on a subset of items (virtual nodes), and the ternary relation initBidTimes models the corresponding bidding time on the virtual nodes, used for the MCA asynchronous conflict resolution mechanism. The binary relations p constructions, respectively, the target capacity, used by the MCA protocol to impose specific loads on the bidding physical nodes, the utility function, that can be sub-modular or not (see Remark 3), and the Release Outbid policy, that if set to true, imposes to agents the release of (and rebid on) the outdated bids subsequent to an outbid item (see Remark 2).

```alloy
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
}
```

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 how to model the fact that physical nodes can bid on virtual nodes only if they have enough physical capacity to host them:

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

By definition, an Alloy relation is modeled with an ordered tuple; 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:

```alloy
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, and similar basic facts and signatures for the virtual nodes that we do not report for lack of space. For example, a selfloop fact is required to avoid that the source and the destination identifiers of a (physical or virtual) link are not identical, or, a vnodeFact set imposes that the identifier of the agent is non-negative, and unique.

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. Such network state updates are captured in our model with the following signature:

```alloy
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 physical nodes. The bidVectors relation contains the current view of each agent, i.e., the vectors a, b, t, and m (defined in Section II-A and depicted in Figure 1). The relation time models the simulation time of each network state, while the set of unprocessed messages is modeled with the buffMsgs binary relation. This relation captures the correspondence between states and the buffer of messages in transit. The signature message can be modeled as follows:

```alloy
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 physical node about the maximum bid so far on every known virtual node (msgBids), their winners (msgWinners), and the time at which the highest bids were generated (msgBidTimes). Note how, after a message is being processed, these relations are used to update the states a, b, t, and the bundle vector m for each physical node.
The core of the MCA protocol is modeled by the constraint paragraphs. In particular, the Alloy fact stateTransition models the sequence of message processing, and the transitions from state s to s’:

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

**Abstractions Efficiency.** The model we have so far presented contains integer variables and ternary relations; see the three signatures pnode, bidVector and message. Ternary relations and integers are convenient, as they directly refer to variables used in the protocol or system to model, but lead to inefficiencies of the Alloy Analyzer. The model containing the conflict resolution table of the asynchronous MCA protocol alone generated over 259,000 SAT clauses, for a scope as limited as three physical nodes and two 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:

```alloy
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:

```alloy
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 <, <=, > and >=, respectively, using the binary predicates val1[], val1E[], val1G[] and val1GE[]. For two instances v1 and v2 of the signature value, we model the inequality v1 <= v2 with the predicate valLE[v1, v2] evaluated to true (which in our Alloy model is equivalent to v1 in v2.pre).

Using these more efficient abstractions, for the same limited scope of three physical nodes and two virtual nodes, we were able to reduce the number of SAT clauses from circa 259,000 to circa 190,000, reducing the running time of our consensus assertion from circa a day to less than two hours. 3

V. USING ALLOY TO ANALYZE MCA CONVERGENCE

Our model, available at [1], enables the study of the convergence properties of the MCA protocol. In this section we first introduce the assertion for checking this property,

![Fig. 2. Wrong policy combinations may lead to unstable MCA protocol: Releasing after outbid items combined with sub-modular utility functions lead to convergence of the MCA protocol. Releasing after outbid items combined with non-submodularity lead to instability of the MCA protocol.](image)

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:

```alloy
assert consensus{
  (#(netState) >= val) implies consensusPred()
}
```

```alloy
pred consensusPred{
  some s: netState | all disj bv1,bv2: s.bidVectors |
  (bv1.winners = bv2.winners) and (bv1.winnerBids = bv2.winnerBids)
}
```

Leveraging the consensus literature [17], we know from [9] 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 (e.g., the number of virtual nodes to assign), and $D$ is the diameter of the network of agents (in number of hops). 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 the theoretical upper bound on the number of messages is reached, a max-consensus on the bid has to be achieved, or the MCA protocol does not converge to a conflict-free assignment.

**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,R is set to sub-modular, and the agents release (and rebid) all subsequent items to an outbid, i.e., the p,R0 policy is set to true.

To understand why the MCA protocol fails for this combination of policies, consider the scenario in Figure 2 (first raw): the agent’s bids do not increase as items are added to the bundle, i.e., bids maybe have been obtained 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-submodular 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, hence the MCA failure to reach a conflict-free assignment.

**Result 2:** We also tested the consensus property under

---

3Our experiments were carried out on a Linux machine running Intel core i3 CPU at 1.4GHz and 4 GB of memory. Unfortunately the Alloy Analyzer does not currently support parallelization.
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 can perform a denial of service attack by rebidding even on outbid items.

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], [22], [11], [23], [20] for a wide range of (networked) applications. The attention towards lightweight model-finding tools, as an alternative to model checking tools 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 [23] and [20], the authors study with an Alloy model Chord [12], a well-known peer-to-peer distributed hash table protocol. As in our work, Alloy has been applied to study the convergence of other protocols as well [3], [7]. 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. Substantial work has been also carried out to verify the correct behavior of many networking mechanisms. For example, there has been recent interest in verifying that network forwarding rules match the intent specified by the administrator [14], [15], or even by implementing a network forwarding plane debugger for a Software-Defined Networks [10]. In [10], each flow entry is marked with traces that are then used for verification, as they present a unified view of a packets journey through the network.

Work on verifying the correct behavior of the routing [18] or the forwarding [2], [6] mechanisms has also been carried out. The authors in [18] for example, propose to statically analyze the routers configuration of data-plane to check isolation errors and network connectivities due to misconfiguration via SAT instances.

Similar to all these approaches, our work also aim to verify the correctness of a network mechanism, but we focus on the Max Consensus Auction mechanisms, and in particular on the virtual network embedding, 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 Auctions (MCA) 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. 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) mechanisms, 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. MCA is application agnostic, but we described our result in context of a specific resource allocation application: the virtual network embedding. 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.

REFERENCES