Solver-Aided Multi-Party Configuration

Kevin Dackow
Brown University
kevin_dackow@alumni.brown.edu

Andrew Wagner
Brown University
andrew_wagner1@alumni.brown.edu

Tim Nelson
Brown University
tbn@cs.brown.edu

Shriram Krishnamurthi
Brown University
sk@cs.brown.edu

Theophilus A. Benson
Brown University
tab@cs.brown.edu

ABSTRACT

Configuring a service mesh often involves multiple parties, each of whom is responsible for separate portions of the overall system. This can result in miscommunication, silent and sudden errors, or a failure to meet goals.

We identify two distinct modes of configuration that call for different solutions. We use synthesis algorithms to extract a set of properties—the *envelope*—that each party needs the other to obey. Administrators can use the envelope to aid verification and synthesis or to support fault-localization and negotiation when goals conflict.

This paper introduces the problem, lays out the modes, presents algorithms for to each, and gives a prototype implementation. We use this to show the feasibility of the approach in the microservices access-control domain and raise new research questions.

ACM Reference Format:
Kevin Dackow, Andrew Wagner, Tim Nelson, Shriram Krishnamurthi, and Theophilus A. Benson. 2020. Solver-Aided Multi-Party Configuration. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks (HotNets '20), November 4–6, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3422604.3425944

1 INTRODUCTION

Network administrators have long confronted the problem of generating and analyzing correct configurations, and yet misconfigurations persist. A brief scan of message boards shows administrators struggling [21–24, 30, 48] with seemingly simple tasks, such as configuring reachability and service-to-service access control.

The problem is particularly thorny when *multiple* administrators, with differing goals, expertise, and levels of abstraction, have to collectively configure a system. For instance, Kirsten Newcomer of RedHat says [41]:

> *"[A]ll of these big companies have multiple teams, multiple business units sometimes in those business units. . . . [T]hey need to make it possible for those different teams, with potentially different security requirements or regulatory requirements for their applications, to deploy to a single cluster."*

HotNets '20, November 4–6, 2020, Virtual Event, USA
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-8145-1/20/11. . . $15.00
https://doi.org/10.1145/3422604.3425944

Similarly, in a conversation about the interplay between administrators, Shriram Rajagopalan, a Principal Engineer at an enterprise service mesh company, says:

> *"[C]onflicts are the main problem as they result in a lot of downtime, or inability to identify what the heck is going wrong in the system."*

This paper seeks to add focus and clarity on *collaborative multi-party configuration.* It presents an initial attempt at solving the problem that leverages, but does not fully rely on, synthesis. Concretely, we observe that synthesis algorithms can produce intermediate objects that effectively define the interface that each party expects the others to satisfy. We extract this object, which we call an *envelope*, and use it to ease verification and enable collaborative negotiation for configurations. In the process, we observe that there are multiple *modes* of multi-party configuration, and specifically address both of these poles:

Conformance A central provider's settings override all others' goals, so tenants must work around these inflexible demands (e.g., a cloud provider's global security goals must be met).

Negotiation In a collaborative setting, administrators are open to the possibility of compromise and negotiation can flow in both directions (e.g., the different teams in a company cooperating to configure their service mesh).

In short, we focus on a practically relevant configuration context (which presents a new kind of synthesis problem), identify different modes of operation, design algorithms for them (Sec. 4), and package the algorithms into a working prototype (Sec. 5). Our tool, riffing on "*mu*lti-party" and "Pu*ppet*" (though Puppet does not contain novel synthesis algorithms), is called Muppet. This paper motivates the work in more detail (Sec. 2) and then provides a walkthrough of Muppet's behavior (Sec. 3) before presenting the technical core, then terminates in related work and discussion.

2 MOTIVATION: MICROSERVICES

Microservices present particularly difficult configuration challenges. Due to the widely distributed nature of typical industry-grade microservice architectures, there are often several distinct parties involved in configuring the system. When two or more administrators have overlapping jurisdiction over portions of the system, it can result in conflicting policies and configurations.

As a working context for this paper, assume one team is configuring the container orchestration while another configures the service mesh. Suppose the Istio service mesh is running on Kubernetes (K8s), with one Istio administrator and one K8s administrator.

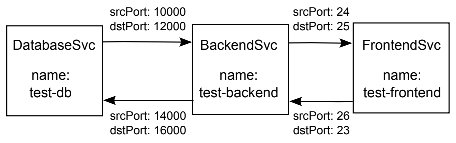

Figure 1: Example microservice architecture.

port	perm	selector
23	DENY	*

Figure 2: K8s goal. Denies traffic for all services on port 23.

srcService	dstService	srcPort	dstPort
test-frontend	test-backend	24	25
test-backend	test-frontend	26	23
test-backend	test-db	14000	16000
test-db	test-backend	10000	12000

Figure 3: Initial Istio goals. These convey the reachability requirements defined in Figure 1.

K8s Configuration. The K8s administrator has configuration authority over pod namespacing, traffic management, scheduling, access control, fault tolerance, service endpoints, and security. A K8s administrator can achieve their goals by permitting or prohibiting traffic with a variety of policy options on domains including ports, IP addresses, and resource labels.

Istio Configuration. The Istio administrator can configure the functionality of the service mesh, access control, telemetry, performance, and some security as well. Like K8s administrators, Istio administrators can also permit or prohibit traffic using policies that operate on ports, IPs, and resource labels to achieve their goals. Confusingly, this means that there is much interaction between the configuration domains of both administrators, despite Istio policies typically focusing on mesh behavior and K8s policies often focusing on structural security.

Configuration Challenges. This overlap creates problems when the two administrators have conflicting goals. Consider that both Istio and K8s administrators can permit or deny traffic based on the port being sent to. If either Istio or K8s denies the traffic it will be denied *even if the other party explicitly allows the traffic.* If human-authored configurations are deployed without careful checking, this can create situations that are challenging to debug: one administrator does not know why their previously-working policies are failing while the other is unaware of the spill-over effects of their own configuration changes. Moreover, existing monolithic synthesis approaches fail to resolve these conflicts, as the union of the two property sets is *unsatisfiable*; some compromise or weakening of goals is necessary to move forward. We will next explore this through a concrete example.

srcService	dstService	srcPort	dstPort
test-frontend	test-backend	$\exists w$	$\exists x$
test-backend	test-frontend	$\exists y$	$\exists z$
test-backend	test-db	14000	16000
test-db	test-backend	10000	12000

Figure 4: The revised Istio goals still convey Fig. 1 reachability, but relax some ports. A $\exists$ indicates that any value is acceptable to the Istio administrator, with the variables capturing which *must* be the same (in this case, none).

3 MUPPET BY EXAMPLE

The conversations that prompted this work involved conflicts that crossed the provider-customer line, and thus it was difficult to obtain concrete configurations and precise data on the nature of the conflicts. The present work thus extrapolates from discussions and our survey of public help posts, which show that many administrators struggle with configuring even reachability. Due to space restrictions, we limit this example to only include settings relevant to reachability in microservices.

Structure. Suppose that traffic on the mesh may be affected by K8s NetworkPolicy and Istio AuthorizationPolicy objects. This illustrative example naturally leaves out a fair bit of functionality; we discuss our modeling choices in Sec. 5.

The service mesh itself consists of three services — a frontend, a backend, and a database — that communicate over the ports shown in Fig. 1. Muppet consumes the YAML files that K8s and Istio administrators use in production to model the system structure.

Goals. We assume that each administrator has their own independent set of goals. Concretely, suppose the K8s administrator learns about Telnet vulnerabilities common to port 23 and decides to globally ban communication over the port. Meanwhile, however, the Istio admin has a working service mesh whose frontend has been configured to receive traffic from the backend via port 23.

Administrators specify these goals as CSV files. We show them graphically in Fig. 2 (K8s) and Fig. 3 (Istio). For simplicity, we have assumed that the K8s admin has only this one (new) goal.

Conflict. The conflict manifests when the K8s administrator pushes the global ban on port 23. The Istio administrator is running their service mesh, but experiences sudden failures because reachability from the frontend to backend is broken. Particularly frustrating to the Istio administrator is the fact that they had not pushed any recent changes that would impact reachability, which makes identifying the root cause of the problem challenging.

Even more confusingly, the policies for Istio and K8s generally operate at different levels in the stack, which further hinders debugging by obfuscating the problem. This is because Istio administrators are typically concerned with higher-level service-mesh-specific policymaking, while the K8s administrator is more preoccupied with system-wide policies that may touch many Istio meshes.

Suppose we are in conformance mode, where the K8s system may support many independently-operated Istio meshes. Ideally, before pushing the configuration change, the K8s administrator needs

```
all src: Service, dst: Service-dst |
/*1*/ 23 not in dst.active_ports or
/*2*/ 23 in {egress: AuthPolicy | egress.target in src.labels}.deny_to_ports or
/*3*/ (some {egress: AuthPolicy | egress.target in src.labels}.allow_to_ports and
          23 not in {egress: AuthPolicy | egress.target in src.labels}.allow_to_ports) or
/*4*/ src in {ingress: AuthPolicy | ingress.target in dst.labels}.deny_from_service or
/*5*/ (some {ingress: AuthPolicy | ingress.target in dst.labels}.allow_from_service and
          src not in {ingress: AuthPolicy | ingress.target in dst.labels}.allow_from_service)
```

For all source-destination service pairs, either:

(1) The destination service does not listen on port 23.
(2) The source service is explicitly blocked from sending to port 23 by an egress policy.
(3) The source service is implicitly blocked from sending to port 23, since it is explicitly allowed to send to some other port but not to 23.
(4) The destination service is explicitly blocked from receiving from the source service by an ingress policy.
(5) The destination service is implicitly blocked from receiving from the source service, since it is explicitly allowed to receive from some other service but not from the source.

Figure 5: An envelope generated by Muppet written in Alloy [34] syntax after applying elementary simplifications.

to provide information to its tenants. For security or intellectual-property reasons, and also to avoid overwhelming their tenants, they may not wish to publish their entire policy goals; also, parts of the goals may be satisfied entirely internally. Rather, they need to provide tenants with rules that guide the tenants' configuration.

From Synthesis to Envelopes. Traditional approaches to configuration synthesis would configure the two systems independently, which is unhelpful in this context because the problem lies in their interaction. We have therefore designed a modified synthesis algorithm that translates each party's goals into an *envelope* (inspired by the "flight envelope" terminology used for airplanes). We use the notation $E_{K8s \rightarrow Istio}$ to mean the conditions the Istio administrator must satisfy in order to be compatible with the K8s administrators goals. An envelope is represented as a necessary and sufficient *set of predicates*, which can be used in two ways:

- they can be applied to a recipient's *configuration*, or
- they can be compared with the recipient's *goals* (which are also a set of predicates over configurations)

to check for compatibility. Furthermore, the provider's envelope and tenant's goals can be combined to serve as input to synthesize the tenant's configuration.

Assuming we are in conformance mode, where Istio must conform to K8s, we show the $E_{K8s \rightarrow Istio}$ in Fig. 5. It is generated by the algorithm in Sec. 4.

Envelope Usage. Before changing their configuration, the K8s administrator sends $E_{K8s \rightarrow Istio}$ to all their Istio customers. This provides the Istio administrators with the ability to check if their configuration or goals conflict with the K8s goals.

Concretely, this envelope states that for all services: either the service is not exposed on port 23; it is explicitly or implicitly banned by Istio AuthorizationPolicies receiving traffic on port 23; or is explicitly or implicitly banned by AuthorizationPolicies from receiving traffic from services that send via port 23. If, for any service, all of these disjuncts are false, then the Istio administrator will be in conflict with the K8s policy, and will therefore potentially have bugs when the K8s admin pushes their new configuration.

The envelope informs the Istio administrator that their goals must change (or can help identify a reachability problem if the K8s configuration has already deployed). They may realize their goals were too strict: it doesn't matter which port is exposed so long as the frontend is reachable. At the same time, the database may be hard to reconfigure. They can thus provide *partial, flexible* goals, shown in Fig. 4. The existential quantifiers allow the synthesizer to choose up to four different ports that are harmonious with both the Istio goals and the K8s envelope. With the goals satisfiable, Muppet generates a configuration.

Muppet also gives an envelope in the other direction: $E_{Istio \rightarrow K8s}$. In the conformance case, this may not be useful. In the cooperative case, however, this can be used as the medium for fault-localization and negotiation between administrators.

Furthermore, by providing just the formulas that need to be true to ensure functionality, Muppet does not restrict the administrators to the synthesized configurations. Both administrators can configure their system however they want and evaluate the resulting configurations against the envelopes they receive. This lets them handle a variety of technical and human considerations that fall outside the scope of synthesis tools, and which will exist no matter how sophisticated the synthesizers become.

4 ENVELOPE EXTRACTION AND USE

Traditional configuration-synthesis tools tend to embrace a workflow (Fig. 6) where the user provides a set ϕ_A of properties or examples, from which the synthesizer produces a configuration. Some synthesizers, especially of the example-driven variety (e.g. [52]), allow the user to add additional examples or properties (triggering another synthesis step) if the output fails to meet their expectations. Other tools (e.g., NetComplete [15]) permit users to give a *partial* configuration as a starting point, or involve some optimization (e.g., [45]). Our approach allows for either of these notions of partiality; Muppet permits administrators to flag portions of their configuration as "soft" and thus open to automated compromise.

We now show how the existing synthesis workflow can be enriched to aid *multi-party* synthesis. While the approach presented

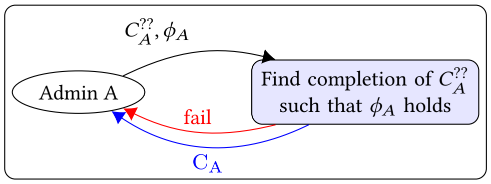

Figure 6: Sketch of property- or example-driven single-party configuration synthesis. $C_A^{??}$ denotes an input configuration (possibly partially defined for autocompletion or given as a target to approximate) and ϕ_A represents formal properties or an example set. Synthesis either produces a complete configuration C_A as output or fails. If C_A is unacceptable, synthesis may be re-run with an augmented input set.

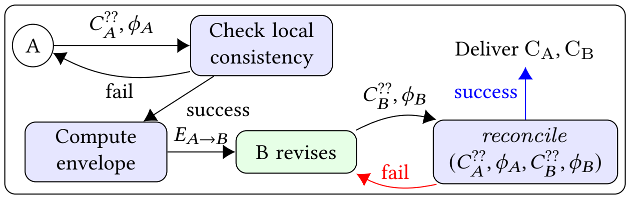

Figure 7: Solver-aided conformance workflow. Here, A might correspond to a service provider and B to a new tenant. Since A provides information only once, the envelope $E_{A\to B}$ need never be recomputed.

here is not domain-specific, we do assume (Alg. 3) that administrator goals can be translated (by the system, not the administrator) to bounded first-order formulas. We also present a version of the model with two (rather than an arbitrary number of) administrators. Neither of these limitations is essential; Sec. 7 discusses both in more detail. In both modes sketched (solver-aided conformance and solver-aided negotiation) all administrators have non-overlapping configuration domains, which may involve different aspects of the network and even different core abstractions.

4.1 Solver-Aided Conformance Model

The solver-aided conformance workflow (Fig. 7) begins with administrator A providing the system with a configuration target ($C_A^{??}$), along with their behavioral goals for the overall system (ϕ_A). Following the lead of Solar-Lezama, et al. [46], we use "??" to indicate a configuration that may be partially specified. This can be thought of as either a configuration with "holes" for autocompletion or a full configuration that labels some settings as "soft". Both varieties of partiality in the configuration can be seen as the leeway that the administrator is willing to grant the system throughout the synthesis process. An empty $C_A^{??}$ would thus indicate complete flexibility on configuration values from A (although not on goals ϕ_A).

The system first performs a local-consistency check to confirm that $C_A^{??}$ does not preclude ϕ_A. Next, the system computes a set of constraints on administrator B's configuration space that are necessary and sufficient to satisfy ϕ_A—we call this constraint-set an *envelope* for B from A (denoted $E_{A\to B}$), which is provided to B to aid (Fig. 8) their configuration task.

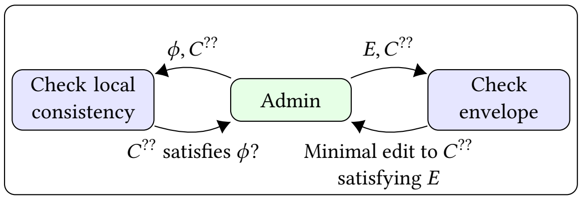

Figure 8: Solver aid provided to an arbitrary administrator *Admin* during their revisions phase after receiving an envelope E. We omit subscripts since the sender is also arbitrary.

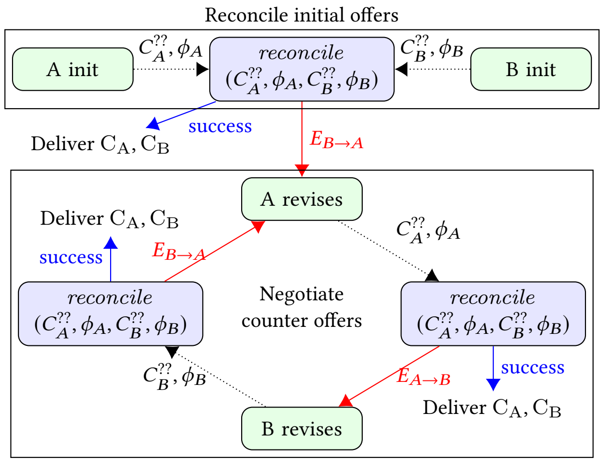

Figure 9: Solver-aided negotiation workflow. After a reconciliation attempt (top) on initial offers, the administrators trade counter-offers via the solver (bottom).

Note that passing $E_{A\to B}$ to B deviates sharply from the traditional single-administrator method (Fig. 6). At this point in the process, B has not yet even stated their goals. Yet, the system is still capable of computing a characterization of A's goals, modulo A's current configuration settings, *strictly in terms of B's domain.* This provides valuable guidance for B's configuration efforts (Sec. 2). The envelope can also be used (Fig. 8) to produce a candidate $C_B^{??}$ that provably satisfies A's goals, which B can then adjust to suit their own goals before submitting both for reconciliation. Should reconciliation succeed, the process ends. If it fails, B has submitted an offer that failed to satisfy either $E_{A\to B}$ or ϕ_B and thus B must make revisions: either by changing their goals or by widening the negotiable region of their partial configuration.

4.2 Solver-Aided Negotiation Model

We now relax the key assumption in Sec. 4.1. Suppose A is now be willing to negotiate over its initial configuration (and perhaps even its goals). This enables a two-way exchange of offers and counter-offers as the solver mediates between the two administrators. Fig. 9 sketches the solver-aided negotiation workflow. Note that the essential datatypes (goals, offers, etc.) and core interactions remain the same as in Sec. 4.1; the only essential change is the generalization

```
fun localConsistency($C^{??}_A, \phi_A$):
    $r \leftarrow \mathrm{sat?}(\exists \mathrm{C_B}, \mathrm{C_A} \supseteq C^{??}_A | \mathrm{C_A} \cup \mathrm{C_B} \vDash \phi_A)$;
    if $\mathrm{unsat}(r)$ then return $\mathrm{fail}(\mathrm{feedback}(r))$;
    return $\mathrm{success}(r.\mathrm{C_A})$;
```

Alg. 1: Checking local consistency of *A*'s offer (symmetric for *B*'s).

```
fun reconcile($C^{??}_A, \phi_A, C^{??}_B, \phi_B$):
    $r \leftarrow \mathrm{sat?}(\exists \mathrm{C_A} \supseteq C^{??}_A,\ \mathrm{C_B} \supseteq C^{??}_B | \mathrm{C_A} \cup \mathrm{C_B} \vDash \phi_A \wedge \phi_B)$;
    if $\mathrm{unsat}(r)$ then return $\mathrm{fail}(\mathrm{feedback}(r))$;
    return $\mathrm{success}(r.\mathrm{C_A}, r.\mathrm{C_B})$;
```

Alg. 2: Offer reconciliation. A set of partial configurations can be reconciled if and only if they can each be extended to total configurations that, together, satisfy the goals of all administrators.

```
fun computeEnvelope($\mathrm{C_A}, \phi_A, B$):
    $\phi'_A \leftarrow decompose(\phi_A)$;
    $E_{A \to B} \leftarrow \varnothing$;
    for $\varphi \in \phi'_A$ do
        if $vars(\varphi) \cap dom(B) \neq \varnothing$ then $E_{A \to B} \xleftarrow{+} subst(\varphi, \mathrm{C_A})$;
    end
    return $E_{A \to B}$;
```

Alg. 3: Computing the envelope for B to satisfy ϕ_A, modulo a fixed configuration $\mathrm{C_A}$. The formulas ϕ_A are decomposed into small subformulas, and those subformulas that do not mention B's configuration domain are filtered out. For those subformulas that do relate to B's domain, any mention of an item from A's domain is substituted with the concrete settings provided by $\mathrm{C_A}$.

to a case where both parties will negotiate. We opted for a round-robin approach—rather than a symmetric one, where all parties receive envelopes simultaneously—to avoid forcing administrators to accommodate a potentially moving target.

Thus, the two differences from Sec. 4.1 are: (1) all parties register their partial configurations and properties in advance; and (2) each administrator gets a turn to revise in a round-robin fashion. After the initial registration, an envelope is sent to an arbitrary participant (represented by *A* in Fig. 9) and the round-robin negotiation begins.

In each revision phase, the pertinent administrator may change either their partial configuration or their goals as needed. As with any negotiation, there will be some situations the solver-mediator can resolve, and others that may require direct communication between administrators. The key here is that the solver mediation helps make administrators aware that such communication is necessary, and envelopes provide a *specific focus* to the discussion.

4.3 Algorithmics

Both local consistency and reconciliation checks involve straightforward queries to a SAT/SMT solver that seek consistent completions. The key difference is that local consistency (Alg. 1) completes one offer and reconciliation (Alg. 2) completes both. Goal tuples given by users (Fig. 2) are translated to logic formulas by substitution using a formalization of network and authorization policy semantics derived from documentation.

In all cases, failure returns with feedback to help the user refine their settings. On target configurations, feedback comes in the form of minimal edits over soft-constrained variables, whereas on configurations with "holes," feedback comes as an unsatisfiable core with blame information.

5 MUPPET

We have built Muppet, a prototype implementation of these ideas. To apply it in the microservices domain, we created a logical model and goal language for service-to-service reachability and port-level security policies. Muppet expands each goal entry to a logical formula over both K8s and Istio configurations that is then used in envelope creation (Alg. 3).

In Muppet, configurations describe two types of components: K8s NetworkPolicies, and Istio AuthorizationPolicies over a common set of Services. We modeled the K8s NetworkPolicy so that K8s administrators can control traffic to and from Services based on service selectors and ports. For AuthorizationPolicies, we modeled the subset relevant to Services, which gives the Istio administrator the ability to allow or deny traffic across services and ports. This scope is sufficient to express reachability problems akin to those seen in our survey of real-world misconfigurations (Sec. 1).

The logical portion of Muppet is built atop the Pardinus [10] target-oriented model finder, which is itself an extension of the Kodkod [50] relational model finder. This provides us with a formula-manipulation library and efficient solver implementation. Although we have not yet run our prototype on very large examples, all queries made in modest scenarios, like the one presented in Sec. 3, finish in under 1 second.

6 RELATED WORK

As has been observed [3, 36], formal methods tools must account for the actual needs of administrators before they can be widely adopted. While this point is often made in the context of verification, the same holds for synthesis. Recent progress in *autocompleting* partial configurations [15], in synthesizing *minimal edits* [45] and in *comparative* synthesis by example [52] is encouraging work in this direction, taking the "human in the loop" into account.

General research on configuration synthesis has thrived, yielding, over the past decades, tools to produce configurations for firewalls (e.g., [12, 13, 17, 42, 54]), routing (e.g., [4, 5, 15]), and other networking targets [14, 38, 44, 47]. Some work, such as Bravetti, et al. [7] also focuses on microservice synthesis. All of these generate a monolithic configuration, sidestepping the multi-party case.

Config2Spec [6] soundly extracts high-level properties from existing configurations. Our system requires properties while also taking the existing configurations into account; we therefore view it as complementary to Config2Spec.

Our work is perhaps closest in spirit to PGA [43], which elegantly allows for verified composition of endpoint policies from different authors. The key difference lies in our focus on enabling negotiation in the conflict case: instead of just reporting unavoidable conflicts to policy authors, Muppet localizes the conflict goals into envelopes that are in terms of each administrator's settings.

A variety of other works (e.g., [2, 16, 18, 35, 37, 53]) have explored distributed constraint solving and distributed notions of configuration. Again, our work differs in its focus on negotiation in the unsatisfiable case. The idea of envelope exchange is most useful, not as part of a fully automated process, but rather as an aid to communication between human administrators.

7 DISCUSSION

Despite the mounting evidence (e.g., [41]) that multi-party configuration is a challenging problem, to our knowledge this work is the first to support solver-aided *collaborative* multi-party configuration. We see use for solver-aided collaboration in many settings: any enterprise with multiple operator teams (e.g., security ops vs. dev ops); academic settings (campus vs. research teams), provider-tenant relationships in the cloud, etc. Our prototype shows that the basic idea of solver-aided multi-party negotiation is feasible.

Work remains to expand the set of microservice configuration features the tool supports. Much as we observe numerous forum help requests related to reachability (Sec. 2), there are many cries for help [19, 20, 25–29, 31–33] in forums and even a talk at Kubecon [51] about debugging interactions between other security elements in Istio and K8s, such as authentication.

Beyond Microservices. The problems discussed in this paper are not limited to networking. Many software systems [8] are built as compositions of features, where different teams produce individual components and a third-party might compose them: sometimes as part of the same organization (negotiation mode) and sometimes as black-boxes (conformance mode). The technical solutions in this paper would apply in those settings also.

Envelopes for Stateful Systems. Our current prototype assumes goals can be expressed as stateless predicates in bounded first-order logic, and thus cannot completely reason about stateful systems in its present form. However, much existing synthesis in the stateful setting use techniques [1, 9, 40, 46] that gradually learn constraints from counterexamples. In principle, complete envelopes could be obtained from these constraints after iterating until the solution space is fully characterized (as Cimatti, et al. [9] do), rather than halting at the first correct candidate.

Configuration Privacy. As Newcomer [41] observes, insider threat (whether unintentional or intentional) and regulatory compliance remain major concerns in large microservice deployments. One might therefore reasonably ask how much one administrator can unnecessarily learn about others' configurations via envelopes. In Sec. 2, the envelope revealed the special status of port 23, but little else; we argue that this envelope therefore provides a minimal necessary amount of leakage from the K8s administrator to the Istio administrator. In general, however, our strategy of substitution (Alg. 3) could potentially leak additional fragments of configuration. While basic simplification techniques will mitigate this issue, we are currently pursuing methods to truly ensure minimal leakage.

Extending Beyond 2 Parties. While Muppet currently supports only two administrators, it is possible to increase the number of participants. Handling three or more administrators would involve adding more steps, i.e., increasing the cycle length in Fig. 9. Envelopes would also need to encapsulate the needs of multiple agents (e.g., $E_{\{A,B\}\to C}$), which our algorithm could produce via multiple passes of substitution. One source of future work here would be in separating out the source of obligations to focus negotiation on administrators whose needs are involved in a conflict.

Human Factors. There are numerous human factors issues that need to be addressed to make this work effective for administrators. We expect that giving partial configurations should be straightforward, since these require only marking areas of potential change in a default configuration. Other issues remain, such as:

Presentation In the revision phase, the administrator iteratively refines their candidate configuration based on feedback from the envelope. What form should this feedback take to usefully guide the administrator?
When goals or configurations are rejected, how should this be presented to help and not mislead the administrator? There are logic-based options, such as unsatisfiable cores [49], which can highlight portions of the envelope that are in contradiction with candidate settings. But ultimately administrators may need to understand the content of the envelope; how best can they be presented? Would a textual translation (as in fig. 5) help? Might a graphical presentation help? In particular, we are mindful of past research in formal methods that shows seemingly helpful output modes can actively mislead users [11].
Another option, which we explore here, is to adapt ideas from target-oriented model finding [10]. The resulting system would not outright reject goals or configurations, but rather return a minimally-edited "counter-offer". This may need to be wedded to principled output forms like "why" and "why not" modalities [39].

Semantics The impact of disobeying the envelope varies between domains. The way in which configuration fragments combine (e.g., does deny override?) are also specific to domains. Thus, envelopes may need to become at least partially domain-specific. In the microservices example, obeying the envelope ensures predictable behavior for all parties, but ignoring it can lead to unpredictable behavior unless it is accompanied by enforcement. Analogous ideas from programming languages, related to types and safety enforcement, could be relevant here.

ACKNOWLEDGMENTS

We are grateful to Shriram Rajagopalan for discussions and for allowing the use of his quote. Rodrigo Fonseca, Theophilos Giannakopoulos, Howard Reubenstein, John Emhoff, and the anonymous reviewers provided valuable feedback.

This work was partly supported by the US National Science Foundation. This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL). The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

REFERENCES

[1] R. Alur, R. Bodík, G. Juniwal, M. M. K. Martin, M. Raghothaman, S. A. Seshia, R. Singh, A. Solar-Lezama, E. Torlak, and A. Udupa. Syntax-guided synthesis. In *Formal Methods in Computer-Aided Design*, 2013.

[2] G. Aschemann and R. Kehr. Towards a requirements-based information model for configuration management. In *International Conference on Configurable Distributed Systems*, pages 181–188, 1998.

[3] R. Beckett and R. Mahajan. Putting network verification to good use. In *Workshop on Hot Topics in Networks*, 2019.

[4] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker. Don't mind the gap: Bridging network-wide objectives and device-level configurations. In *Conference on Communications Architectures, Protocols and Applications (SIGCOMM)*, 2016.

[5] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker. Network configuration synthesis with abstract topologies. In *Programming Language Design and Implementation (PLDI)*, 2017.

[6] R. Birkner, D. Drachsler-Cohen, L. Vanbever, and M. Vechev. Config2Spec: Mining network specifications from network configurations. In *Networked Systems Design and Implementation*, 2020.

[7] M. Bravetti, S. Giallorenzo, J. Mauro, I. Talevi, and G. Zavattaro. Optimal and automated deployment for microservices. In *International Conference on Fundamental Approaches to Software Engineering*, 2019.

[8] M. Calder, M. Kolberg, E. H. Magill, and S. Reiff-Marganiec. Feature interaction: A critical review and considered forecast. *Computer Networks*, 41(1):115–141, Jan. 2003.

[9] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta. Parameter synthesis with IC3. In *Formal Methods in Computer-Aided Design*, 2013.

[10] A. Cunha, N. Macedo, and T. Guimarães. Target oriented relational model finding. In *International Conference on Fundamental Approaches to Software Engineering*, pages 17–31. Springer, 2014.

[11] N. Danas, T. Nelson, L. Harrison, S. Krishnamurthi, and D. J. Dougherty. User studies of principled model finder output. In *Software Engineering and Formal Methods*, 2017.

[12] C. Diekmann, J. Naab, A. Korsten, and G. Carle. Agile network access control in the container age. *IEEE Trans. Network and Service Management*, 16(1):41–55, 2019.

[13] C. Diekmann, S. Posselt, H. Niedermayer, H. Kinkelin, O. Hanka, and G. Carle. Verifying security policies using host attributes. In *Formal Techniques for Distributed Objects, Components, and Systems*, 2014.

[14] A. El-Hassany, P. Tsankov, L. Vanbever, and M. T. Vechev. Network-wide configuration synthesis. In *International Conference on Computer Aided Verification*, 2017.

[15] A. El-Hassany, P. Tsankov, L. Vanbever, and M. T. Vechev. NetComplete: Practical network-wide configuration synthesis with autocompletion. In *Networked Systems Design and Implementation*, 2018.

[16] A. Felfernig, G. E. Friedrich, D. Jannach, and M. Zanker. Towards distributed configuration. In *KI 2001: Advances in Artificial Intelligence*, pages 198–212, Berlin, Heidelberg, 2001.

[17] J. D. Guttman. Filtering postures: Local enforcement for global policies. In *IEEE Symposium on Security and Privacy*, pages 120–129, 1997.

[18] A. Hubaux. *Feature-based Configuration: Collaborative, Dependable, and Controlled.* PhD thesis, University of Namur, Belgium, 2012.

[19] Istio forum user bappr. Istio RBAC - 1.1.5 - K8S. https://discuss.istio.io/t/istio-rbac-v1-1-5-k8s/2543, 2019. Accessed June 11, 2020.

[20] Istio forum user bappr. Istio RBAC requires mTLS? https://discuss.istio.io/t/istio-rbac-require-mtls/2797/2, 2019. Accessed June 11, 2020.

[21] Istio forum user claudiobizzotto. Network Policy not taking effect. https://discuss.istio.io/t/networkpolicy-not-taking-effect/3341, 2019. Accessed June 11, 2020.

[22] Istio forum user courcelm. Ingress gateway IP whitelist with AuthorizationPolicy. https://discuss.istio.io/t/ingress-gateway-ip-whitelist-with-authorizationpolicy/5558, 2020. Accessed June 11, 2020.

[23] Istio forum user Fredrik. AuthorizationPolicy and namespaces. https://discuss.istio.io/t/authorizationpolicy-and-namespaces/5399, 2020. Accessed June 11, 2020.

[24] Istio forum user jebinjeb. AuthorizationPolicy not allowing health endpoint. https://discuss.istio.io/t/authorizationpolicy-not-allowing-health-endpoint/6242, 2020. Accessed June 11, 2020.

[25] Istio forum user magic. Multicluster control options for gateway. https://discuss.istio.io/t/multicluster-control-options-for-gateway/6064, 2020. Accessed June 11, 2020.

[26] Istio forum user MarioPeck. Authentication policy origins JWT - internal vs public access. https://tinyurl.com/istio-mariopeck, 2020. Accessed June 11, 2020.

[27] Istio forum user obelisk. Istio and kubernetes network policies. https://discuss.istio.io/t/istio-and-kubernetes-network-policies/4858, 2020. Accessed June 11, 2020.

[28] Istio forum user Peter_Flanagan. RBAC returns either 403 or 302 for each route randomly. https://tinyurl.com/istio-peterf, 2019. Accessed June 11, 2020.

[29] Istio forum user rlljorge. Restrict access by gateway/service using source ip. https://discuss.istio.io/t/restrict-access-by-gateway-service-using-source-ip/6588/3, 2020. Accessed June 11, 2020.

[30] Istio forum user sethokayba. Openshift Istio ServiceEntry. https://discuss.istio.io/t/openshift-istio-serviceentry/4247, 2019. Accessed June 11, 2020.

[31] Istio forum user Steven_O_brien. Application roles and RBAC. https://discuss.istio.io/t/applications-roles-and-rbac/4006, 2019. Accessed June 11, 2020.

[32] Istio forum user y0zg. Jwt tokens propagation between multiple clusters. https://discuss.istio.io/t/jwt-tokens-propagation-between-multiple-clusters/6604, 2020. Accessed June 11, 2020.

[33] Istio forum user yuzisun. RBAC denied for connection check. https://discuss.istio.io/t/rbac-denied-for-connection-check/3627, 2019. Accessed June 11, 2020.

[34] D. Jackson. *Software Abstractions: Logic, Language, and Analysis.* MIT Press, 2 edition, 2012.

[35] D. Jannach and M. Zanker. Modeling and solving distributed configuration problems: A CSP-based approach. *IEEE Transactions on Knowledge and Data Engineering*, 25(3):603–618, 2013.

[36] S. Krishnamurthi and T. Nelson. The human in formal methods (invited talk). In *International Symposium on Formal Methods (FM)*, 2019.

[37] M. Mendonça, D. Cowan, W. Malyk, and T. Oliveira. Collaborative product configuration: Formalization and efficient algorithms for dependency analysis. *Journal of Software*, 3:69–82, 01 2008.

[38] S. Narain, G. Levin, S. Malik, and V. Kaul. Declarative infrastructure configuration synthesis and debugging. *J. Netw. Syst. Manage.*, 16(3), Sept. 2008.

[39] T. Nelson, N. Danas, D. J. Dougherty, and S. Krishnamurthi. The power of "why" and "why not": Enriching scenario exploration with provenance. In *Foundations of Software Engineering*, 2017.

[40] T. Nelson, N. Danas, T. Giannakopoulos, and S. Krishnamurthi. Synthesizing mutable configurations: Setting up systems for success. In *Workshop on Software Engineering for Infrastructure and Configuration Code*, 2019.

[41] K. Newcomer. Securing a multi-tenant Kubernetes cluster. https://www.infoq.com/presentations/securing-kubernetes-cluster/, 2019. Accessed June 26, 2020.

[42] Y. Permpoontanalarp and C. Rujimethabhas. A unified methodology for verification and synthesis of firewall configurations. In S. Qing, T. Okamoto, and J. Zhou, editors, *Information and Communications Security*, pages 328–339, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[43] C. Prakash, J. Lee, Y. Turner, J.-M. Kang, A. Akella, S. Banerjee, C. Clark, Y. Ma, P. Sharma, and Y. Zhang. PGA: Using graphs to express and automatically reconcile network policies. In *ACM Computer Communication Review*, page 29–42, 2015.

[44] M. Reitblatt, M. Canini, A. Guha, and N. Foster. FatTire: Declarative fault tolerance for software-defined networks. In *Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, 2013.

[45] S. Saha, S. Prabhu, and P. Madhusudan. NetGen: Synthesizing data-plane configurations for network policies. In *Symposium on SDN Research (SOSR)*, 2015.

[46] A. Solar-Lezama, L. Tancau, R. Bodik, S. Seshia, and V. Saraswat. Combinatorial sketching for finite programs. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2006.

[47] R. Soulé, S. Basu, R. Kleinberg, E. G. Sirer, and N. Foster. Managing the network with Merlin. In *Workshop on Hot Topics in Networks*, 2013.

[48] stack overflow forum user: Leonardo Carraro. Kubernetes Network Policy - Allow specific IP. https://stackoverflow.com/questions/53617527/kubernetes-network-policy-allow-specific-ip, 2018. Accessed June 11, 2020.

[49] E. Torlak, F. S.-H. Chang, and D. Jackson. Finding minimal unsatisfiable cores of declarative specifications. In *International Symposium on Formal Methods (FM)*, 2008.

[50] E. Torlak and D. Jackson. Kodkod: A relational model finder. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 632–647. Springer, 2007.

[51] M. Turner. Walk-through: Debugging an RBAC problem in Istio (but without the swearing. https://tinyurl.com/kubecon-turner-k8s, 2019. Accessed June 11, 2020.

[52] Y. Wang, C. Jiang, X. Qiu, and S. G. Rao. Learning network design objectives using a program synthesis approach. In *Workshop on Hot Topics in Networks*, 2019.

[53] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):673–685, 1998.

[54] S. Zhang, A. Mahmoud, S. Malik, and S. Narain. Verification and Synthesis of Firewalls using SAT and QBF. *IEEE International Conference on Network Protocols (ICNP)*, 2012.