The 5th International Workshop on Information Quality and Quality of Service for Pervasive Computing 2013, San Diego (22 March 2013)

Compatibility among Diversity

Foundations, lessons, and directions of semantic communication

(Invited Paper)

Brendan Juba
School of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138
Email: bjuba@alum.mit.edu

***Abstract*—We give an overview of a theory of semantic communication proposed by Goldreich, Juba, and Sudan. The theory is intended to capture the obstacles that arise when a diverse population of independently designed devices must communicate with one another. The aim of the theory is to provide conceptual foundations for the design and evaluation of devices that are compatible with such a diverse population. Conclusions drawn from the theory *(i)* identify a kind of information-sensing that is inherently necessary for compatibility whenever the population is sufficiently diverse and *(ii)* identify tensions between the richness of diversity and the computational cost of coping with such diversity in a population. We will review how these considerations are reflected in the formulation and design of an example application, a self-patching packet network stack. In particular, this application will illustrate the utility of explicit consideration of various computational complexity measures in addressing both *(i)* and *(ii)*. We will also review work aimed at identifying kinds of populations across which compatibility can be achieved efficiently.**

I. Introduction

Problems of incompatibility – colloquially, misunderstanding – naturally arise in systems without a centrally coordinated design. Incompatibility can arise in any large enough system due to lack of coordination among its engineers: an infamous example is the failure of the Mars Climate Orbiter due to one team of engineers writing an application employing imperial units of measurement that communicated with another application, designed by a different team, that expected to communicate in metric units [1]. As the amount of shared context among these designers is diminished, the likelihood of miscommunication among parts of the system naturally increases; what is natural and simple in one context may well seem strange to the designer of another artefact, serving a different purpose, working at a different time. Within an organization or, more broadly, within an engineering discipline, these kinds of errors are usually controlled by centrally imposing *standards*, but standards-making bodies can only fully anticipate domains of limited scope. The scope and richness of ambitious projects may extend beyond the domains envisioned by any one standards committee, across multiple domains. In particular, *"pervasive computing,"* by definition, is intended to enable rich applications that span diverse domains. Due to the combination of diversity of requirements and domains, and the decentralization of design in the networked applications underpinning pervasive applications, it seems inevitable that miscommunication will occur. The question is, how can we manage such misunderstanding?

Capturing, let alone addressing, these problems of incompatibility requires a view of the system that extends beyond the communications channel, in contrast to Shannon's theory of communication [2]. Actually, more accurately, the approach to this problem that is naturally suggested by Shannon's theory – that we should attempt to learn the codebook used by our partner – is far too hard to be of use. The *indeterminacy of translation* as asserted by Quine [3, Chapter II], [4] means that this problem cannot be solved in full generality (cf. also [5, Section 2.3] for a more formal example). By contrast, if we adopt more limited ambitions in terms of the effect of our system on its operating environment, as advocated by Goldreich, Juba, and Sudan [6], most natural communications problems seem to be solvable. Fortunately, essentially every natural problem can be cast as achieving some desired effect or another on an operating environment, so this adoption of a wider model comes at no cost to generality, only to the model's complexity. This widening of the model of communication to incorporate a model of the operating environment is what makes this a "semantic" communication model.[1]

In the present article, we will review the work on addressing compatibility problems under this semantic communication model. We will begin in Section II by describing the model in more detail, and outlining the main theorems, characterizing a kind of information-sensing that is necessary and sufficient to achieve a given effect on the environment. This characterization provides a useful guide to approaching the formulation of communication problems, allowing us to consider the information-sensing aspect separately from the *learning* aspect that, by contrast, actually poses a formidable obstacle to practical systems. Work on this latter problem is discussed in Section III, summarizing the directions pursued

Preparation of this survey supported by ONR grant no. N000141210358

[1]This is in contrast to "semantics" as construed in a formal logic or linguistics, which aims to assign a precise meaning to the symbols and form of a syntactic expression; linguists would call what we study "pragmatics" by way of contrast. In Weaver's terms [7], our theory addresses bridging the gap from level A, the communications channel, to level C, the effectiveness of the communication, in a sense side-stepping level B, the "semantic" level, corresponding to the notion from logic. We take this approach because, while such a precise mapping could be useful, we know no grounds on which we can take its existence for granted. Unless we know *something more* about the structure of the system, we know of no "semantics" other than the pragmatic level.

978-1-4244-9529-0/13/$31.00 ©2013 IEEE

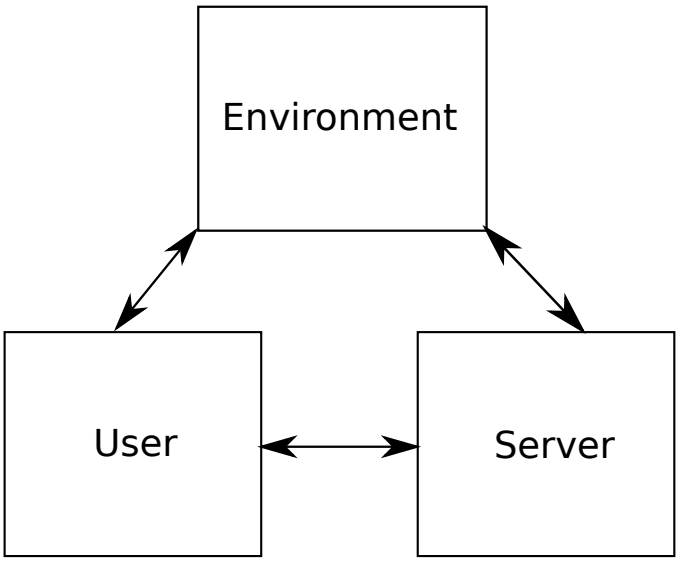

Fig. 1. The basic three entity system in the semantic communication model. Arrows represent a two-way communication channel.

to date. We will illustrate how this work could inform the development of a more concrete application in Section IV, as we review the development of a self-patching packet network stack. The approaches to learning pursued to date are still works in progress, though, and we conclude by summarizing some next steps for current and future work in Section V.

II. The basic theory

A. Model of communication

Our model focuses on communication between a pair of entities, as this suffices to capture the issues of miscommunication that may arise.[2] One entity, the *user,* represents the portion of the system we wish to design to communicate on our behalf, and the other, the *server,* represents another portion of the system from which we seek assistance via communication. For simplicity, we consider a synchronous model of communication, proceeding in discrete rounds, in which the entities are described by their *strategies*: (possibly randomized) mappings that take the state of an entity and its incoming messages to a new state and a new set of outgoing messages on each of its communication channels. As alluded to in the Introduction, for the sake of defining correctness in the presence of miscommunication, we introduce a *third* formal entity into this system that we call *the environment,* which represents the rest of the system and/or the operating environment of the user and server (see Figure 1 for a diagram). We assume that communication always has a purpose that can be formulated in terms of the states of the environment; precisely, we assume the existence of a *referee* predicate defined over (sequences of) states of the environment that, together with the environment, defines a *goal of communication* for the user.

For example, we might informally consider a goal in which we wish to control robots that store and retrieve goods in a warehouse. The environment might reasonably model the state of the warehouse, and the server strategy might be taken to correspond to the software controlling one of the robots. The communications channels joining the entities might variously correspond to sensors or network links. The goal might correspond to a family of instances which specify that goods are to be moved from one location in the warehouse to another, which is then the condition checked by the referee. The user remotely controls the robot over the network to achieve these ends in the system, and our objective as designers would be to give a user strategy that accomplishes this.

More precisely, there are two main families of goals that have been considered, that we refer to respectively as *finite* and *infinite* goals. Finite goals, known elsewhere as "goals to achieve," are specified by a termination condition: the user should terminate its execution when the environment is in a satisfactory state. Correspondingly, the referee defining a finite goal only examines a single state of the environment to determine whether the goal is achieved. Such goals naturally represent subgoals or elementary subroutines that we might wish to carry out, and are studied by Juba [5, Chapters 2 and 5]. Infinite goals, by contrast, are defined in terms of an entire, infinite execution of the environment in the three-entity system. Such goals are suitable for reactive systems such as network infrastructure or operating systems that are intended to remain available in a single execution. We almost always restrict our attention to infinite goals having the form of a series of *sessions,* where each session could be viewed as an instance of a finite goal; infinite goals allow richer possibilities than a sequence of finite goals if we are willing to tolerate the failure of a few (a bounded number) of the sessions. This richer but somewhat more complex setting is the main focus of Goldreich, Juba, and Sudan [6]. This collection of "multi-session" infinite goals is only one natural kind of infinite goal, though, and the work of Goldreich, Juba, and Sudan only hints at the variety of further kinds of possible infinite goals.

In this enriched communication model, we can formulate our compatibility problems by supposing that there is more than one possible server strategy. That is, we suppose that there is a *set* of possible server strategies, and an actual server strategy for our three-entity system is selected adversarially from this set. We say that a given user strategy is *compatible* with the set of server strategies for the given goal if, for every server strategy from the set, the user strategy satisfies the referee in the three-entity execution.

B. Sensing for a goal

The key theoretical notion, for finite goals in particular, is a kind of (Boolean) feedback we call *sensing* (for a goal), that is computed from the user's local view of the system by a *sensing function*. As the name suggests, sensing functions are *potentially capable* of detecting success (or failure) of a goal, but need not *always* detect it. In the case of finite goals, this means that the sensing function can sometimes detect when the referee is satisfied. For example, in the warehouse goal, if our sensors don't have a line-of-sight to the point of delivery of the goods, then they may not detect that the package has been carried to the proper location, and we generally deem that acceptable. What we do require, for finite goals, is that there is some further action the user can take (or some way of going about achieving the goal) in the system that permits

[2]Settings with more entities may be treated, to some extent, by separately considering the pairwise interactions [6, Section 7].

us to obtain a success signal from the sensing function. For example, this may be accomplished in the warehouse example by removing the obstruction to our line-of-sight or arranging for some other robot to signal to us that the package is delivered. If there is some feasible, efficient user strategy that always triggers the sensing function's success signal with a given server, then we say that the sensing function is *viable* for the server and goal.

Of course, the intuition above presumes that when the sensing function detects success, then this actually means that (except for perhaps exceptionally rare circumstances) the referee is satisfied and therefore the goal is achieved. We refer to this property of the sensing function as *safety*. Precisely, we consider safety as a property of a fixed goal and a given *set* of servers—ideally, with all possible servers, but in general possibly relying on some special property of the servers. We note that in order for sensing functions to be useful to us, they must have some form of these two properties: the always-positive sensor is viable but unsafe, whereas the always-negative sensor is safe but unviable.

Another, more subtle example of sensing can be found in outsourcing computation, e.g., to a cloud, the goal originally considered in the work of Juba and Sudan [8]. Here, sensing is achieved by obtaining an interactive proof of the correctness of the computation from the server: the soundness of the proof immediately guarantees safety, and viability can be obtained since an efficient user strategy can extract such an interactive proof from the server (by requesting a series of computational tasks from the server/cloud).

More concretely, suppose we have a user with limited memory that wishes to utilize the vaster capabilities of a cloud to solve a problem. Suppose that some small-space user strategy could interface with the cloud, and that the problem is complete for polynomial-time (e.g., linear programming or circuit evaluation). Then, completeness for polynomial time guarantees that there is a small-space reduction that obtains full computation histories from solving instances of the original problem. These computation histories may then be checked space-efficiently by checking transitions in the history at random, which only requires storing a pointer into the history and a constant amount of information about the local state. By checking enough random transitions, one obtains confidence that the entire history is consistent, and then it is safe to output the result indicated at the end of the history. Of course, by composing the small-space reduction with the strategy for interfacing with the cloud and this history-verification strategy, one obtains a small-space user strategy for which these history checks will all pass, i.e., yielding viability of the corresponding sensing function (cf. [5, Section 5.5.2] for more).

We consider a slightly different, weaker notion of sensing for infinite goals. Recalling that the class of infinite goals we commonly consider, multi-session goals, can be viewed as an infinite sequence of finite goals in which we are satisfied if we fail in no more than a finite number of these sub-goals, it is natural that the notion of sensing should likewise be weaker. This weaker notion of safety essentially asserts that when a sub-goal is failed, the sensing function should give a negative indication not too much later (unless some corrective action has been taken in the interim). The corresponding viability notion demands that there is a user strategy such that after some bounded "grace period," the sensing function ceases to produce negative indications. The key property of both of these respective notions of sensing is that they suffice for achievement of goals in their respective settings. This is the subject of the main theorems on semantic communication, as we review in the next section.

C. The theorems: sensing and compatibility

Sensing naturally suggests a control-theoretic architecture for a user strategy: the sensing function computes a feedback signal that is fed into a separate *controller* strategy that attempts to find a strategy that satisfies the sensing function. Such architectures are quite natural and appear in many other areas such as autonomic computing [9]. The main theorem for finite goals essentially asserts that such a control-theoretic architecture based on sensing functions is sufficient and moreover can be assumed without loss of generality.

Theorem 1 (Theorem 2.25, [5] - informal statement): Fix a finite goal of communication and a set of server strategies. There is a user strategy that is compatible with the set of server strategies for the goal if and only if there is a sensing function for the goal that is safe for the set of servers and viable with every server in the set.

The generic solution to controller problem used in the proof of Theorem 1 is based on Levin's enumeration technique [10] and is not so efficient—we will return to this point in Section III. The key feature is merely that it is a fixed overhead that depends only on the server, not on the complexity of the goal instance. Still, the positive direction of an essentially analogous theorem for space-bounded computation [5, Theorem 5.22], together with our sensing functions for polynomial-time complete computation yield for example a construction of *universal* small-space user strategies for outsourcing polynomial-time complete computations.

More generally, we can generically define universal compatibility as follows: we say that a server is *helpful* for a goal if there exists a (feasible) user strategy that is compatible with it, i.e., achieves the goal with that server. Now consider the set of all servers that are helpful for a goal: this is the most diverse possible set of servers with which a user could be compatible for the fixed goal. So, we say that a user strategy compatible with this set is *universal*. Sensing functions like the one we described for computing that are viable with every helpful server and safe with every helpful server suffice, by Theorem 1, for the construction of universally compatible user strategies for a given goal.

The direction of Theorem 1 asserting that a suitable sensing function always exists turns out to be useful for studying what is necessary for compatibility. In particular, for the study of universal compatibility, it enables the following significant theorem:

Theorem 2 (Theorem 2.37, [5]): If a sensing function is safe with the set of all helpful servers for a goal, then it is actually safe with all servers.

Theorem 2, when combined with the characterization of Theorem 1, establishes that our ability to achieve correctness with a sufficiently diverse set of servers is intimately connected to our ability to obtain reliable information about where our

goals stand. In the context of information-processing goals, this requirement of reliable information in the face of potentially adversarial behavior leads us to necessarily consider *cryptographic* techniques in order to enable *compatibility.* For example, the theorems immediately imply that the computational problems that can be universally outsourced are precisely those that possess a certain kind of interactive proof system [8]. We will see a particularly concrete example of this invocation of cryptography in Section IV, where security features originally included in the optional IPsec extension to IP [11] turn out to be exactly what we require to enable self-patching.

As for infinite goals, the positive direction of Theorem 1 – establishing that sensing suffices for the construction of a compatible user strategy – still holds [6, Theorem 4.6, etc.] as does the analogue of Theorem 2 [6, Theorem 4.26]. But, the characterization of Theorem 1 *does not* hold in this setting. Indeed, it turns out that there are infinite goals for which there exist user strategies that are compatible with a diverse collection of servers, even though the user strategy receives no feedback on its progress in any meaningful sense [6, Section 4.3]. Naturally, this example relies heavily on the weakness of our criteria for success in multi-session goals: the number of failed sessions may be arbitrarily large as long as it is finite. Sensing still remains useful in this setting at least as a means to construct user strategies that guarantee bounds on the number of failures [6, Theorem 4.17], and the analogue of Theorem 2 at least addresses the kind of sensing used in this application.

III. Efficiency of learning

We noted that our proof of Theorem 1 uses a somewhat unsatisfactory enumeration-based approach to the "controller problem," that is, to searching for a satisfactory user strategy. Unfortunately, it turns out that this is inherent: solutions to the controller problem that are sufficiently general for universally compatible users (and sufficient to prove Theorem 1) must inherently be slow. This is actually quite intuitive, as its scope includes servers employing communications protocols that use, e.g., "magic strings" to identify the protocol being used. Specifically, we suppose that the protocol specifies that the server ignores the user unless the user provides a specific "magic string" prefixing its first message. Such a "magic string" is essentially a *password,* and if the class of servers contains servers using every such password, it necessarily will take any given user strategy a long time before it tries at least one of these passwords [6, Theorem 4.25] (or [5, Theorem 4.3] for finite goals). So, if the set of servers is rich enough to include those demanding all possible such "magic strings," users compatible with all of them must experience substantial overhead with some (actually, most) of them.

To clarify, we don't believe that anyone should be particularly happy with strategies that perform as poorly as an enumeration-based strategy. The point, rather, is that improving on these kinds of strategies requires some kind of knowledge of the server (and what it expects). Somehow, the set of servers must be restricted so that not all of these password-protected servers are within the scope we wish to address. Two distinct approaches have been considered so far, as we review below. As we will note in Section V, though, the main directions for future work still concern this general problem.

A. Efficiency from similar "beliefs"

The first approach, suitable for the setting of finite goals, appears in the work of Juba and Sudan [12]. They suppose that a server is designed to work well with a certain kind of population of users, reflected in a probability distribution over user strategies. More precisely, fixing a goal and a distribution over user strategies, we suppose that the server is designed so that a large fraction of users under the distribution achieve the goal quickly. Now, suppose that the user can sample from a distribution over user strategies that is close (in statistical distance) to the server's "belief" about the user population (the distribution used in the server's design and evaluation). In this case, it is possible to design a controller strategy for the user based on sampling from the user's distribution that performs roughly as well at achieving the goal as a typical user under the server's beliefs—the overhead is given by some modestly large constants and a penalty for the gap between the user's distribution and the server's belief distribution. So, if the server has actually been designed to work well with this user population, our user strategy also works about as well.

B. Efficiency from simple strategies

The second approach, considered by Juba and Vempala [13], assumes that the sensing function is viable with *very* simple user strategies, and (in the multi-session goal model) gives feedback essentially immediately. In such a case, a controller strategy turns out to be precisely an algorithm for on-line learning in the mistake-bound model (originally introduced by Bārzdiņš and Freivalds [14] and developed by Littlestone [15]). Here, the concept representation learned by the algorithm corresponds to the class of user strategies searched by the controller. This gives us a few positive examples of efficient algorithms, but the problem here is that very little is learnable in this model. So these controllers must search for strategies that only compute, e.g., threshold functions of their inputs. We will see another variant of this general approach in the next section.

IV. Example: self-patching packet networks

We now consider a problem inspired by the transition from IPv4 to IPv6, considered by Juba [5, Chapter 9]. Suppose that our devices are communicating over a packet network which is being upgraded so that the previous packet format is no longer supported, and packets sent under this old format may be dropped or incorrectly delivered. We would like to design an encoding and decoding strategy for the network's packets that remains compatible with the new format, that is *"forward-compatible."* Naturally, the network is the "server" in this example, and we would like to design an end-user stack that is compatible with an entire *set* of different networks running different protocols that could potentially be "the next version."

This problem could be relevant to pervasive computing since, if we are going to endow most of the everyday objects and infrastructure around us with a network connection, we can't expect the owners to have much hope of managing manual "upgrades" of the software used by all of them. (This is not a question of "incentives"—it may easily be simply infeasible.) Furthermore, allowing the devices to receive patches distributed over the network exposes a vulnerability of

terrifying proportions in this everyday infrastructure. But then, what will happen when the current network protocol is phased out? Although the most recent transition has been handled by running IPv4 and IPv6 in a "dual-stack" configuration, this does not seem like a solution that will scale in the long term, as it means that the complexity of our network infrastructure must strictly increase over time. Furthermore, it leaves any legacy IPv4 devices incapable of communicating with future IPv6-only devices. Given the stable of bad alternatives available, the "forward-compatibility" approach presents itself as a potentially hopeful direction. In any case, this problem also turns out to serve as a concrete illustration of how the theory described in the previous sections impacts and may usefully inform our approaches to solving compatibility problems.

The problem that immediately arises is that the specification for the IP layer provides *no* feedback: When a packet is sent, the protocol never provides a packet-level confirmation. Likewise, when a candidate decoding of an incoming packet is returned to the end-user, no confirmation of whether or not the decoding was performed correctly is ever expected. Supposing we treat the sending and decoding of individual packets as a goal that we would like to reliably achieve, Theorem 1 tells us that we can't take a specification like that of IP alone since there is no way to provide sensing. If we redefine the IP layer to (somehow) receive these two kinds of feedback, then since these comprise the full functionality we require from IP, we know the compatibility problem can be solved: we can at least design an enumeration-based controller, though of course its efficiency is still lacking. Still, we focus on how to provide the necessary kind of feedback within the context of a higher-level protocol or application, in which the contents of the packets are *not* completely arbitrary (as opposed to obtaining some feedback from beyond the network, which would be another possibility).

A natural choice of such context to consider is the TCP protocol: in most cases where IP is used, TCP is running on top of IP, and moreover, TCP provides feedback for sending by design. That is, we will assume that *two* end-users are using (our own modified version of) TCP in order that at least one of the users may learn the new packet format. Then, by using a variant of the acknowledgement scheme already present in TCP, our modified version of TCP could verify successful sending of our packets if we only knew how to decode packets at the IP layer. We thus reduce the number of obstacles to correctness to just one, verifying decoding.

The difficulty with decoding is reminiscent of the problem with studying compatibility using Shannon's model: a packet is simply a string, and if the mappings from packets to their payloads is completely arbitrary, then the problem is impossible. The key to solving this problem is that we wouldn't realistically expect these mappings to be *completely* arbitrary. Complexity theory now presents one way forward: We will formulate the problem more carefully, restricting our attention to simple encoders that use a bounded amount of memory and compute encodings in a single pass—real packet encodings naturally tend to be sufficiently simple that they can be computed within such limited resources. We will likewise consider only "next versions" of the packet format for which short "patch" programs, computable in similar requirements, convert packets in the old format to packets in the new format. That is, instead of seeking a "universal" packet network stack, we will only seek a "self-patching" stack that continues to work so long as the patches are sufficiently simple.

We can now finally solve the verification problem: a mere hash function tag (e.g., as proposed by Gilbert, MacWilliams, and Sloane [16]) is provably secure as a signature scheme against such simple functions. It is safe because we can drive down the probability of collisions low enough so that for every possible message, state of the encoder, and incorrect candidate patch, an incorrect message with a correct hash under the randomly chosen function is only produced with exponentially small probability. The length of the tag required is linear in the sum of the representation lengths of the messages, states, and patches. (Notice that we do crucially rely on the bounds on the latter two complexities that we introduced in our reformulation.) We note that essentially such a message authentication feature is actually already optionally provided as part of IPsec [11]; we merely require that the users use such a feature (specifically sharing, e.g., the hash function as a key). Together with the previously mentioned variant of the acknowledgement scheme already present in TCP, we can thus obtain the feedback necessary to ensure correctness and thereby complete a basic self-patching network stack.

Moreover, the search for an encoding function is now over these short patches instead of over the entire algorithm. When the patches are short but the overall algorithm is long, this approach potentially yields much less overhead. This is a different example of how the efficiency may be improved by assuming that the user strategy has a simple form, cf. Section III-B. We also briefly note that there are learning algorithms in this case – where, due to the unreliability of the network, even packets using the correct encoding may sometimes be dropped – that perform much better than our enumeration-based strategies. Namely, the controller's search for a patch may now be cast as a nonstochastic multi-armed bandit problem, and the algorithm of Auer et al. [17] for example would suffer far less overhead than a naïve enumeration. Any further improvement of the efficiency of such self-patching is an open question.

RECAP. Our search for a packet network stack that would feature "forwards-compatibility" with a diverse collection of candidates was largely guided by the need to obtain feedback, as outlined by Theorem 1. It motivated us to make two crucial assumptions: first, that a protocol like TCP would be running on top of the packet network layer so that sending would be verifiable; and second, that the encoding and patching were performed by functions that were "simple" in a complexity-theoretic sense so that decoding of a packet would be verifiable using essentially a feature for message authentication already present in IPsec. This latter assumption also opened the door to a reduction of the overhead for the search for a user strategy, as short patch programs can be searched relatively efficiently (notice, only relatively few "passwords" can be encoded in a short program).

V. CURRENT AND FUTURE DIRECTIONS

We now conclude by describing the current and future directions for work in this semantic communication model. Despite the work described in Section III, it is still premature

to expect acceptable performance out of the algorithms we have described, and thus extensive empirical work based on the solutions we've proposed is almost surely likewise premature.[3] Thus, most of the current directions concern means for enabling systems that will be efficient while still retaining nontrivial flexibility.

A. On-line learning approach

In Section III-B, we noted that the drawback of the on-line learning approach of Juba and Vempala [13] was that only extremely simple functions can be learned in the on-line mistake-bound learning model. That is because the success/fail feedback provided by the sensing functions is rather impoverished. However, it seems likely that in a real-world scenario, richer kinds of sensing should be available, e.g., not just "right" or "wrong," but "off by x." Juba and Vempala note at least one such example where somewhat richer feedback leads to an exponential improvement in performance [13, Example 17]. Likewise, in the case of the self-patching network stack, we might wish to allow the controller algorithm to choose some packets to send, which might be useful in enabling a kind of "query" learning. Of course, to be truly meaningful, all of these approaches would require fixing a real-world problem and seeing what kind of sense data is not only sufficient, but also feasible to collect.

B. Beliefs approach

The main obstacle to the approach described in Section III-A that was not dealt with in the work of Juba and Sudan [12] is that while the approach assumes that the server is designed to efficiently serve a large population of user strategies that can be efficiently sampled, we actually do not know how to design servers that perform well with respect to natural, sampleable families of algorithms for nontrivial goals. Although in light of this lacuna it may seem like this approach merely begs the question, we note that the presence of a distribution over the user strategies leads to a problem that is quite different from the "worst-case" compatibility problems that we had originally set out to address. In fact, it seems like this remaining "average-case" performance optimization problem is roughly similar to some problems being addressed by work elsewhere, for example, on self-optimizing systems in autonomic computing [9]. We are therefore hopeful that this approach is not so far from realization after all.

C. Knowledge-based approach

Yet another natural approach to the controller's problem is to use existing AI techniques for planning for control. Using such planning techniques requires first learning the server's behavior somehow, and therefore moreover, requires that the server's behavior is somehow learnable (e.g., obeys simple, observable time-invariant relations). Valiant's PAC semantics [18] provides a (logical) semantics suitable for the purposes of integrating learned rules into classical AI-style reasoning approaches. Remarkably, it turns out that due to how the learned rules are used in reasoning, we might be able to lift the usual representation restrictions imposed on learning [19] which pose the main barrier to, e.g., the on-line learning approach above. The only potential barrier to such an approach is then the speed of algorithms for planning, but such problems at least have been the focus of much research over the years, and are actually tractable in practice in some cases.

[3] We also have not attempted to evaluate the performance of self-patching networks partially on account of the performance and suitability of the scheme seeming to depend crucially on the choice of patch encoding, which is separate issue entirely from those we have considered so far.

REFERENCES

[1] A. G. Stephenson, L. S. LaPiana, D. R. Mulville, P. J. Rutledge, F. H. Bauer, D. Folta, G. A. Dukeman, R. Sackheim, and P. Norvig, "Mars climate orbiter mishap investigation board phase I report," NASA press release, 1999.

[2] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379–423, 623–656, 1948.

[3] W. van Orman Quine, *Word and Object*. Cambridge: MIT Press, 1960.

[4] ——, "Ontological relativity," in *Ontological Relativity and Other Essays*. New York: Columbia University Press, 1969.

[5] B. Juba, *Universal Semantic Communication*. Berlin: Springer, 2011.

[6] O. Goldreich, B. Juba, and M. Sudan, "A theory of goal-oriented communication," *J. ACM*, vol. 59, no. 2, pp. 8:1–8:65, 2012.

[7] W. Weaver, "Some recent contributions to the mathematical theory of communication," in *The Mathematical Theory of Communication*, C. E. Shannon and W. Weaver, Eds. University of Illinois Press, 1949.

[8] B. Juba and M. Sudan, "Universal semantic communication I," in *Proc. 40th STOC*, 2008, pp. 123–132.

[9] M. C. Huebscher and J. A. McCann, "A survey of autonomic computing—degrees, models, and applications," *ACM Computing Surveys*, vol. 40, no. 3, pp. 7:1–7:28, 2008.

[10] L. A. Levin, "Universal search problems," *Probl. Inform. Transm.*, vol. 9, pp. 265–266, 1973.

[11] S. Kent and K. Seo, "Security Architecture for the Internet Protocol," RFC 4301 (Proposed Standard), Internet Engineering Task Force, December 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4301.txt

[12] B. Juba and M. Sudan, "Efficient semantic communication via compatible beliefs," in *Proc. 2nd Innovations in Computer Science*, 2011, pp. 22–31.

[13] B. Juba and S. Vempala, "Semantic communication for simple goals is equivalent to on-line learning," in *Proc. 22nd ALT*, ser. LNAI. Springer, 2011, vol. 6925, pp. 277–291.

[14] J. Bārzdiņš and R. Freivalds, "On the prediction of general recursive functions," *Soviet Math. Dokl.*, vol. 13, pp. 1224–1228, 1972.

[15] N. Littlestone, "Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm," *Mach. Learn.*, vol. 2, no. 4, pp. 285–318, 1988.

[16] E. N. Gilbert, F. J. MacWilliams, and N. J. A. Sloane, "Codes which detect deception," *Bell Sys. Tech. J.*, vol. 53, pp. 405–424, 1974.

[17] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "The non-stochastic multiarmed bandit problem," *SIAM J. Comput.*, vol. 32, no. 1, pp. 48–77, 2003.

[18] L. G. Valiant, "Robust logics," *Artificial Intelligence*, vol. 117, pp. 231–253, 2000.

[19] B. Juba, "Learning implicitly in reasoning in PAC-semantics," arXiv:1209.0056v1 [cs.AI], 2012.