An Efficient Interaction Protocol Inference Scheme for Incompatible Updates in IoT Environments

HEESUK SON and DONGMAN LEE, Korea Advanced Institute of Science and Technology

Incompatible updates of IoT systems and protocols give rise to interoperability problems. Even though various protocol adaptation and unknown protocol inference schemes have been proposed, they either do not work where the updated protocol specifications are not given or suffer from inefficiency issues. In this work, we present an efficient protocol inference scheme for incompatible updates in IoT environments. The scheme refines an active automata learning algorithm, L*, by incorporating a knowledge base of the legacy protocol behavior into its membership query selection procedure for updated protocol behavior inference. It also infers protocol syntax based on our previous work that computes the most probable message field updates and adapts the legacy protocol message accordingly. We evaluate the proposed scheme with two case studies with the most popular IoT protocols and prove that it infers updated protocols efficiently while improving the L* algorithm's performance for resolving the incompatibility.

CCS Concepts: • **Computer systems organization** → **Heterogeneous (hybrid) systems**; **Reconfigurable computing**; • **Networks** → **Protocol testing and verification**; • **Software and its engineering** → **Interoperability**; • **Theory of computation** → **Active learning**; • **Computing methodologies** → *Probabilistic reasoning;*

Additional Key Words and Phrases: Knowledge-based inference, protocol adaptation, interoperability, protocol heterogeneity, meaningful interaction, Internet of Things

ACM Reference format:
Heesuk Son and Dongman Lee. 2021. An Efficient Interaction Protocol Inference Scheme for Incompatible Updates in IoT Environments. *ACM Trans. Internet Technol.* 22, 2, Article 54 (October 2021), 25 pages.
https://doi.org/10.1145/3430501

1 INTRODUCTION

As IoT devices turn into smart objects having a certain level of autonomy, they have pervasive opportunities to interact and communicate among themselves as active participants [5, 43]. Such increasing interaction opportunities enable a group of smart objects to collaborate toward the common good as if they are a single organism [38, 46]. For example, in edge computing environments, adjacent smart objects can accomplish a complex task with low network latency by sharing

Heesuk Son's current affiliation is Facebook, USA.
This work was supported by an Institute of Information and Communications Technology Planning and Evaluation (IITP) grant funded by the Korean government (MSIT) (No. 2019-0-01126, Self-learning based Autonomic IoT Edge Computing).
Authors' address: H. Son and D. Lee (corresponding author), Korea Advanced Institute of Science and Technology, 291, Daehak-ro, Yuseong-gu, Daejeon, Republic of Korea, 34141; emails: heesuk.chad.son@gmail.com, dlee@kaist.ac.kr.

1533-5399/2021/10-ART54 $15.00
https://doi.org/10.1145/3430501

duties for sub-tasks and federating their computing capabilities [22, 33, 39]. However, high-level requirement changes often induce incompatible protocol updates [14], which disrupt a large portion of smart object interactions and suppress the total available market (TAM) of IoT [38]. To resolve protocol incompatibility, various protocol adaptation schemes [4, 6, 7, 9, 17, 28, 35] have been presented for the past decades. Given the specifications of interacting protocols, a dedicated adapter translates between heterogeneous syntax (i.e., message structure or interface signature) and merges their behaviors (i.e., message exchange or interface invocation order).

Despite their effectiveness, an IoT environment is highly dynamic, and it is not realistic to assume that protocol specifications of encountered IoT devices are shared in advance. Some of the protocol adaptation schemes leverage active automata learning (L* algorithm [2]) to infer an interaction target protocol's unknown behavior at runtime without its corresponding specification. Given a set of compatible protocol syntax, the algorithm learns a behavior model through multiple queries and reactions. However, if the syntax incompatibility issue is not resolved in advance, the behavior inference scheme loses its learning target and cannot even initiate a single inference procedure. Furthermore, the L* algorithm suffers from a large number of membership queries, which take a long time to complete a behavior learning task. Considering that a large portion of IoT interactions is made at runtime, a protocol adaptation scheme needs to infer both updated protocol syntax and behavior efficiently before the interaction opportunity is gone.

In this article, we present an efficient interaction protocol inference scheme for incompatible updates in IoT environments. To infer an updated protocol efficiently without its corresponding specification, we leverage the protocol knowledge base as a helpful bias. With respect to updated syntax inference, we leverage our previous work [40] that utilizes the knowledge over various IoT protocols for computing probable message field updates and adapting the legacy message. Once the incompatible protocol syntax is successfully resolved, a protocol behavior inference procedure starts. For efficient behavior inference, we modify the original L* algorithm by incorporating a knowledge base about the legacy protocol's behavior into the algorithm's membership query selection process. By having a legacy protocol's message exchanging behavior, we can identify which message sequences can be transmitted by the legacy protocol agent. Then, the modified L* algorithm can avoid composing membership queries for message sequences that are not supposed to be transmitted by the agent. Such avoided membership queries could learn unnecessary behaviors that are never likely to be performed between the protocol agents. Eventually, the total number of membership queries drastically decreases, and a more compact behavior model for the updated protocol agent can be learned.

For evaluation, we conduct two case studies with real update scenarios: **Message Queuing Telemetry Transport (MQTT)** and **Service Location Protocol (SLP)**. MQTT and SLP are representative IoT protocols that perform data exchange and service discovery, respectively. These two different protocols allow us to prove our proposed scheme's applicability across different types of IoT interactions. The case study results show that our proposed scheme successfully infers the updated message syntax despite a certain amount of syntactic errors and semantic loss. In addition, the syntax inference completes shortly enough to maintain an encountered interaction opportunity. In protocol behavior learning, we verify that incorporating the knowledge base improves the original L* algorithm's efficiency drastically. The number of required membership queries decreases down to 23.8%, and the result behavior model of the proposed knowledge-based inference is twice as compact as that of the original L* algorithm.

In summary, our contributions on top of our previous work [40] are as follows:

- We propose an efficient protocol inference scheme for both message syntax and behavior updates with a lesser number of queries than the state-of-the-art algorithm, which enables

interactions between incompatible IoT protocol versions without corresponding protocol specifications.
- We extend the protocol knowledge base presented in our previous work [40] and modify L* algorithm so that a protocol learner can incorporate the knowledge to compose the minimum number of necessary membership queries for hypothesis construction. This helps in drastically decreasing the number of wasted queries and learning more compact behavior models.
- We conduct extended case studies and prove that the proposed scheme not only infers updated protocols efficiently but also improves the performance of the L* algorithm in terms of the membership query count and compactness.

The rest of the article is organized as follows. Section 2 describes existing works and their limitations in detail. Section 3 presents our key requirements for the knowledge-based efficient protocol inference. Section 4 describes key component designs to overcome the existing works' limitations and accomplish an efficient inference of updated protocol syntax and behavior. Section 5 explains case study results for two protocol update scenarios. Section 6 discusses open questions about the current scheme. Finally, Section 7 provides a summary of our work and future works.

2 RELATED WORKS

This section presents our systematic survey on existing protocol interoperability solutions and additional applicable schemes to resolve incompatible protocol update issues. Then, we point out their limitations concerning the inference of updated protocol syntax and behavior at runtime.

2.1 Overview

Table 1 summarizes a landscape of related works on protocol heterogeneity as the uncertainty level of the target protocol increases. Some of the related works present novel schemes that solve both syntax and behavior together, but the others solve only one specific issue. It becomes more challenging to solve the heterogeneity problem as the target protocol's uncertainty level increases due to the absence of its specification. To achieve a successful interaction without knowing the specification, it is necessary to perform a non-trivial inference upon the heterogeneous protocol even if its success is not guaranteed. In addition, the uncertainty increases when not all syntax and behavior of involved protocols are semantically compatible and mismatches exist between them. Such a mismatch makes it not feasible to translate between the protocol messages or interfaces even if the specifications are given. A behavioral mismatch may lead an interaction to end up with a non-terminate state. Halting in a non-terminate state implies that at least one participating agent may consider that the interaction is still active, although it is not. Then, the agents should wait until a force quit condition (e.g., timeout error) is triggered.

2.2 Protocol Adaptation When Specifications Are Given: *Low-Mid Uncertainty Level*

2.2.1 Protocol Syntax Adaptation. The most primitive requirement for enabling an interaction, when the corresponding specification to an interaction target's protocol is given, is the translation between the interacting protocols' heterogeneous syntax. For effective translation, various dictionary-based adapters [11, 30, 34, 35] have been presented in the early stages. Once an adapter designer who knows both interacting protocols develops protocol translation logic, a proxy daemon is generated in each protocol agent for adapting its protocol syntax to that of the interaction target protocol. Designing translation logic becomes infeasible and dictionary-based adapters cannot even guarantee a successful translation, as too many interaction protocols are introduced and their syntax specifications become complex. It should then be automatically detected which messages or interfaces are semantically compatible and how to translate compatible ones while

Table 1. A Landscape of Related Works on Protocol Interoperability as the Uncertainty Level of the Target Protocol Increases

Heterogeneity Dimension	*Low Uncertainty Level*: Spec. Given & No Mismatch Exists	*Mid Uncertainty Level*: Spec. Given & Unidentified Mismatches Exist	*High Uncertainty Level*: Spec. Not Given
Protocol Syntax (Message/Interface)	• **Key questions:** "How can I translate my protocol to the target protocol?" • **Challenges:** Scalable syntax/interface translation • **Existing works:** Dictionary-based adaptation (1-to-1) [11, 30, 34, 35] & A common intermediary agent/representation (M-1-N) [8, 19, 20]	• **Key questions:** "How are they different?" & "How can I adapt one to another?" • **Challenges:** Matching similar interfaces with the minimum semantic loss • **Existing works:** Ontology-based compatibility inference [6, 7] & Constraint-based interface matching [4, 28]	• **Key questions:** "What does it look like?" • **Challenges** Successful inference for the syntax & Inference efficiency • **Existing works:** Knowledge-based extrapolation [18] & Trace-based automatic protocol reverse engineering [26, 45] & Protocol executable analysis [10, 21, 25]
Protocol Behavior	• **Key questions:** "How can I integrate both protocols' behaviors?" & "How can I handle mismatches?" • **Challenges:** Protocol behavior translation • **Existing works:** Protocol behavior translation & integration [9, 17, 27, 35]	• **Key questions:** "How can I mediate these behaviors to achieve the goal?" • **Challenges:** Efficient behavior mediation • **Existing works:** Iterative sink-free mediator computation [4, 6]	• **Key questions:** "How does it behave?" • **Challenges:** Successful and efficient inference for the protocol behavior • **Existing works:** Model-based protocol behavior learning with active automata learning [5, 15, 42, 44]

minimizing the semantic loss caused by syntactic mismatches. To identify semantics of protocol syntax and their compatibility, ontology-based matching techniques and Liskov substitution principle [24] have been leveraged [6, 7]. In addition, constraint-based interface matching schemes [4, 28] leverage business-level requirements to compute the optimal interface matching that fulfills the corresponding constraints to the requirements best.

2.2.2 Protocol Behavior Adaptation. When multiple pairs of semantically compatible messages/interfaces have been detected, it should be found in which order they need to be executed to perform an intended task (service discovery, authentication, etc.) successfully. In the early stages, various protocol behavior translation and integration schemes [9, 17, 27, 35] have been presented. When specifications of interacting protocol behaviors are given as finite automata, domain experts write how to merge the given behaviors in a dedicated description language. The description is then compiled, and a merged automata is generated to draw a federated behavior of the involved protocol agents. As the behavior models become too complex to write their federation logic manually, it is automated to compute the federated behavior. Iterative sink-free mediator computation schemes have been presented [4, 6] to avoid behavioral mismatches in the federated behavior. Their proposed algorithms iterate merging all possible pairs of semantically compatible messages/interfaces until its resulting federated automata accomplishes intended tasks and does not halt in a non-terminate state.

2.2.3 Limitations of Existing Works. Even if existing schemes effectively adapt a protocol to another, it is hard to assume that a specification of the interaction target protocol is given as a prior knowledge in realistic scenarios, which is the common assumption of the existing works. Especially in IoT environments where interaction protocols dynamically evolve, updated specifications cannot be given in advance. No dedicated adapter can be designed by the human experts without the updated specifications, and existing works cannot resolve the incompatible protocol update issue.

2.3 Protocol Adaptation When Specifications Are Not Given: *High Uncertainty Level*

For resolving the incompatible protocol update issue without the specifications as an *a priori* knowledge, it is necessary to infer the syntax and behavior of the updated protocol and adapt the legacy protocol to the inferred specification at runtime so that they can perform a federated behavior successfully. This section describes three types of existing schemes that we can leverage for the protocol inference without corresponding specifications to updated protocols and limitations of each type, respectively.

2.3.1 Automatic Protocol Reverse Engineering for Protocol Syntax Inference. **Automatic Protocol Reverse Engineering (APRE)** is applicable to infer a protocol syntax without its corresponding specification. Protocol message trace-based syntax extraction [26, 45] is a popular APRE approach. It collects many message traces from the network for a specific protocol and identifies byte sequences that frequently appear in the protocol messages as the protocol's keyword candidates. Then, it computes relative positions among the identified keywords and models their transition probabilities as a finite state machine. Based on the state machine, it infers the syntax of the target protocol and reconstructs a complete message corresponding to the inferred syntax. Syntax extraction using a protocol agent's executable [10, 21, 25] is another APRE approach. When an executable of the target protocol's agent is given, an outbound message structure of separate message fields is reconstructed by reversing its output message buffer. Then, each field's semantics is inferred by tracing other memory buffers from which values of the identified message fields come.

Even though APRE approaches are proved to be effective, they are not applicable to runtime scenarios in which interaction opportunities between IoT devices are not permanent. In the case of the syntax extraction schemes using protocol message traces, it is hard to collect enough network traces to extract frequent keyword patterns before an interaction opportunity disappears. Furthermore, much effort and resources are required for network monitoring, packet aggregations, and structure learning. When it comes to the syntax extraction schemes using a protocol agent's executable, acquiring a target protocol's executable is not feasible in most IoT environments due to security concerns. Therefore, we need an efficient protocol syntax inference scheme that does not require a large number of protocol artifacts and completes the inference shortly enough to maintain interaction opportunities.

2.3.2 Universal Semantic Communication. Juba [18] presents *universal semantic communication* to infer an unknown protocol's syntax efficiently without any artifacts or specification of the protocol. In his proposal, an interactive Turing machine iterates an adaption of a legacy protocol's message until it is translated into the same statement of the unknown target protocol and its semantics is understood correctly. The interactive Turing machine leverages a protocol knowledge base that provides a useful bias to infer the most probable adaptation method, which prevents a random message adaptation and completes the adaptation procedure quickly. Exp3 algorithm [3] is incorporated into the interactive Turing machine to optimize its message adaptation policy and establish an upper bound on the iteration count. For evaluation, he applies the proposed scheme to a protocol update scenario from IPv4 to IPv6 and verifies that an IPv4 message is successfully

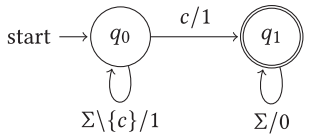


Fig. 1. An example hypothesis model $\mathcal{H}_1$.

translated into the same statement in IPv6 agent with a bounded iteration. However, Juba [18] provides only a computational model and its proof without any realistic architectural view. For example, it is not presented with how to incorporate the knowledge base into the interactive Turing machine. In addition, the Exp3 algorithm does not apply to our problem because our challenge is not to determine an optimized message adaptation policy.

2.3.3 Active Automata Learning for Protocol Behavior Inference. Compared to the protocol syntax inference problem, protocol behavior inference has been studied more often and an active automata learning algorithm, Angluin's L^* *algorithm* [2], has become a lingua franca. Based on the L* algorithm, many research works learn a target protocol agent's behavior when its interaction syntax is given [5, 15, 42, 44]. L* algorithm works as an alternation of the exploration phase and testing phase. In an exploration phase, a protocol learner constructs a hypothesis on the target protocol behavior by transmitting a series of input messages complying with the identified syntax and observing the target protocol agent's reactions. Each transmission of an input message is called a *membership query*, and observed reactions are recorded in a dedicated data structure, *observation table*. An observation table represents a mapping function $Obs(\mathcal{U}, \mathcal{D}) : \mathcal{U} \times \mathcal{D} \rightarrow \Omega$, where $\mathcal{U} = S_p \cup L_p$ is a set of prefixes and $\mathcal{D}$ is a set of suffixes. From the perspective of the protocol agent under learning, a prefix represents an input message sequence that gets the protocol agent to the current state, and a suffix represents an input message sequence transmitted by the protocol behavior learner. Ω is an output of the protocol agent under learning when input is given at a certain state. The S_p elements are explicitly constructed states in the Mealy machine, and those in L_p are possible future behaviors at the resulting state of the prefixes in S_p. Elements in L_p are not added to the hypothesis Mealy machine as an explicit state because they belong to equivalence classes to those in S_p. More details about the algorithm and observation table are described elsewhere [2, 41]. When enough membership queries are tried to derive a complete behavior hypothesis (i.e., an observation table is *closed*), the protocol learner proceeds a testing phase by running an *equivalence testing*. The equivalence testing evaluates whether the constructed hypothesis conforms to the target protocol agent's actual behavior. If any violation is found, another exploration phase starts. Otherwise, the learning procedure terminates, and the hypothesis becomes the learning result. Table 2 and Figure 1 show an example of a closed observation table and its output hypothesis model.

Despite its popularity and effectiveness, L* algorithm suffers from inefficiency caused by a large number of membership queries. L* algorithm was originally designed as a generic algorithm for learning a regular behavior model covering corresponding behaviors to all input actions of the target protocol agent (i.e., output messages of the protocol learner). Therefore, a protocol learner (i.e., legacy protocol agent) tries membership queries for all input actions regardless of whether the input action is declared in the legacy protocol agent's specification. If an input action is not declared in the specification but tried as a membership query, that membership query is unnecessarily wasted for observing an input-output action pair that is not supposed to happen. Such wasted membership queries decrease the overall behavior learning procedure's efficiency, and the impact of such degradation becomes more critical in a runtime IoT interaction scenario. Therefore, we need to devise a more efficient protocol learning mechanism that prevents unnecessarily wasted

Table 2. A Closed Observation Table for a Hypothesis Model $\mathcal{H}_1$ in Figure 1

		$\mathcal{D}$			
		a	b	c	d
S_p	ϵ	1	1	0	1
	c	0	0	0	0
L_p	a	1	1	0	1
	b	1	1	0	1
	d	1	1	0	1
	$c \cdot a$	0	0	0	0
	$c \cdot b$	0	0	0	0
	$c \cdot c$	0	0	0	0
	$c \cdot d$	0	0	0	0

membership queries and eventually completes a learning procedure before a runtime interaction opportunity disappears.

3 DESIGN CONSIDERATIONS

To infer an updated protocol's syntax and behavior shortly and adapt the legacy protocol to it without requiring corresponding specifications and a large number of protocol artifacts, we need an effective bias that helps to make a more probable inference than random guesses. This section clarifies our design goal and describes key requirements of designing and leveraging such bias, *protocol knowledge base*, for efficient protocol inference at runtime.

3.1 Definition of Meaningful Interactions

The ultimate goal of the updated protocol inference and legacy protocol adaptation is to enable an updated protocol agent (i.e., message receiver) to understand the meaning of a received message transmitted by a legacy protocol agent (i.e., message sender). Based on the understanding, they can perform a federated behavior that accomplishes an intended task. However, our design goal and corresponding complexity should vary depending on how we define a successful inference. Juba [18] calls an inference successful when it translates the sender's statement into the same statement in the receiver's language and they can achieve an effect through the interaction. Then, he defines the interaction *meaningful*. He devises a knowledge-based interactive Turing machine that adapts the legacy protocol message to accomplish precise translation. Case study results show that meaningful interaction can be made by bounded iterations of message adaptation. However, the complexity of the inference becomes too high to be applied to a runtime interaction scenario due to the precise translation demand.

In this work, the early completion of protocol inference is the key requirement with regard to runtime interaction scenarios. Therefore, we need to refine Juba's definition of meaningful interaction and establish a practical design goal with an applicable complexity. For that purpose, we first relieve the strict condition of a successful inference, a precise translation between a legacy protocol and its updated version. In real interaction scenarios, a message receiver with an updated protocol can understand a message sender's intention with a legacy protocol even when a partially precise translation is performed. For example, reserved message fields for future uses or extended features are often ignored by message receivers. Invalid translations upon those fields do not necessarily

trigger a parsing error and disrupt a meaningful interaction. In addition, Juba's definition deals only with precise translation between two heterogeneous syntaxes, whereas both protocol syntax and behavior of interacting protocol agents should be compatible to make a meaningful interaction. Therefore, we need to refine his definition so that our design goal covers the accomplishment of a federated behavior as well.

Considering these two refinements, we define an interaction meaningful when (1) the receiver understands the intention of the received messages and (2) returns a non-error message that leads to a terminal state of the federated interaction and eventually achieves the intended effect. We hold this definition and aim to infer an updated protocol to enable meaningful interaction between incompatible versions as our design goal.

3.2 Knowledge Base Construction

Multiple past research works [16, 18] have leveraged the knowledge base as an effective bias that facilitates more probable inference than random guesses. The protocol knowledge base can be very effective in decreasing our investigation scope, which is the number of possible update candidates of the legacy protocol syntax and behavior. A protocol update is initiated when new requirements are made as a consensus. In protocol syntax, a protocol message comprises multiple fields, each of which contains the necessary information for a protocol functionality. When a new requirement is made, it either affects an existing functionality and its corresponding message fields or requires to encode a new functionality into the protocol message. Various protocol updates have been made across many interaction protocols and eventually revealed a lot of common patterns in such dynamics of requirement, functionality, and message fields. For example, less important message fields are removed, or their lengths are shortened to make a protocol more lightweight for resource-constrained devices. Furthermore, additional fields and their corresponding values are designed for authentication purposes to enhance the security aspect. This generic knowledge can provide us with a high-level insight that helps to extrapolate which probable changes can be made to a legacy protocol message. What we have learned from the past can be leveraged to predict the future.

The protocol behavior inference can leverage the knowledge base differently. L* algorithm has been proved effective for learning an unknown behavior of a target protocol. Therefore, rather than designing a new algorithm and its supporting data structures with a clean slate, it is more reasonable to exploit the knowledge base to add a useful bias to the L* algorithm so that we can improve its efficiency and finish a behavior learning procedure shortly enough to hold runtime interaction opportunities valid. The L* algorithm's goal is to build a regular model that can perfectly describe a software agent's behavior by observing its reactions (i.e., output messages) to all possible input messages. However, the pursuit of such *perfection* makes the algorithm waste more membership and equivalence queries than necessary and take a longer time to complete a learning procedure in our problem domain. A legacy protocol agent (i.e., message sender) needs to know how to exchange relevant messages to an intended task (e.g., service discovery, connection establishment.), whereas it does not need to know irrelevant message exchange sequences. Even in the relevant message exchange behavior, the legacy protocol agent does not need to learn reactions of the updated protocol agent (i.e., message receiver) to input message sequences that are not supposed to be transmitted by the sender agent itself. We can aim at achieving such *good enough* interoperability [38] as our design goal for protocol behavior learning since IoT interactions are mostly task-driven. Hence, we need to tune the goal of the L* algorithm from learning a perfect behavior model to learning a good enough behavior model by constructing a knowledge base that declares task-specific message exchange behaviors of legacy protocols and making the L* algorithm compose membership queries only for message sequences conforming to the declared knowledge.

3.3 Efficient Search for a Valid Syntax Adaptation Sequence

When a protocol is updated, it entails a sequence of updates across different message fields. For example, when SLP was updated from version 1 to version 2, eight message fields were updated (deletion, addition, length change, etc.). In other words, it is not enough to detect a single updated field, but we need to infer the whole sequence of field updates correctly for adapting the legacy protocol syntax and making it compatible with its updated version. In addition, the order of the identified updates is also critical. For example, when adapting the legacy protocol message by adding two new fields, adding them in a different order makes the adapted message different. Such two characteristics turn the protocol syntax update inference problem into detecting a right permutation of valid message field adaptation methods, and its complexity grows exponentially with the addition of adaptation method candidates. Therefore, we need to design an efficient search algorithm and data structures for a valid syntax adaptation sequence for the early achievement of meaningful interaction between incompatible protocol versions.

3.4 Knowledge-Based Efficient Behavior Learning

To make L* algorithm compose membership queries only for message sequences conforming to task-specific legacy protocol behavior, we first need to design a knowledge model representing a protocol agent's dynamic behavior; possible actions in each state and corresponding state transitions to the actions. L* algorithm should then be modified to interactively refer to the declared knowledge for composing valid membership queries only for the current protocol task. Not all prefix-suffix pairs in an observation table are guaranteed to be composed as membership queries. When a learner performs such a selective membership querying in an exploration phase, the learner may not completely fill the observation table with membership query results and initiate the closedness evaluation. As a result, a hypothesis upon the target protocol agent's behavior cannot be constructed either. Hence, an observation table's closedness evaluation procedure also should be modified to prevent such an issue so that the learner can evaluate an incomplete table with non-answered prefix-suffix pairs and construct its corresponding hypothesis model.

4 PROPOSED SCHEME

This section presents the proposed architecture and its key components that accomplish the knowledge-based iterative probing mechanism to infer updated protocol syntax and behavior.

4.1 System Architecture Overview

Figure 2 shows the proposed scheme's system architecture. Two different parties are involved with the proposed scheme's interaction scenario: a user agent (i.e., interaction requester) and a service agent (i.e., target IoT device). A user agent brings a legacy protocol and runs our proposed scheme to interact with a service agent running an updated version of the legacy protocol. When a user agent tries to interact with an updated protocol of a service agent, it infers the updated protocol and adapts its legacy protocol via three phases. In the initialization phase (denoted by 0), a user agent starts and loads the protocol knowledge base. After loading is done, **Protocol Syntax Learner (PSL)** starts a syntax learning phase (denoted by 1) by making a query to **Knowledge Base (KB)** for acquiring a sequence of the most probable adaptation methods of a legacy protocol message. The returned sequence is forwarded to **Probe Message Transceiver (PMT)** and **Probe Message Composer (PMC)** so that these two modules apply the sequence to the legacy protocol message. The user agent transmits the message to the target service agent and forwards its reaction to PSL to evaluate whether the adapted message is compatible with the updated protocol. If the reaction is a non-error reply message with an expected result, we conclude that the updated syntax is

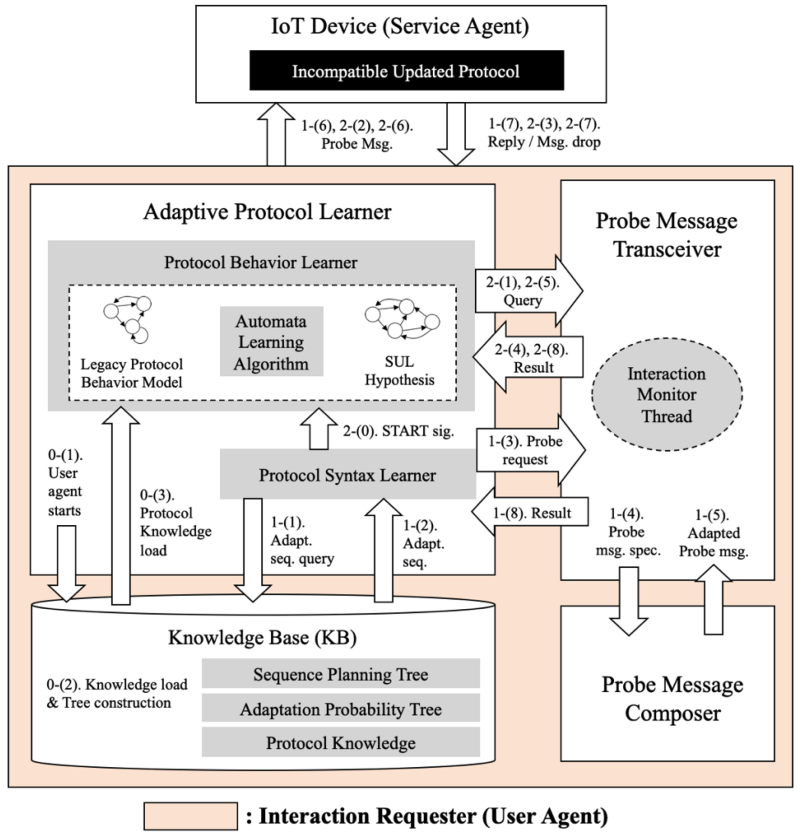


Fig. 2. A system architecture overview.

correctly inferred and a protocol behavior learning phase (denoted by 2) starts. In this phase, the **Protocol Behavior Learner (PBL)** runs a modified L* algorithm that refers to the legacy protocol behavior model to selectively compose valid membership queries. PBL goes through the same learning procedure with the original L* algorithm, an alternation of the exploration and testing phases, other than the membership query selection procedure. The user agent and service agent can eventually make a meaningful interaction and complete an intended task together when PBL successfully infers the service agent's updated protocol behavior.

4.2 Protocol Knowledge Base Construction

The protocol knowledge base should be leveraged as an effective bias to make more probable guesses than random trials and infer an updated protocol's syntax and behavior shortly with. For that purpose, the knowledge base should describe relationships between protocol requirements, functionalities, and message fields, and task-specific message exchange behaviors of legacy protocols. This work extends our prior ontology design [40] to describe not only syntax updates but also the behavior of IoT interaction protocols (Figure 3).

Ontology design for protocol syntax defines six core classes and predicates between them, describing important dynamics relevant to protocol syntax updates (details can be found in our previous work [40]). When representing a protocol's behavior, *Mealy machine* is the most widely used method because a protocol agent is a variant of a finite state machine that exchanges input and output messages according to its state. Therefore, the knowledge structure for protocol be-

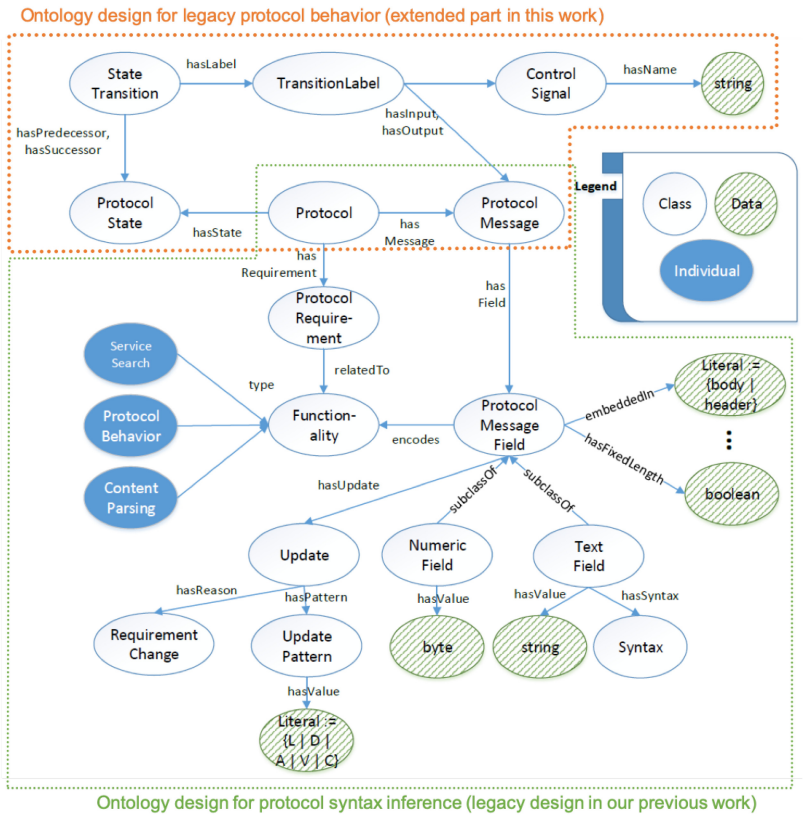


Fig. 3. Ontology design for IoT interaction protocol syntax and behavior.

havior should be able to describe a protocol's message exchanging behavior in terms of the Mealy machine. We define three additional core classes, {*ProtocolState, StateTransition, TransitionLabel*}, to fulfill such requirement. Then, we define necessary predicates between the core classes to represent a protocol's Mealy machine: {*hasState, hasPredecessor, hasSuccessor, hasLabel, hasInput, hasOutput*}. When instantiated statements for such ontology triples are written by a domain expert and stored in KB, they are transformed to an active knowledge representing a Mealy machine of a legacy protocol's behavior and loaded to PBL in the initialization phase.

For example, when an MQTT client's behavior for connection establishment task is given as shown in Figure 4, it is stored in protocol behavior learner as a transformed Mealy machine in Figure 5. The MQTT client starts with an initial state, q_0. When an *INIT* signal is given, the client transmits a *CONNECT()* message and performs a state transition from q_0 to q_1. If an MQTT broker returns a *CONNACK(0)* reply message with a return code *zero*, the client terminates the connection establishment procedure and generates an *ESTABLISHED* signal. When terminating the connection establishment procedure, the client's state transits from q_1 to q_0.

4.3 Protocol Syntax Inference

The protocol syntax learning phase starts when the protocol knowledge base is ready. PSL computes sequences of protocol message field updates and applies the most probable sequence to adapt a legacy message by identifying possible field updates and computing their probabilities. This

- {*MQTT - hasState - InitialState*}, where *InitialState* indicates q_0 in Figure 5
- {*MQTT - hasState - WaitState*}, where *WaitState* indicates q_1 in Figure 5
- {*Transition1 - hasPredecessor - InitialState*}
- {*Transition1 - hasSuccessor - WaitState*}
- {*Transition2 - hasPredecessor - WaitState*}
- {*Transition2 - hasSuccessor - InitialState*}
- {*CONNACK_SUCCESS - hasField - ReturnCode_SUCCESS*}
- {*ReturnCode_SUCCESS - hasValue - 0*}
- {*CONNACK_FAIL - hasField - ReturnCode_FAIL*}
- {*ReturnCode_FAIL - hasValue - N/Z*}
- {*Transition1 - hasLabel - Transition1_Label1*}
- {*Transition2 - hasLabel - Transition2_Label1*}
- {*Transition2 - hasLabel - Transition2_Label2*}
- {*Transition1_Label1 - hasInput - INIT*}
- {*Transition1_Label1 - hasOutput - CONNECT*}
- {*Transition2_Label1 - hasInput - CONNACK_SUCCESS*}
- {*Transition2_Label1 - hasOutput - ESTABLISHED*}
- {*Transition2_Label2 - hasInput - CONNACK_FAIL*}
- {*Transition2_Label2 - hasOutput - FAIL*}

Fig. 4. Instantiated ontology statements to describe connection establishment behavior of MQTT clients.

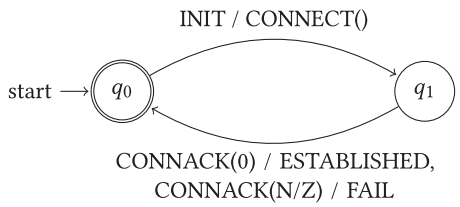


Fig. 5. A Mealy machine for MQTT connection establishment mechanism by an MQTT client.

section briefly describes the syntax learning phase, and details about the following components can be found in our previous work [40].

4.3.1 Update Probability Tree. A tree structure, called *update probability tree*, is designed to identify possible message field updates and compute their probabilities. A tree comprises multiple nodes with five different depth levels having different semantics and downward edges connecting a parent node to its children. An update probability tree has only one root node (depth 1). A tree node in each depth represents a protocol functionality (depth 2), a possible requirement change (depth 3), a related field to the requirement change (depth 4), and an update type (depth 5), respectively. A path from the root node down to a leaf node (depth 5) represents a probable update of a protocol message field and its underlying rationale. Along with the path, weight values are computed using the update history of the child nodes and assigned onto the edges to indicate the relative probabilities of a child node to the other siblings. A relative probability of each possible update represented by a path is computed as a product of the four weight values.

4.3.2 Adaptation Sequence Planning Tree. When relative probabilities of all update method candidates are given, a sequence of the most probable candidates should be iteratively computed until a valid one is found. A tree-based data structure, called *adaptation sequence planning tree*, and its traversal algorithm efficiently search for the most probable sequence. A tree comprises multiple layers of nodes representing update method candidates. Considering that achieving

efficient inference and its early termination is the most important requirement, we adopt various principles such as tree node sorting, memoization-based dynamic programming, and heuristics-based tree node pruning to minimize our search algorithm's complexity and its memory usage. An optimized tree model and its traversal algorithm repeatedly compute a sequence of update method candidates with the highest probability.

4.3.3 Bit-Level Message Adaptation. When a sequence of update method candidates is computed, PMC takes it to perform a bit-level adaptation for the legacy protocol message and transmit the adapted message to the service agent. For field addition, PMC adds a new field that is filled by zero or a well-known value if the field semantics has a well-established standard (e.g., Character Encoding). The new field's length is determined based on statistics of the same type of message field in the knowledge base. For field deletion, PMC deletes the existing field. For vocabulary change, PMC currently adopts *no-change* policy because changing vocabulary requires external knowledge sources, but the current version of the proposed scheme does not leverage any of them. For length change, PMC leverages the statistics of the knowledge base as we do for field addition. According to the numeric field's semantics that can be acquired from KB, PMC increases or decreases the field's value for numeric value change.

4.4 Protocol Behavior Inference

4.4.1 Overview of Protocol Behavior Inference. When the adapted legacy protocol becomes compatible with the updated version in terms of protocol syntax, PSL initiates the protocol behavior learning phase by informing PBL of the updated protocol syntax and triggering a START signal. Figure 6 illustrates the overall interactions between key components of protocol behavior learner. It is the key requirement for efficient updated behavior learning to prevent the membership query selection mechanism from composing queries for prefix message sequences that are not supposed to be transmitted by the legacy protocol agent. Our modified L* algorithm fulfills this requirement by checking, whenever composing a membership query, the legacy protocol's behavior model that is loaded from KB in the initialization phase.

We put a *Mapper* component and make it perform *Abstraction* devised by Aarts [1] to make the behavior inference effective. Many message fields have a numeric value that varies arbitrarily or depending on the previous interaction traces. For example, it is common to have reply messages that contain a return code implying the result of the request message and its reason. If we consider such varying numeric values as separate input or output alphabets, the alphabets' size becomes almost infinite, and learning a behavior model incorporating those alphabets becomes infeasible. We prevent such an infinite number of alphabets by establishing a fixed number of abstract variables corresponding to the variable numeric values. For example, a reason code *zero* in *CONNACK* message implies that the connection is established successfully. However, a non-zero reason code implies that a connection establishment fails due to some reasons. We abstract the field values by establishing only two abstract values, *zero* and *N/Z*, which implies *SUCCESS* and *FAIL*. Such abstraction is performed based on a predefined abstraction mapping table that a domain expert writes before the whole system runs.

4.4.2 L^* Algorithm Modification for Knowledge Incorporation. The legacy protocol behavior model (Section 4.2) is incorporated into the protocol behavior learning procedure by modifying the original membership query selection procedure in the L* algorithm. L* algorithm is separated into three sub algorithms [41]: *Close Table*, *Process counterexample*, and *L^**. *Close Table* algorithm constructs a hypothesis model $\mathcal{H}$ for a target protocol agent by repeatedly composing membership queries and filling an observation table with query outputs until the observation table fulfills the closedness condition. Then, L^* algorithm performs an equivalence testing to evaluate

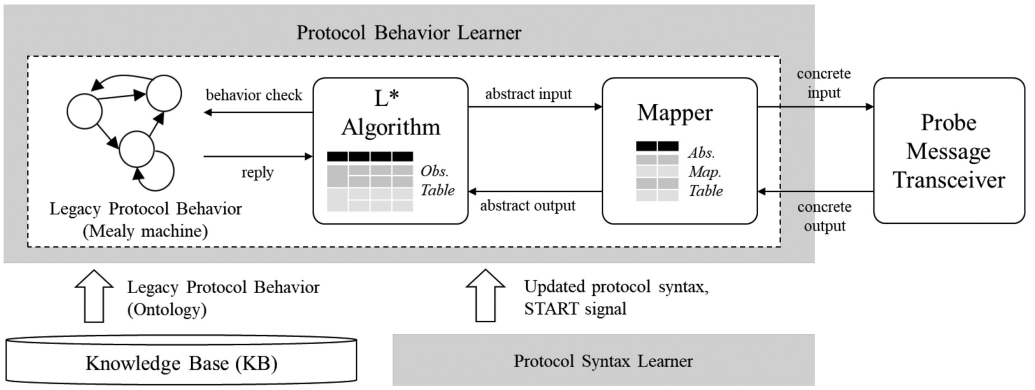


Fig. 6. An overview of protocol behavior learner and its interactions with related components.

the constructed hypothesis. If no counterexample is returned, L^* returns the hypothesis $\mathcal{H}$ and terminates. Otherwise, it passes the counterexample to the *Process counterexample* algorithm so that a distinguishing suffix is added to $\mathcal{D}$, and the next closed hypothesis model is constructed. To prevent many suffixes in a suffix-closed set conforming to the counterexample from entailing a large number of membership queries, we leverage the suffix selection algorithm proposed by Steffen et al. [41] that returns only one distinguishing suffix with binary search. Algorithms 1 and 2 present the pseudo-codes of L^* and *Close Table* algorithms, as shown in the work of Steffen et al. [41], describing how three algorithms work together and which parts are modified to incorporate the knowledge base into the algorithm. Section 2.3.3 explains how the backbone of the L* algorithm, prefixes, suffixes, and an observation table, work together.

Considering how three sub algorithms perform the hypothesis construction and its evaluation, the *Close Table* algorithm needs to be modified for improving the efficiency of membership query composition. Therefore, the knowledge of the legacy protocol behavior is integrated into L^* *algorithm* by modifying a condition for membership query composition in the *Close Table* algorithm. Algorithm 2 shows the modified *Close Table* algorithm. In line 2, one extra condition, $Legacy(u, v) = true$, is added at the last so that a membership query is transmitted only when the legacy protocol performs a corresponding behavior to the given prefix u and suffix v. When $Legacy(u, v)$ function is called, PBL retrieves the Mealy machine representation of the legacy protocol behavior and evaluates whether a corresponding message exchange behavior to the input message sequence (u, v) is defined in the specification. If the evaluation function returns true, it implies that the given input message sequence conforms to the specification, and the user agent can transmit the messages to the service agent. If the evaluation function returns false, it implies that the given message sequence violates the specification, and the algorithm should skip the membership query corresponding to the pair of (u, v). By interfacing the knowledge and the *Close Table* algorithm with this condition, the overall performance of the protocol behavior learning procedure can be drastically improved in terms of membership query count and computing resources to perform the queries.

Allowing to skip membership queries can make an observation table have an empty row if it has a prefix message sequence that is not supposed to be sent by the user agent. Considering that the empty row indicates that its prefix is not feasible by the user agent, the prefix should not be added to S_p because prefixes in S_p are realized as feasible states in the hypothesis Mealy machine. Such false state addition can end up with an internal error that asks the user agent to transmit a sequence of messages that do not conform to its specification. We prevent it by modifying the

ALGORITHM 1: L*

Input : A set of inputs Σ
Output : A Mealy machine $\mathcal{H} = \langle S, s_0, \Sigma, \Omega, \delta, \lambda \rangle$

1 **Loop**
2 **do**
3 construct $\mathcal{H}$ by Algorithm *Knowledge-based Close Table*
4 check semantic suffix-closedness of observation table
5 **if** *a new suffix d is required to fulfill semantic suffix-closedness* **then**
6 $\mathcal{D} := \mathcal{D} \cup \{d\}$
7 **end**
8 **until** *semantic suffix-closedness is established*;
9 $\bar{c} := eq(\mathcal{H})$
10 **if** $\bar{c}$='ok' **then**
11 **return** $\mathcal{H}$
12 **end**
13 **else**
14 get suffix d from $\bar{c}$ by Algorithm *Process counterexample*
15 $\mathcal{D} := \mathcal{D} \cup \{d\}$
16 **end**
17 **EndLoop**

ALGORITHM 2: Knowledge-based Close Table

Input : A set of inputs Σ, A legacy protocol behavior model $Legacy(\mathcal{U}, \mathcal{D})$
Output : A hypothesis $\mathcal{H} = \langle S, s_0, \Sigma, \Omega, \delta, \lambda \rangle$

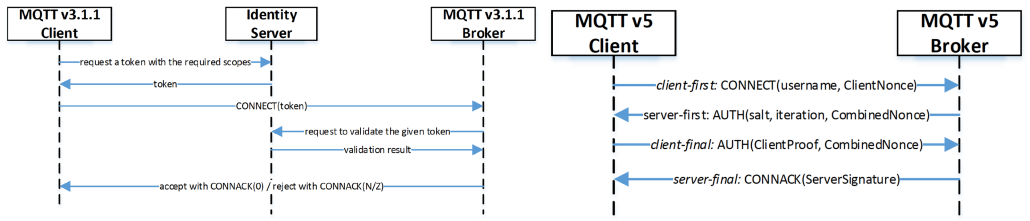
1 **do**
2 fill table by mq(uv) for all pairs $u \in \mathcal{U}$ and $v \in \mathcal{D}$, where $Obs(u, v) = \phi$ and $Legacy(u, v) = true$.
3 **if** $\exists u \in L_p \forall u' \in S_p.\ Obs_u \neq Obs_{u'}$ *and* $Obs_u \neq \phi$ **then**
4 $S_p := S_p \cup \{u\}$
5 $L_p := (L_p \cap \{u\}) \cup \{u\alpha | \alpha \in \Sigma\}$
6 **end**
7 **until** *closedness is established*;
8 **return** hypothesis $\mathcal{H}$ for $Obs(\mathcal{U}, \mathcal{D})$

condition that is evaluated when moving a prefix from L_p to S_p (line 3 in Algorithm 2). By adding an extra condition, $Obs_u \neq \phi$, empty rows in the observation table are skipped when evaluating prefixes that are to be moved from L_p to S_p.

For equivalence testing (line 9 in Algorithm 1) concerning real protocol scenarios, it is not possible to assume an equivalence oracle that is aware of all protocol behaviors and returns a result (i.e., 'ok' signal or a counterexample) of the testing. This issue is well known in various application domains of the L* algorithm. Therefore, the guided random walk testing [23] is used instead of assuming an oracle. In the random walk testing, many membership queries are finitely generated based on random walks over the constructed hypothesis model. If any membership query detects a behavioral violation, it is taken as a counterexample, and an exploration phase starts to build a refined hypothesis.

5 EVALUATION

We conduct two case studies with update scenarios of famous IoT interaction protocols to evaluate our proposed scheme. In the first case study, a user agent (i.e., MQTT client) bringing MQTT



(a) Authentication procedure in MQTT version 3.1.1 with a third-party authority, *identity Server*

(b) Authentication procedure in MQTT version 5

Fig. 7. Authentication procedure update from MQTT version 3.1.1 to MQTT version 5.

version 3.1.1 tries to establish a connection to a service agent (i.e., MQTT broker) using MQTT version 5 by sending a CONNECT message. Since the CONNECT message becomes incompatible in version 5, it is a good target with which to evaluate our proposed PSL. In addition, its authentication behavior does not support backward compatibility in version 5 (Figure 7). In MQTT version 3.1.1, a dedicated third-party authentication authority is involved in authenticating the client and broker. Only when the token transmitted by the MQTT client is valid, the broker accepts the connection establishment request. However, in MQTT version 5, an MQTT client and a broker directly exchange authentication credentials to authenticate each other without help from a third-party authority. The authentication credentials are exchanged via *CONNECT - AUTH - AUTH - CONNACK* message sequence. Due to this behavioral difference, an MQTT client with version 3.1.1 cannot authenticate itself and an MQTT broker with version 5 even if the updated message structure is inferred successfully. PBL running on the MQTT client resolves such behavioral differences and enables authentication between two different versions by inferring the broker's updated authentication procedure. We leverage this scenario to evaluate how effectively our proposed PBL improves the L* algorithm's efficiency.

In the second case study, a user agent brings **SLP version 1 (SLPv1)** and tries to interact with a service agent using SLP version 2 (SLPv2). Since the message syntax of SLPv2 is not compatible with that of SLPv1, they cannot exchange a service request and reply message that is essential to discover each other and share service information. The user agent enables meaningful interaction by running the proposed scheme and repeatedly adapting its service request message until a non-error reply message is returned. Since the SLP update entails a compatibility issue only for protocol syntax, protocol behavior learning is not performed in this case. We assume a peer-to-peer interaction between the user agent and service agent without a Directory Agent (DA) to focus on two communicating agents while excluding other parties.

5.1 Implementation and Experiment Setup

We implement the proposed scheme in Java language[1] and leverage an open source library, LearnLib [36], for L* algorithm implementation. We implement MQTT and SLP agents using open source projects such as Moquette [29], Paho [12], and jSLP [37]. Since MQTT version 5 has been recently published, there is no stable release complying with the new version. Therefore, we modify the existing MQTT version 3 library and its underlying Netty framework [31] to implement an MQTT broker with version 5. In addition, since the current SLP libraries comply with version 2 only, we modify the jSLP library to implement the user agent with SLPv1. We leverage Raspberry Pi 3 model B to run the proposed scheme and protocol agents on IoT devices, considering that it has

[1]Github repository URL: https://github.com/HeesukSon/sem2bit.

Table 3. Implementation and Experiment Setups

Setup	Description
Programming language	Java
Target machine	Raspberry Pi 3 model B (1.2-GHz processor, 1 GB of RAM)
SLP implementation	jSLP [37] library
MQTT implementation	Moquette MQTT broker [29] and Paho MQTT client [12] libraries
Protocol behavior learning	LearnLib [36]
Performance profiling	JProfiler [13]

been widely used for various IoT applications. Table 3 shows a summary of the implementation and experiment setups.

5.2 Evaluation Metrics

Through the case studies, we need to evaluate whether the proposed PSL successfully infers the updated protocol syntax and its inference terminates quickly enough to maintain a user's intention to interact with nearby devices. For that purpose, we first measure how many iterations are required and how long it takes until an adapted legacy protocol message is compatible with its updated version and the service agent understands the user agent's intention (i.e., iteration count and elapsed time). Then, we investigate the accuracy of the inference in terms of precision, recall, and semantic loss. We can measure those metrics by comparing conducted adaptation methods on the legacy protocol message fields to the updated protocol message syntax. The adaptation methods here include 'still' actions that leave the legacy protocol message field unchanged. Semantic loss tells us how the inference-based message adaptation affects the semantics that is supposed to be delivered to the service agent. This metric is used to get an insight into how much of semantic incompatibility is tolerable for a meaningful interaction regarding protocol syntax. Semantic loss is computed as shown in Equation (1).

$$SemanticLoss = \frac{Number\ of\ fields\ which\ lost\ semantics\ from\ the\ receiver's\ perspective}{Number\ of\ fields\ to\ be\ evaluated\ by\ the\ receiver} \quad (1)$$

However, it is also important to verify that the proposed syntax learner is lightweight and resource-constrained IoT devices can afford to run it. Therefore, it is measured how much memory overhead the proposed syntax learner causes.

The number of membership and equivalence queries performed by the L* algorithm has been widely used as its performance metric. This metric is used to show the extent to which the proposed behavior learner improves the original L* algorithm's efficiency. It was also argued that existing works based on the original L* algorithm perform membership queries that do not conform to the legacy protocol agent's behavior and learn more complex Mealy machines than necessary. It is important to learn fewer behaviors as long as the learning result does not violate the updated protocol behavior because the more effort is required to learn the more complex model. We define a metric called *Compactness* to measure how fewer behaviors are learned and how close the learned behavior model is to the minimum canonical Mealy machine of the updated protocol behavior. Compactness is computed as shown in Equation (2) using the minimum canonical Mealy machine and the learning result.

$$Compactness = \frac{Number\ of\ input-output\ pairs\ in\ the\ minimum\ canonical\ Mealy\ machine}{Number\ of\ input-output\ pairs\ in\ the\ learned\ Mealy\ machine} \quad (2)$$

Table 4. Experiment Results for MQTT Message Syntax Inference

Most Probable AdaptSeq Length	Iteration Count	Elapsed Time (ms)	Precision	Recall	Semantic Loss	Memory Usage (MB)
3	4	533	0.909	1	0	8.65

An input-output pair indicates an output behavior instance corresponding to the given input. The higher a compactness value is, the more faithful behavior model an algorithm learns. Since the L* algorithm guarantees that the final observation table's closedness is established, the learned Mealy machine always has more input-output pairs than the minimum canonical Mealy machine. Therefore, a numeric value between 0 and 1 is assigned to compactness.

5.3 MQTT Case Study

This section presents experiment results for syntax and behavior inference in the MQTT update scenario. First, we analyze the efficiency and overhead of the protocol syntax inference. We analyze the membership query count and compactness of the proposed protocol behavior learner for the updated behavior inference.

5.3.1 Protocol Syntax Inference. We measure the proposed scheme's efficiency and accuracy when the service agent understands an adapted message and returns a non-error reply message. Table 4 shows the experiment results. The most probable sequence length in MQTT case is 3, and its corresponding elapsed time is 533 ms. According to the response time analysis [32], a response time shorter than 1 second is regarded as a seamless response by users. Considering that MQTT often requires near-real-time data exchanges, the proposed scheme helps MQTT agents with incompatible versions to achieve such performance requirements. In terms of memory usage, we measure heap and non-heap memory usages after a garbage collection and add them to represent our proposed scheme's actual memory usage. The result shows that the memory overhead in MQTT case is 8.65 MB. Recent IoT devices that are designed to interact with each other, not just to sense its surrounding environment, have at least 1 GB of RAM. Hence, we conclude that our proposed scheme is lightweight and various IoT devices can run it without a memory shortage problem. Detail evaluation results of the syntax inference are shown in our previous work [40].

5.3.2 Protocol Behavior Inference. It is necessary to make some assumptions on critical factors that may distract the experiment's focus on behavior learning performance. First, we assume that all MQTT agents and brokers in versions 3.1.1 and 5 are using *SCRAM* (*Salted Challenge Response Authentication Mechanism*) for authentication. By assuming this common authentication method, we can exclude possibilities that authentication fails due to the use of different methods. This assumption is reasonable because SCRAM is a widely used SASL authentication method as a standard. Second, we assume that authentication credentials are agreed upon in advance and never become invalid. We can exclude possibilities that authentication fails due to invalid authentication credentials or their computation results by making this assumption. Last, we assume that incompatible protocol message syntax issues have been solved before a protocol behavior learning procedure starts.

Before initiating a protocol behavior learning procedure, the learner should have a behavior model of the legacy protocol agent with regard to the given task (i.e., authenticated connection establishment) in advance so that our modified L* algorithm interactively refers to it. Figure 8 shows our Mealy machine design that describes a connection establishment behavior of an MQTT client with version 3.1.1. Since we assume that incompatible message syntax is all resolved in advance, state q_1 takes input and output alphabets from $\{AUTH(0), AUTH(18)\}$ on its self-loop in this experiment. We cope with finitely many reason code values in *CONNACK* message by abstracting

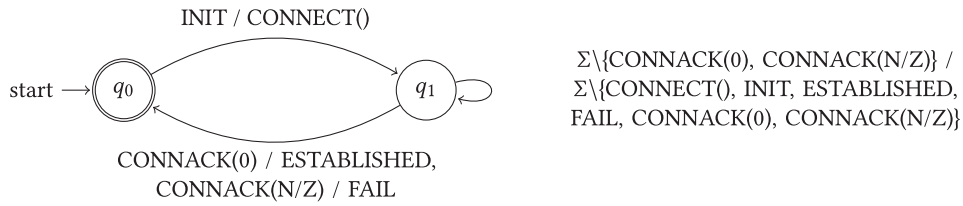


Fig. 8. Authentication behavior model of the MQTT version 3.1.1 client in Mealy machine representation.

Table 5. An Observation Table Filled by Membership Queries at the First Iteration Round of Output Word Filling Procedure

		$\mathcal{D}$		
		$CONNECT()$	$AUTH(0)$	$AUTH(18)$
S_p	ϵ	$AUTH(18)$	–	–
L_p	$CONNECT()$	–	$CONNACK(N/Z)$	$CONNACK(0)$
	$AUTH(18)$	–	–	–
	$AUTH(0)$	–	–	–

the reason code to either 0 or *N/Z* using the mapper. In addition, we assume that authentication credentials are agreed upon in advance and never become invalid. Such an assumption prevents authentication credentials from affecting the agents' future behaviors, which means we do not have to keep them as a variable contained in the alphabet. Therefore, we exclude authentication credentials from the set of alphabets in the abstraction procedure.

When PBL starts, an empty word ϵ is the only prefix in S_p, an initial state q_0 in the hypothesis model. Since it has {*CONNECT(), AUTH(0), AUTH(18)*} as the output alphabet at state q_0, the observation table has a distinguishing suffix set, $\mathcal{D}$={*CONNECT(), AUTH(0), AUTH(18)*}. Since L_p is one-letter continuation of prefixes in S_p, the table has L_p={*CONNECT(), AUTH(0), AUTH(18)*}. Table 5 shows the observation table whose output words are filled by membership queries, but rows in the upper and bottom parts are not compared and closedness is not evaluated yet. Since *CONNECT()* is the only output alphabet at the initial state q_0 of the legacy protocol behavior model, the output word *AUTH(18)* corresponding to $\epsilon + CONNECT()$ is filled but the other output words corresponding to ϵ + $AUTH(0)$ and ϵ + $AUTH(18)$ stay empty (i.e., denoted '–' in the table). When it comes to prefixes in L_p, since *CONNECT()* is the only output alphabet at state q_0, *AUTH(18)* and *AUTH(0)* cannot have any suffixes. In addition, *CONNECT()* message cannot be sent consecutively by the user agent. Therefore, only two output words out of nine can be filled up. With the original *Close Table* algorithm, it is supposed that 12 membership queries are tried and their corresponding output words are recorded in the first iteration. Compared to that, the membership query count of our proposed scheme decreases down to a quarter.

Since the row Obs_ϵ is different from $Obs_{CONNECT()}$ and $Obs_{CONNECT()}$ is not empty, our modified algorithm moves *CONNECT()* in L_p and its corresponding row $Obs_{CONNECT()}$ to the upper part. Then, one-letter continuations of *CONNECT()* are added to L_p and corresponding output words to them are filled via membership queries. Table 6 shows the result observation table after this iteration round. Since $CONNECT()$ message can be sent by the user agent only at the initial state and after receiving a $CONNACK()$ message, no membership query is composed for a row $Obs_{CONNECT()CONNECT()}$. For remaining prefixes, $CONNECT()AUTH(0)$ and $CONNECT()AUTH(18)$, their output words should be equivalent to those of the initial state according to the legacy protocol client behavior in Figure 8. Therefore, only a suffix $CONNECT()$ is tried for membership queries and their corresponding output words are recorded in the table.

Table 6. An Observation Table Filled by Membership Queries at the Second Iteration Round of Output Word Filling Procedure

		$\mathcal{D}$		
		$CONNECT()$	$AUTH(0)$	$AUTH(18)$
S_p	ϵ	$AUTH(18)$	–	–
	$CONNECT()$	–	$CONNACK(N/Z)$	$CONNACK(0)$
L_p	$AUTH(0)$	–	–	–
	$AUTH(18)$	–	–	–
	$CONNECT()$ $CONNECT()$	–	–	–
	$CONNECT()$ $AUTH(0)$	$AUTH(18)$	–	–
	$CONNECT()$ $AUTH(18)$	$AUTH(18)$	–	–

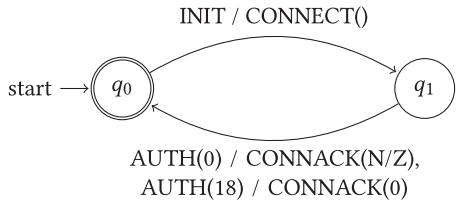


Fig. 9. The first hypothesis $\mathcal{H}$ constructed by protocol behavior learner.

The constructed hypothesis (Figure 9) after the second round conforms to the MQTT broker in version 5. This is verified by an 'ok' signal returned by the random walk testing, which results in no exploration phase initiated. Therefore, we conclude that the MQTT broker's behavior in version 5 is inferred successfully by the protocol behavior learner.

In terms of membership query count, the number of required membership queries by PBL until a successful inference is made is much less than that caused by the original L* algorithm. Table 7 shows the final observation table learned by the original L* algorithm. The results show that our proposed PBL consumes five membership queries while the original L* algorithm consumes 21 membership queries, which implies that the number of required membership queries decreases down to 23.8%. By comparing two final observation tables in Table 6 and 7, it is clearly shown that excluding membership queries corresponding to message sequences that do not conform to the legacy protocol's behavior is effective to decrease the amount of membership query consumption.

Our proposal shows not only improved learning efficiency but also higher compactness of the learning result. The minimum canonical Mealy machine that covers all authentication behavior of an MQTT broker in version 5 is composed of two states (q_0 and q_1) and two transitions (one with $CONNECT()/AUTH(18)$ and another with $AUTH(18)/CONNACK(0)$). Our proposal learns three different input-output pairs while the original L* algorithm learns six, implying that our proposal's compactness is 100% higher than that of the original L* algorithm. Hence, we conclude that our modified L* algorithm learns a more compact behavior model with fewer membership queries in this case study.

5.4 SLP Case Study

In this section, we analyze experiment results for the SLP syntax update scenario, which is summarized in Table 8. First of all, the most probable adaptation sequence length is 7, and its corresponding iteration count and elapsed time are 326 and 7.952 ms, respectively. Based on a

Table 7. The Final Observation Table Constructed by the Original L* Algorithm

		$\mathcal{D}$		
		$CONNECT()$	$AUTH(0)$	$AUTH(18)$
S_p	ϵ	$AUTH(18)$	$CONNACK(N/Z)$	$CONNACK(N/Z)$
	$CONNECT()$	$CONNACK(N/Z)$	$CONNACK(N/Z)$	$CONNACK(0)$
L_p	$AUTH(18)$	$AUTH(18)$	$CONNACK(N/Z)$	$CONNACK(N/Z)$
	$AUTH(0)$	$AUTH(18)$	$CONNACK(N/Z)$	$CONNACK(N/Z)$
	$CONNECT()$ $CONNECT()$	$AUTH(18)$	$CONNACK(N/Z)$	$CONNACK(N/Z)$
	$CONNECT()$ $AUTH(0)$	$AUTH(18)$	$CONNACK(N/Z)$	$CONNACK(N/Z)$
	$CONNECT()$ $AUTH(18)$	$AUTH(18)$	$CONNACK(N/Z)$	$CONNACK(N/Z)$

Table 8. Experiment Results for SLP Message Syntax Inference

Most Probable AdaptSeq Length	Iteration Count	Elapsed Time (ms)	Precision	Recall	Semantic Loss	Memory Usage (MB)
7	326	7,952	0.8	0.8	0.1429	8.19

well-established response time analysis [32], a response time shorter than 10 seconds is short enough to keep a user's attention focused on the current interaction. According to such analysis, we verify that the proposed scheme successfully infers the updated protocol message syntax and helps a legacy protocol agent perform an intended task before a user's attention is distracted and an interaction opportunity is gone in the SLP case. Especially, considering that SLP is a passive protocol by nature and does not strictly require realtimeness, keeping a user's attention is enough to discover a service hosted by an updated SLP agent. In addition, the results show that meaningful interaction is achieved even though 20% of the applied adaptation methods are inaccurate and 14.29% of the expected message semantics are lost. This result justifies our refined definition of meaningful interaction (Section 3.1). In terms of memory usage, the results show that the proposed scheme requires only 8.19 MB, which is similar to that of the MQTT case study. Detailed evaluation results of the syntax inference are shown in our previous work [40].

6 DISCUSSION

6.1 Generality of the Behavior Learning Performance

Since our proposed inference scheme is a heuristic approach that leverages the knowledge base as a bias to enhance the learning efficiency, its performance is considerably affected by knowledge construction. In our previous work [40], we conducted a complexity analysis that statistically approximates how knowledge base affects the performance of the protocol syntax inference. In the case of protocol behavior inference, even though multiple complexity analysis has been performed for the L* algorithm, it is still non-trivial to computationally analyze how much its performance can be improved by the involvement of the knowledge base. Therefore, the performance improvement in terms of membership query count presented in the MQTT case study can be case-specific, and we cannot guarantee that much of improvement for the other protocols. For example, the introduction of an empty cell in the observation table is regarded as an extra alphabet for the L* algorithm, which diversifies the representation of the target protocol's behavioral characteristics. Then, more prefixes and suffixes can be derived and compose more membership queries. However, when a different knowledge model about another legacy protocol is given and its complexity is higher than that of MQTT version 3.1.1 client, it can also require more membership queries.

To conduct a complexity analysis with regard to such influential factors and provide an upper bound on membership query count, it is necessary to computationally analyze the complexity of

a legacy protocol behavior and its corresponding entropy of an observation table. However, not enough protocol update cases or reference works with which we can conduct such modeling are present for the moment. Therefore, it should be a meaningful future achievement if we generalize the impact of the knowledge base on improving behavior learning performance via information-theoretic modeling.

6.2 Applicability to Open API-Based Protocol Inference

In this work, the proposed scheme focuses on adapting legacy message syntax and behavior of binary protocols such as SLP and MQTT. However, many IoT devices interact with each other using not only binary protocols but also open APIs. For example, most networked printers use various binary protocols, including SLP, for the purpose of service discovery and run an HTTP web server to provide open APIs via web interfaces. Such open API is connected to the back-end functionality via dedicated interfaces such as GCI script. Users or other devices can then exploit the open API-based protocol to conduct high-level interactions such as printing a document and configuration setup. There have been many existing works [5, 6] that resolve the incompatibility between open API-based protocols. Considering its popularity, it is worth evaluating whether our proposed scheme can be applied to such open API-based protocol's heterogeneity issues as well.

In terms of protocol syntax inference, the proposed scheme cannot be applied, but its philosophy (i.e., knowledge-based adaptive probing) should be effective. We can infer an updated open API syntax by segmenting each interface into meaningful keywords and understanding their semantics based on linguistic analysis. It will then be possible to infer possible adaptation candidates and their relative likelihood by gathering an update history of various open APIs as performed in this work. Since semantic analysis for interface keywords should be involved, the difficulty of the inference may increase. However, there are more examples of open API interfaces than binary protocols, and this larger dataset can help raise the inference accuracy. The current version of the proposed scheme is not equipped with such linguistic analysis capability, and we regard this as a limitation and valuable future work. In terms of protocol behavior inference, the proposed scheme can be applied without any modifications, and its effect should be preserved. L* algorithm is proved to be effective in both open API-based protocols and binary protocols in the existing works. Therefore, by rewriting only the knowledge of the legacy protocol behavior, we expect the proposed scheme to increase the learning efficiency even in the open API-based protocol behavior learning.

6.3 Inference of Completely Different Protocol Beyond Incompatible Version Update

When designing a new protocol inference scheme, it is critical to carefully decide to what extent we want to solve the heterogeneity. Depending on the goal and scope, the required level of inference and applied techniques change. Juba [18] mentions that communication under the complete absence of any common background suffers from severe overheads for practical purposes. This implies that a protocol agent cannot infer the syntax and behavior of a completely different protocol from its own in a reasonably short time. Based on the rationale, he devises the knowledge-based interactive Turing machine and verifies it with a protocol update scenario of IPv4. For the same reason, rather than inferring a completely unknown protocol's syntax and behavior, we set up our design goal to enable protocol agents with incompatible versions to accomplish their intended tasks when the updated protocol does not support backward compatibility. However, to the best of our knowledge, APRE schemes introduced in Section 2 are the most advanced practical solutions that try to infer a completely unknown protocol syntax. Even though they have applicability issues due to IoT environments' runtime characteristics, they perform the inference successfully with pattern analysis. Considering that making even completely different protocols interoperable at runtime is the ultimate challenge in this area, it must be a meaningful and interesting future

work to advance our proposed scheme by combining the essence of knowledge-based heuristic approaches and pattern analysis-based statistical approaches. If we can devise a hybrid approach that breaks their trade-off, it will be possible to perform a knowledge-based efficient inference while autonomously extending the knowledge by analyzing patterns of collected protocol traces at runtime. Once the knowledge becomes rich enough, we expect that a successful runtime inference of a completely different protocol can also be possible.

7 CONCLUSION

This article proposes an efficient protocol inference scheme for unknown but backward-incompatible protocol updates in IoT environments. The proposed scheme infers a message syntax and its exchanging behavior for the updated version of a legacy protocol based on a knowledge-based adaptive probing method at runtime to achieve meaningful interactions. By selectively composing membership queries only for input message sequences conforming to the legacy protocol's behavior, we prevent unnecessarily wasted membership queries and drastically decrease the number of required membership queries for behavior learning. The case study results show that our proposed scheme infers updated protocol syntax successfully with an affordable level of elapsed time and memory overhead and decreases the number of membership queries consumed by the L* algorithm for learning a compact canonical Mealy machine.

For future work, we plan to verify the performance of the proposed inference scheme computationally. It is not trivial to model the uncertainty of high-level factors involved with protocol updates. Despite such difficulty, performance verification is still essential to generalize our scheme's applicability. As another future work, we plan to evaluate whether our proposal can be applied to a real-world application from a practical point of view. We expect that findings from the real-world practice can be shared with protocol developers to remind the relevant communities of the importance of maintaining the protocol compatibility for the acceptance of IoT technologies.

REFERENCES

[1] Fides Aarts, Bengt Jonsson, and Johan Uijen. 2010. Generating models of infinite-state communication protocols using regular inference with abstraction. In *Proceedings of the IFIP International Conference on Testing Software and Systems*. 188–204.

[2] Dana Angluin. 1987. Learning regular sets from queries and counterexamples. *Information and Computation* 75, 2 (1987), 87–106.

[3] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. 2002. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing* 32, 1 (2002), 48–77.

[4] Marco Autili, Paola Inverardi, Romina Spalazzese, Massimo Tivoli, and Filippo Mignosi. 2019. Automated synthesis of application-layer connectors from automata-based specifications. *Journal of Computer and System Sciences* 104 (2019), 17–40.

[5] Nelly Bencomo, Amel Bennaceur, Paul Grace, Gordon Blair, and Valérie Issarny. 2013. The role of models@ run.time in supporting on-the-fly interoperability. *Computing* 95, 3 (2013), 167–190.

[6] Amel Bennaceur and Valérie Issarny. 2014. Automated synthesis of mediators to support component interoperability. *IEEE Transactions on Software Engineering* 41, 3 (2014), 221–240.

[7] Amel Bennaceur, Valérie Issarny, Richard Johansson, Alessandro Moschitti, Romina Spalazzese, and Daniel Sykes. 2011. Automatic service categorisation through machine learning in emergent middleware. In *Proceedings of the International Symposium on Formal Methods for Components and Objects*. 133–149.

[8] Michael Blackstock and Rodger Lea. 2014. IoT interoperability: A hub-based approach. In *Proceedings of the 2014 International Conference on the Internet of Things (IOT'14)*. IEEE, Los Alamitos, CA, 79–84.

[9] Yerom-David Bromberg, Paul Grace, and Laurent Réveillère. 2011. Starlink: Runtime interoperability between heterogeneous middleware protocols. In *Proceedings of the 2011 31st International Conference on Distributed Computing Systems*. IEEE, Los Alamitos, CA, 446–455.

[10] Juan Caballero, Pongsin Poosankam, Christian Kreibich, and Dawn Song. 2009. Dispatcher: Enabling active botnet infiltration using automatic protocol reverse-engineering. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*. ACM, New York, NY, 621–634.

[11] Archi Delphinanto, J. J. Lukkien, A. M. J. Koonen, A. J. P. S. Madureira, I. G. M. M. Niemegeers, F. T. H. den Hartog, and F. Selgert. 2007. Architecture of a bidirectional Bluetooth-UPNP proxy. In *Proceedings of the 4th Annual IEEE Consumer Communications and Networking Conference (CCNC'07)*. IEEE, Los Alamitos, CA, 34–38.
[12] Eclipse Foundaion. 2017. Eclipse Paho (1.2.0). Retrieved September 24, 2021 from https://www.eclipse.org/paho/.
[13] ej-technologies. 2018. JProfiler. Retrieved September 24, 2021 from https://www.ej-technologies.com/products/jprofiler/overview.html.
[14] Maryam Eslamichalandar, Kamel Barkaoui, and Hamid Reza Motahari-Nezhad. 2013. Dynamic adapter reconfiguration in the context of business protocol evolution. In *Proceedings of the 2013 IEEE 16th International Conference on Computational Science and Engineering (CSE'13)*. IEEE, Los Alamitos, CA, 301–308.
[15] Paul Fiterău-Broştean, Ramon Janssen, and Frits Vaandrager. 2014. Learning fragments of the TCP network protocol. In *Proceedings of the International Workshop on Formal Methods for Industrial Critical Systems*. 78–93.
[16] Hans Freudenthal. 1961. Lincos, design of a language for cosmic intercourse, part I, studies in logic and the foundations of mathematics. (1961).
[17] Paul Grace, Yérom-David Bromberg, Laurent Réveillère, and Gordon Blair. 2012. Overstar: An open approach to end-to-end middleware services in systems of systems. In *Proceedings of the ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*. 229–248.
[18] Brendan Juba. 2011. *Universal Semantic Communication*. Springer Science & Business Media.
[19] Jussi Kiljander, Alfredo D'elia, Francesco Morandi, Pasi Hyttinen, Janne Takalo-Mattila, Arto Ylisaukko-Oja, Juha-Pekka Soininen, and Tullio Salmon Cinotti. 2014. Semantic interoperability architecture for pervasive computing and Internet of Things. *IEEE Access* 2 (2014), 856–873.
[20] Ji Eun Kim, George Boulos, John Yackovich, Tassilo Barth, Christian Beckel, and Daniel Mosse. 2012. Seamless integration of heterogeneous devices and access control in smart homes. In *Proceedings of the 2012 8th International Conference on Intelligent Environments*. IEEE, Los Alamitos, CA, 206–213.
[21] Choongin Lee, Jeonghan Bae, and Heejo Lee. 2018. PRETT: Protocol reverse engineering using binary tokens and network traces. In *Proceedings of the IFIP International Conference on ICT Systems Security and Privacy Protection*. 141–155.
[22] Dongman Lee, Si Young Jang, Byoungheon Shin, and Yoonhyoung Lee. 2019. Towards dynamically reconfigurable IoT camera virtualization for video analytics edge cloud services. *IEEE Internet Computing* 23, 4 (2019), 10–17.
[23] David Lee, Krishan K. Sabnani, David M. Kristol, and Sanjoy Paul. 1996. Conformance testing of protocols specified as communicating finite state machines-a guided random walk based approach. *IEEE Transactions on Communications* 44, 5 (1996), 631–640.
[24] Barbara H. Liskov and Jeannette M. Wing. 1994. A behavioral notion of subtyping. *ACM Transactions on Programming Languages and Systems* 16, 6 (1994), 1811–1841.
[25] Min Liu, Chunfu Jia, Lu Liu, and Zhi Wang. 2013. Extracting sent message formats from executables using backward slicing. In *Proceedings of the 2013 4th International Conference on Emerging Intelligent Data and Web Technologies*. IEEE, Los Alamitos, CA, 377–384.
[26] Jian-Zhen Luo and Shun-Zheng Yu. 2013. Position-based automatic reverse engineering of network protocols. *Journal of Network and Computer Applications* 36, 3 (2013), 1070–1077.
[27] Radu Mateescu, Pascal Poizat, and Gwen Salaün. 2011. Adaptation of service protocols using process algebra and on-the-fly reduction techniques. *IEEE Transactions on Software Engineering* 38, 4 (2011), 755–777.
[28] Hamid Reza Motahari Nezhad, Guang Yuan Xu, and Boualem Benatallah. 2010. Protocol-aware matching of web service interfaces for adapter development. In *Proceedings of the 19th International Conference on World Wide Web*. ACM, New York, NY, 731–740.
[29] Eclipse Foundation. 2017. Moquette MQTT. Retrieved September 24, 2021 from https://projects.eclipse.org/projects/iot.moquette.
[30] Jin Nakazawa, Hideyuki Tokuda, W. Keith Edwards, and Umakishore Ramachandran. 2006. A bridging framework for universal interoperability in pervasive systems. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*. IEEE, Los Alamitos, CA, 3.
[31] Netty Project. 2018. Netty Project (4.1.25). Retrieved September 24, 2021 from http://netty.io/.
[32] Jakob Nielsen. 1994. *Usability Engineering*. Elsevier.
[33] Richard Olaniyan, Olamilekan Fadahunsi, Muthucumaru Maheswaran, and Mohamed Faten Zhani. 2018. Opportunistic edge computing: Concepts, opportunities and research challenges. *Future Generation Computer Systems* 89 (2018), 633–645.
[34] Hyunho Park, Byoungoh Kim, Yangwoo Ko, and Dongman Lee. 2011. InterX: A service interoperability gateway for heterogeneous smart objects. In *Proceedings of the 2011 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops'11)*. IEEE, Los Alamitos, CA, 233–238.

[35] Thomas Pramsohler, Simon Schenk, Andreas Barthels, and Uwe Baumgarten. 2015. A layered interface-adaptation architecture for distributed component-based systems. *Future Generation Computer Systems* 47 (2015), 113–126.
[36] Harald Raffelt, Bernhard Steffen, Therese Berg, and Tiziana Margaria. 2009. LearnLib: A framework for extrapolating behavioral models. *International Journal on Software Tools for Technology Transfer* 11, 5 (2009), 393.
[37] Jan S. Rellermeyer. 2008. JSLP project, Java Service Location Protocol, Retrieved October 4, 2021 from http://jslp.sourceforge.net
[38] Eve M. Schooler, Milan Milenkovic, Keith A. Ellis, Jessica McCarthy, Jeff Sedayao, and Brian McCarson. 2018. Rational interoperability: A pragmatic path toward a data-centric IoT. In *Proceedings of the 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS'18)*. IEEE, Los Alamitos, CA, 1139–1149.
[39] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge computing: Vision and challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646.
[40] Heesuk Son and Dongman Lee. 2019. Towards interactive networking: Runtime message inference approach for incompatible protocol updates in IoT environments. *Future Generation Computer Systems* 96 (2019), 563–578.
[41] Bernhard Steffen, Falk Howar, and Maik Merten. 2011. Introduction to active automata learning from a practical perspective. In *Formal Methods for Eternal Networked Software Systems*. Lecture Notes in Computer Science, Vol. 6659. Springer, 256–296.
[42] Chris McMahon Stone, Tom Chothia, and Joeri de Ruiter. 2018. Extending automated protocol state learning for the 802.11 4-way handshake. In *Proceedings of the European Symposium on Research in Computer Security*. 325–345.
[43] Harald Sundmaeker, Patrick Guillemin, Peter Friess, and Sylvie Woelfflé. 2010. Vision and challenges for realising the Internet of Things. *Cluster of European Research Projects on the Internet of Things, European Commission* 3, 3 (2010), 34–36.
[44] Martin Tappler, Bernhard K. Aichernig, and Roderick Bloem. 2017. Model-based testing IoT communication via active automata learning. In *Proceedings of the 2017 IEEE International Conference on Software Testing, Verification, and Validation (ICST'17)*. IEEE, Los Alamitos, CA, 276–287.
[45] Yipeng Wang, Xingjian Li, Jiao Meng, Yong Zhao, Zhibin Zhang, and Li Guo. 2011. Biprominer: Automatic mining of binary protocol features. In *Proceedings of the 2011 12th International Conference on Parallel and Distributed Computing, Applications, and Technologies*. IEEE, Los Alamitos, CA, 179–184.
[46] Franco Zambonelli. 2015. Engineering self-organizing urban superorganisms. *Engineering Applications of Artificial Intelligence* 41 (2015), 325–332.

Received July 2019; revised June 2020; accepted October 2020