Model-based Testing IoT Communication via Active Automata Learning

Martin Tappler Bernhard K. Aichernig
Institute of Software Technology
Graz University of Technology, Austria
{martin.tappler, aichernig}@ist.tugraz.at

Roderick Bloem
Institute of Applied Information Processing and Communications
Graz University of Technology, Austria
roderick.bloem@iaik.tugraz.at

ICST `16 (International Conference on Software Testing, Verification and Validation)
Presenter: Dongwoo kim

Software Safety Engineering LAB

Overview

Motivation

- Black-box System의 행위를 파악하기 위한 연구가 활발히 진행되고 있으며
  Active Automata Learning은 학습 기반으로 black-box system의 행위를 파악한다.
- 같은 specification을 공유하는 다수의 implementation이 있을 경우 Learning을 통하여
  각 implementation을 학습하고 서로 비교함으로써 implementation에 포함된 버그를 탐지할 수 있을 것이다.
  - e.g.) MQTT protocol implementations, Kakaotalk GUI

Approach

- Learning-based method를 이용하여 IoT 기반의 reactive system의 failure case를 탐지하고자 한다.

1. 기존에 구현된 여러 시스템의 행위를 Active Automata Learning을 이용하여 Mealy machine 형식으로 추출
2. 추출된 machine간의 equivalence를 확인하여 동일하지 않은 경우 fault가 있다고 의심
3. 수작업으로 동일하지 않은 행위에 대해서 Specification을 참조하여 분석하고 fault 여부 확인

Evaluation(Case study)

- IoT 기반의 system이 통신 protocol을 잘못 사용하여 spec을 위반하는 경우를 탐지
- 대상 : MQTT protocol(IoT에서 사용되는 protocol)을 구현한 5개의 시스템
- 위의 방법을 이용하여 생성한 model을 비교하여 18개의 오류 발견하였으며
  하나의 시스템을 제외한 모든 시스템에 오류가 있음을 확인하였다.

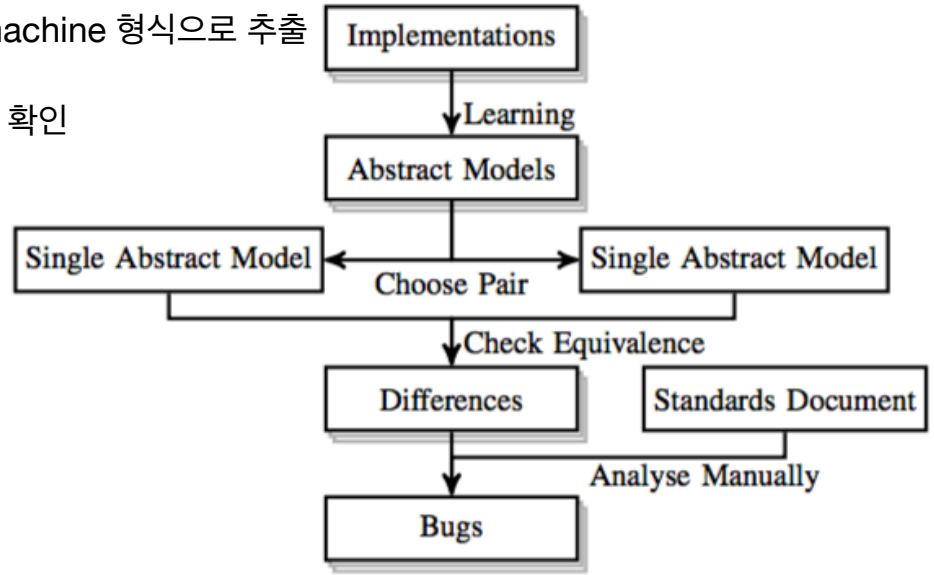


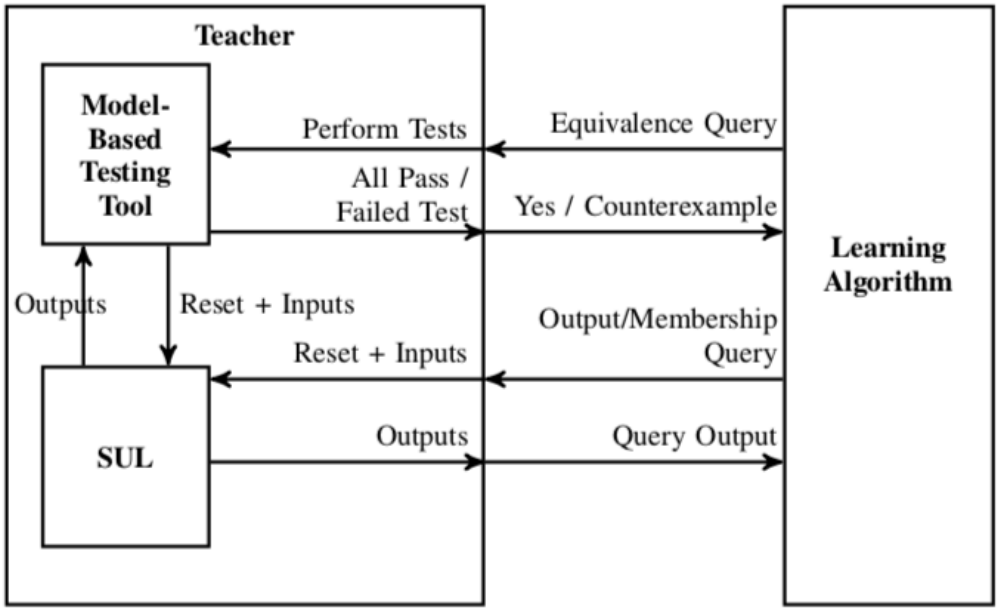
Fig. 1. Overview of bug-detection process.

Preliminaries - Active Automata Learning

- Learning algorithm이 Teacher에게 질의를 통하여 모델을 생성
- 생성된 모델은 Mealy machine의 형태

- **다음의 세가지 질의를 이용하여 Learning을 수행**
  - reset: SUL를 초기화
  - output query: input을 제공한 후 어떤 output이 출력되는지 확인하여 learned model에 추가
  - equivalence query: learned model이 충분히 학습되었는지 확인
    - 몇 가지 input을 이용하여 learned model이 같은 결과를 출력하는지 확인

- **Learning Algorithm**
  1. output query를 수행한 후 learned model을 생성
  2. Learned Model이 SUL과 equivalent한지 확인
  3. Equivalent하지 않으면 goto 1.

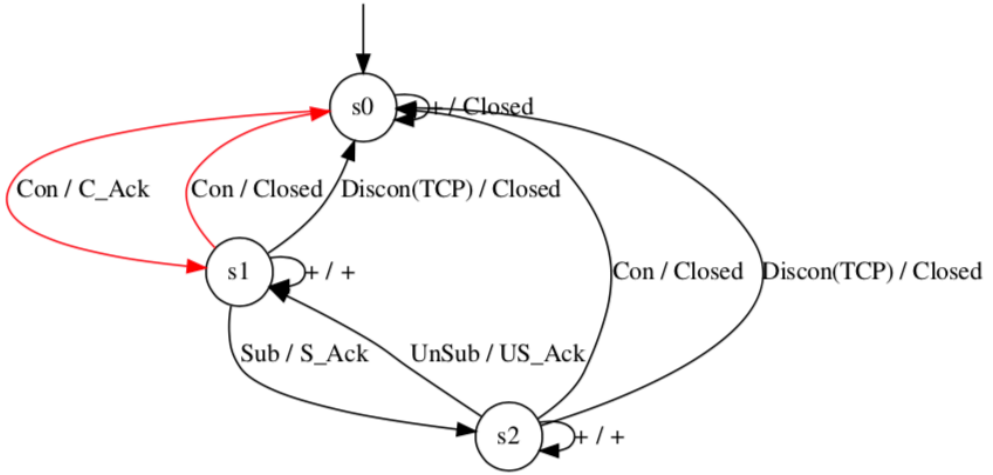


<Active Automata Learning 개념도>

Preliminaries - Mealy Machines

Definition III.1 (Mealy Machines). A Mealy machine is a 6-tuple $\langle Q, q_0, I, O, \delta, \lambda \rangle$ where

- Q is a finite set of states,
- q_0 is the initial state,
- I/O is a finite set of input/outputs symbols,
- $\delta : Q \times I \to Q$ is the state transition function, and
- $\lambda : Q \times I \to O$ is the output function

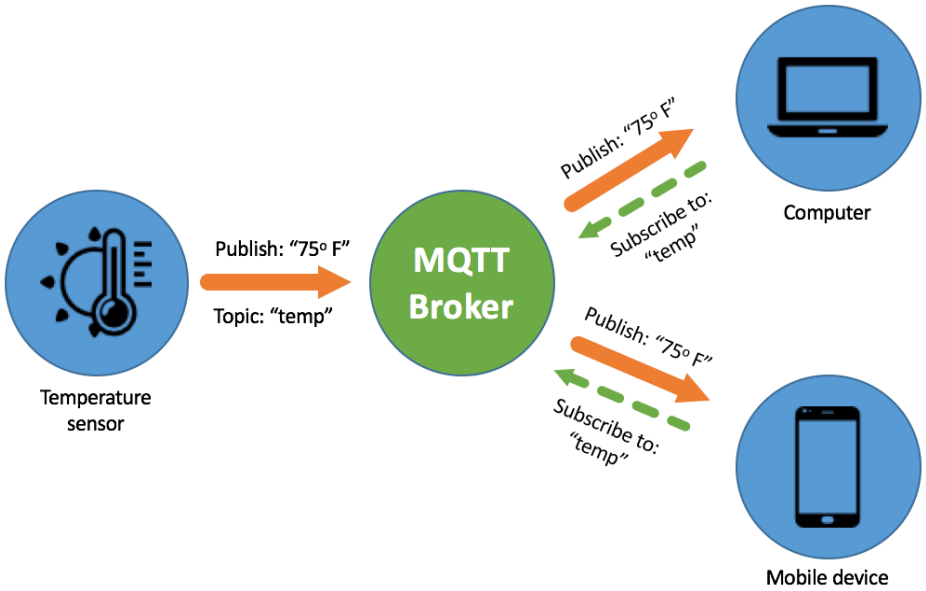


<Example of Mealy Machine>

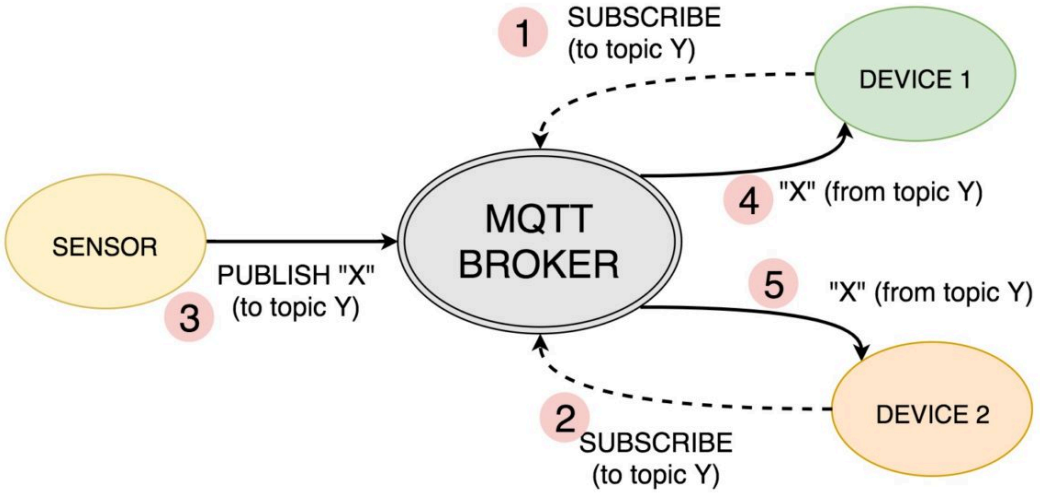
- Active Automata Learning을 이용하여 Mealy machine을 생성
- Java의 LearnLib를 활용하면 blackbox로부터 Mealy machine을 추출할 수 있음
- 본 연구에서 정의하는 Mealy Machine은 input enabled/deterministic
  - input enabled: 각각의 (상태, 입력) 튜플에 대해서 출력과 다음 상태가 존재
  - deterministic: 각각의 (상태, 입력) 튜플에 대해 하나의 출력과 다음 상태가 존재

Approach - MQTT Protocol

- **MQTT protocol**
  - Light-weight publish/subscribe protocol
  - Publisher : 메시지 전송을 수행 / Subscriber : 메시지 수신
  - Publisher가 전송할 메시지를 Broker에 등록하면 Broker가 Subscriber에게 Forward
  - Well-suited in resource constrained environment such as IoT
  - 본 연구에서는 MQTT Broker implementation 5종에게 Case study를 수행함



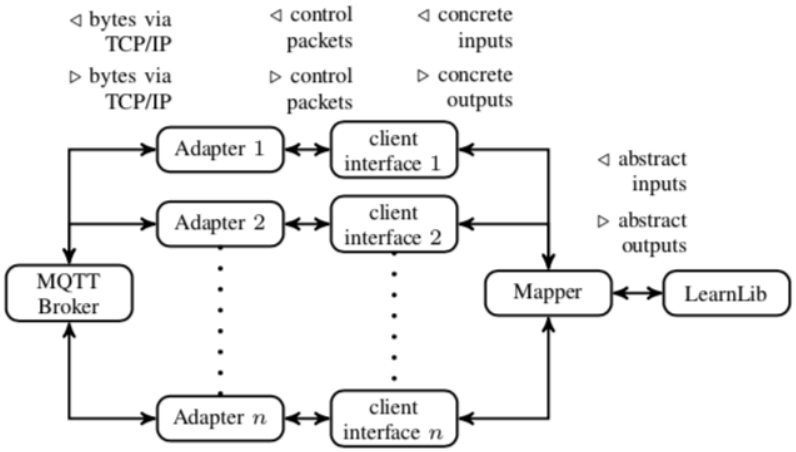
<Broker와 Client의 관계>



<Message publish 과정>

Approach - Environment 구성

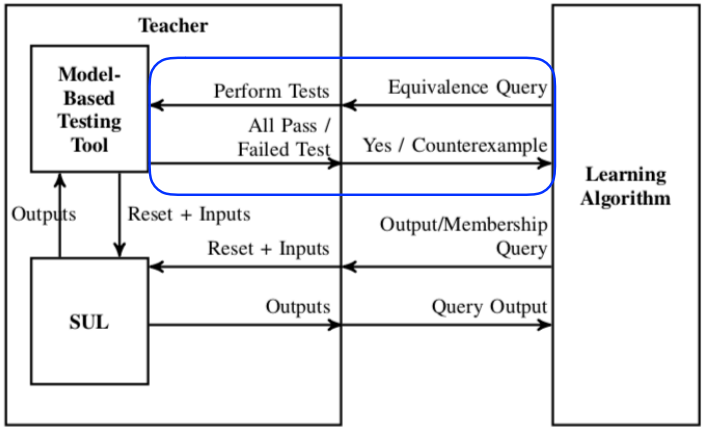
- **MQTT의 environment**
  1. client간의 dependency를 어떻게 정의할 것인가?
     - 본 연구에서는 7가지 환경을 정의하여 사용
       - Simple: 하나의 client만 존재 / 기본적인 publish/subscribe 기능 최대 7개 수행
       - Two clients with Retained will: 두 개의 client 존재 /
         - 하나의 client c1이 topic p1에 will을 남기고 비성장 종료 /
         - 다른 client c2가 p1을 subscribe
       - …
  2. 많은 수의 가능한 input/output을 어떻게 다룰 것인가?
     - 본 연구에서는 Mapper component를 두어 input/output을 abstract하여 다룬다.
     - LearnLib는 사용할 수 있는 abstract input을 선택하여 입력하고
     - Mapper는 abstract input을 concretize하여 각 client에 명령을 수행
     - 각 client로부터 발생된 각 출력은 Mapper에 의해 하나로 통합
     - Mapper는 하나의 추상화된 출력을 생성



<MQTT implementation을 학습하기 위한 환경>

- **Practical Considerations**
  - timeout: 일정 timeout 시간동안 도착하지 않은 메시지는 “empty” 메시지로 간주
  - outputs: 여러 client로 부터 생성된 메시지는 다양한 순서로 Mapper에 도착하므로 Mapper에서는 이를 정렬하여 처리 (같은 입력에 대해 다양한 출력이 생기는 것을 방지)
    - 본 방법을 사용했을 때 도착한 메시지 순서 정보는 잃게 되어 completeness는 잃게되지만 efficiency는 얻을 수 있다.
  - Restrictions: 시간과 관련된 행위는 mealy machine으로 충분히 표현할 수 없기 때문에 제외 (ex. ping)

Approach - Learning & Equivalence Checking



- Learning Algorithm을 통해 생성한 모델이 SUL과 일치하는지 확인하기 위해 Random test를 사용하였다.
  - 사용한 옵션은 다음과 같다.
  - probability of resetting the SUL: 0.05 // 5% 확률로 reset한 후 처음부터 다시 비교
  - maximum number of steps: 10000 // 10000 steps
  - reset step count after finding a counterexample: *true*

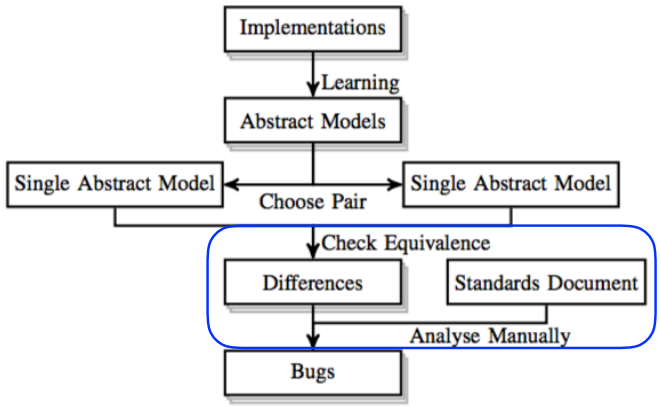


Fig. 1. Overview of bug-detection process.

- **Bi-simulation Checking**: 두 모델이 동일하다는 결론을 내리거나 counter example을 찾을 수 있다.
  - Counter example은 두 모델 중 하나에 이상이 있다거나 두 모델 모두에 이상이 있음을 나타낸다.
  - Bi-simulation checking 방법
    1. Product graph를 생성: 각 상태가 두 모델의 상태의 pair로 이루어진 graph
    2. Product graph에 equivalent하지 않음을 나타내는 fail-state 존재
    3. 두 모델 모두에 같은 transition(input/output이 동일)이 있으면 product graph에 추가
    4. 둘 중 하나의 모델에만 존재하는 transition은 fail state로 연결
  - Bi-simulation을 통해서 counter example이 발견된 경우 fail state에 도달하는 state sequence를 사용자에게 보고

Case study(1)

- **Implementation**
  - Implemented in Scala
  - Java-library LearnLib 사용
    - 대부분의 기능은 LearnLib에서 구현
  - Mapper는 직접 구현
    - 본 연구에서는 7종류의 environment를 나타내는 mapper를 사용

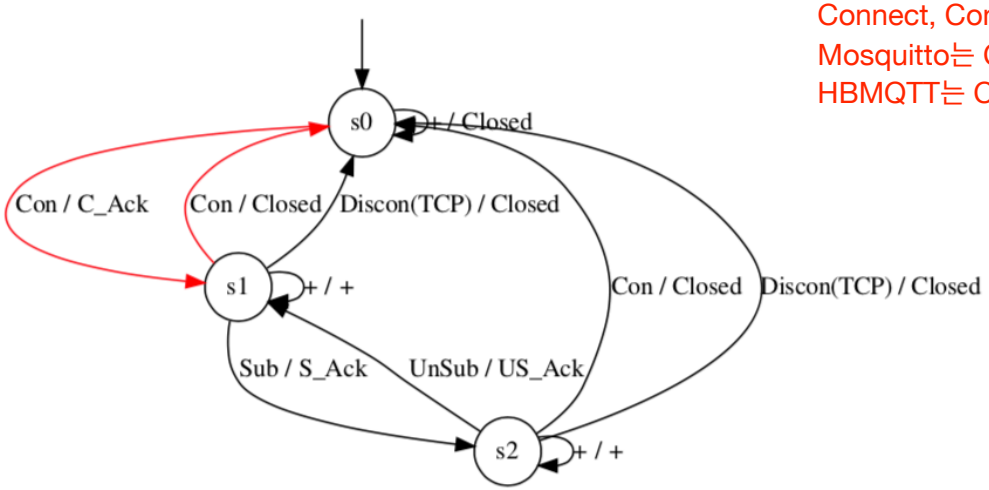
- **Experiment**
  - MQTT protocol을 구현한 아래의 5개의 implementation으로부터 모델을 생성
    - Apache ActiveMQ 5.13.3[2]
    - emqttd 1.0.2 [3]
    - HBMQTT 0.7.1 [4]
    - Mosquitto 1.4.9 [5]
    - VerneMQ 0.12.5p4 [6]

Implementation	Timeout in Milliseconds
Apache ActiveMQ	300
emqttd	25
HBMQTT	100
Mosquitto	100
VerneMQ	300

  - 각각의 implementation은 다른 timeout 수치를 사용하는 것을 실험을 통해 확인
  - 각 implementation에서 제공하는 timeout을 timeout bound로 사용

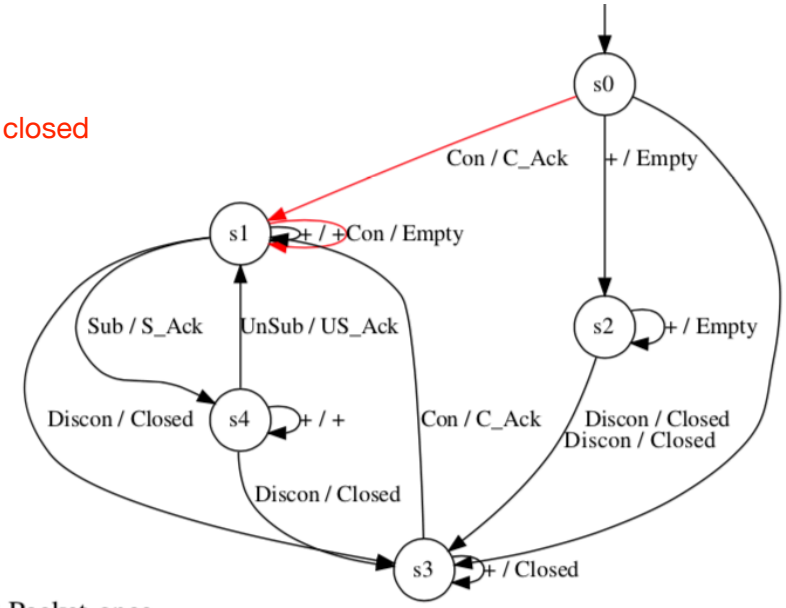
Case study(2)

- Case study를 통해서 17개의 버그를 발견(Mosquitto에서는 발견되지 않음)
  - 이 중 네 개는 이미 reported 나머지는 수정되었거나 검토 중
  - Specification 위반
    1. Simple Mapper - Mosquito, HBMQTT 가 서로 다른 행동을 하는 것을 확인



(a) Mosquitto model

Connect, Connect 입력에 대해
Mosquitto는 C_Ack, Connection closed
HBMQTT는 C_Ack, Empty를 출력



(b) HBMQTT model

A Client can only send the CONNECT Packet once over a Network Connection. The Server MUST process a second CONNECT Packet sent from a Client as a protocol violation and disconnect the Client [MQTT-3.1.0-2].

Spec에 따라 Mosquitto가 바른 행위를 보여주었음을 확인

Case study(3)

- Case study를 통해서 17개의 버그를 발견(Mosquitto에서는 발견되지 않음)
  - 이 중 네 개는 이미 reported, 나머지는 수정되었거나 검토 중
  - Specification 위반
    1. Simple Mapper - Mosquito, HBMQTT 가 서로 다른 행동을 하는 것을 확인
    2. Two clients with Retained Will - Mosquitto, emqttd, ActiveMQ 가 서로 다른 행동을 하는 것을 확인

1) A client connects with client identifier `Client1`
2) A client connects with client identifier `Client2` with retained will message `bye` for topic `c2_will`
3) `Client2` disconnects unexpectedly (such that the will message is published)
4) `Client1` subscribes to `c2_will`
5) `Client1` subscribes to `c2_will`

// 다섯번째 step에서 Mosquitto와 emitted,ActiveMQ가 다르게 행동

<버그를 발견한 시나리오>

- Mosquitto는 5번째 step에서 bye를 전송, 나머지는 empty를 전송
  - 표준에 따라 Mosquitto가 바른 행위를 하는 것을 확인

cording to [MQTT-3.8.4-3] [14] which states that repeated subscription requests must replace existing subscriptions and that "any existing retained messages matching the Topic Filter MUST be re-sent".

  - 표준에 따르면 subscription을 반복할 시에 will message를 반복해서 보내주어야한다고 명시

Case study(4)

- Case study를 통해서 17개의 버그를 발견(Mosquitto에서는 발견되지 않음)
  - 이 중 네 개는 이미 reported, 나머지는 수정되었거나 검토 중
  - Specification 위반
    1. Simple Mapper - Mosquito, HBMQTT 가 서로 다른 행동을 하는 것을 확인
    2. Two clients with Retained Will - Mosquitto, emqttd, ActiveMQ 가 서로 다른 행동을 하는 것을 확인
  - Non-determinism
    - 본 연구에서는 총 35개의 모델 생성 중 세 개의 모델에서 non-determinism이 포함된 것을 발견하여 실험에서 제외
  - Efficiency

	ActiveMQ	emqttd	HBMQTT	Mosquitto	VerneMQ
# states	4	3	5	3	3
MQ time[s]	59.72	3.87	31.94	14.01	43.91
MQ # queries	88	59	110	56	57
CT time[s]	914.18	78.3	491.06	278.21	915.77
CT # queries	525	519	482	487	490
# equivalence queries	4	3	4	3	3

<Simple Mapper>

	ActiveMQ	emqttd	HBMQTT	Mosquitto	VerneMQ
# states	18	18	17	18	17
MQ time[s]	1855.55	167.32	557.14	641.89	1570.8
MQ # queries	732	735	640	730	625
CT time[s]	4787.92	481.36	2022.47	1612.59	4355.97
CT # queries	672	816	613	670	658
# equivalence queries	13	12	11	9	11

<Two clients with Retained Will Mapper>

Implementation	Timeout in Milliseconds
Apache ActiveMQ	300
emqttd	25
HBMQTT	100
Mosquitto	100
VerneMQ	300

- 본 연구에서 생성한 모델 중 가장 큰 모델은 18개의 상태를 가지고 있었다.
  - 상태 수가 적은 이유는 한 번의 query에 소요되는 실행시간이 크기 때문에 적은 수의 상태를 가진 모델을 사용하더라도 오랜 시간이 걸린다.
- ActiveMQ나 VerneMQ의 경우 항상 많은 시간이 소요됨을 알 수 있다. (timeout 수치가 크기 때문)
  - 이를 해결하기 위해서 domain-specific optimization, heuristics, smart test selection 등을 활용할 수 있다.

Conclusion

- 학습기법을 통해 Black-box 시스템으로부터 버그를 탐지하는 방법을 소개
- Active Automata Learning을 통해 MQTT Protocol의 Implementation 들을 학습하고 Equivalence를 비교함으로써 Implementation의 비정상 행위 탐지
- 5개의 MQTT Implementation으로부터 학습한 모델을 서로 비교해본 결과 총 18개의 행위 불일치를 탐지할 수 있었음
- 제안한 방법의 수행 속도는 각 implementation의 runtime에 영향을 받아 소요 시간의 편차가 크고 학습에 오랜 시간이 걸리는 것을 확인