A Comparison of IoT Communication Libraries: APIs and Performances

Jianbin Lai
03/11/2024
Concordia University

Supervisor: Sandra Céspedes,Yann-Gaël Guéhéneuc

IoT
Internet of Things

Search and rescue mission (drone)

Car connection (BMW car sharing)

Smart home

Field monitoring (track humidity)

Health (iWatch, Fitbit)

ref:https://www.hivemq.com/article/mqtt-standard-for-connected-car/
ref:https://mqtt.org/use-cases/
ref:https://www.freepik.com/free-photo/medical-banner-with-doctor-holding-tablet_30555917.htm#query=iot%20healthcare&position=2&from_view=search&track=ais&uuid=2e4185fd-77fd-4abb-8dd3-04376ccdebf0

Context

IoT Devices Quantity

- In 2019, 8.6 billions
- In 2030, 29.4 billions

⇒ **On average 3 devices/person in the world in 2030**

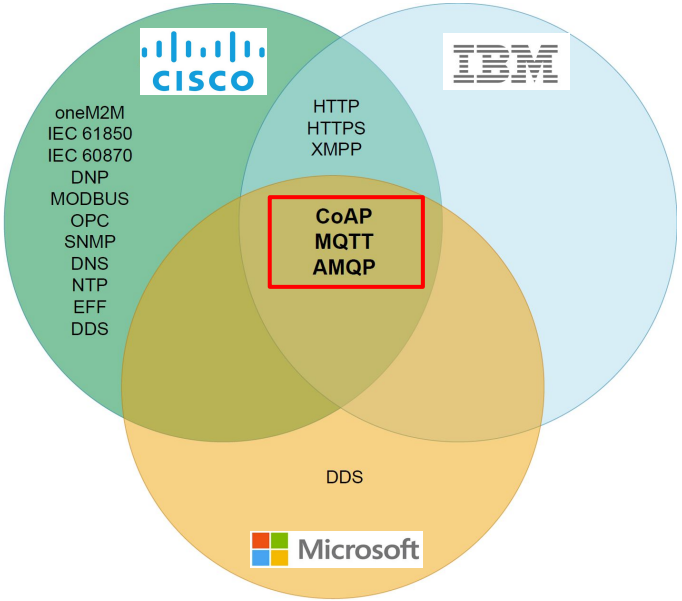
Application With Diverse Requirements

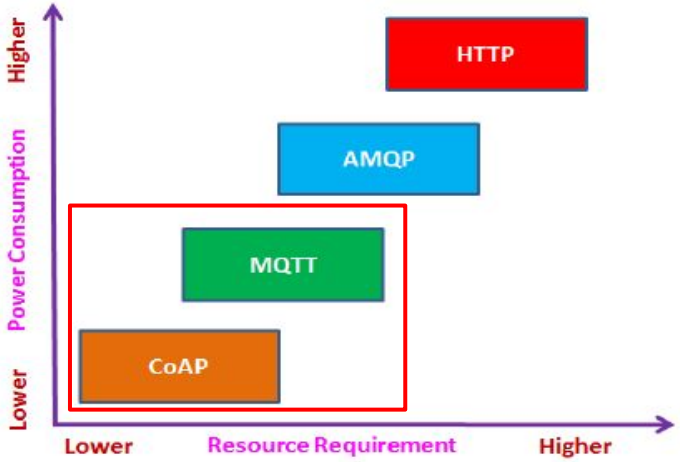
- Real-time communication: XMPP
- Large data transmission: HTTP/HTTPS
- Easy to connect: BLE

⇒ **Many choices, need for comparison**

ref:https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

Background



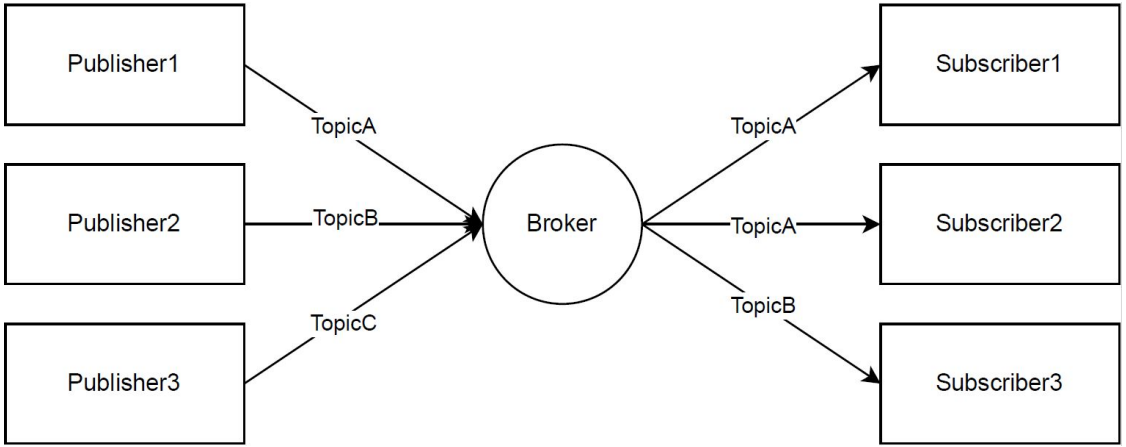


ref:https://azure.microsoft.com/en-ca/solutions/iot/iot-technology-protocols/
ref:https://www.ciscopress.com/articles/article.asp?p=2923211&seqNum=6
ref: https://developer.ibm.com/articles/iot-lp101-connectivity-network-protocols/
ref:https://ieeexplore.ieee.org/document/9480384
ref:https://ieeexplore.ieee.org/document/8088251

Background

MQTT

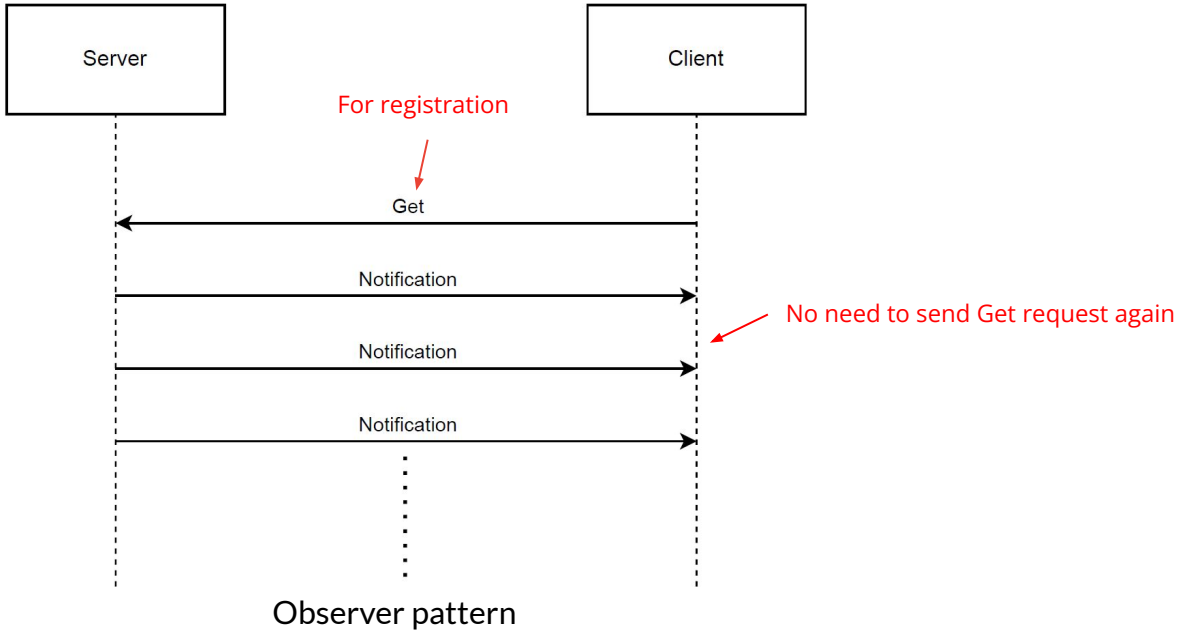
The protocol mainly runs on TCP/IP. MQTT adopts a publish/subscribe messaging pattern.



ref:https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

Background

CoAP (Constrained Application Protocol)



Observer pattern

ref:https://www.rfc-editor.org/rfc/rfc7252

Background

	CoAP	MQTT	MQTT-SN
Underlying Protocol	UDP, TCP	TCP	UDP
Reliability	NON, CON	QoS0, QoS1, QoS2	QOS-1, QoS0, QoS1 QoS2, QoS
Security	DTLS (UDP), OSCORE, TLS (TCP)	TLS	DTLS
Messaging exchange Mode	Asynchronous, Synchronous	Asynchronous	Asynchronous
Messaging pattern	Request-response pattern Observer pattern	Publish-subscribe pattern	Publish-subscribe pattern

ref:https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

Problems

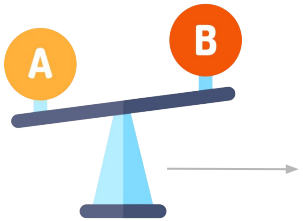
Literature review:

1. Some studies just talk about API, not related to IoT
   E.g., [19][20]
2. Some studies just survey or compare the protocols, not related to APIs
   E.g., [5][7][23]
3. Some studies just use one implementation of CoAP to compare with one implementation of MQTT
   E.g., [13][25][26]

Summary:

Lack of studies comparing implementations of different protocols, while considering their APIs

Approach



Comparison

Static Metrics	Packet Metrics	Runtime Metrics
1. Coupling Between Object (CBO)	1. Theoretical packet sizes	1.Goodput
2. Cyclomatic complexity (CC)	2. Practical packet sizes	2.Throughput
3. Lack of Cohesion between methods (LCOM)	3. Adjusted practical packet sizes	3.Overhead
4. Lines of Code (LOC)		

Scenarios

Approach

1. Survey and select different libraries of the protocols
2. Set up an IoT infrastructure
3. Collect, devise, and implement scenarios using the chosen libraries
4. Collect static metrics on the implemented senders and receivers
5. Analyse packet sizes theoretically and practically
6. Collect runtime performances when running the scenarios
7. Compare the collected measures and recommend a protocol/library

Approach

1. **Survey and select different libraries of the protocols**
2. Set up an IoT infrastructure
3. Collect, devise, and implement scenarios using the chosen libraries
4. Collect static metrics on the implemented senders and receivers
5. Analyse packet sizes theoretically and practically
6. Collect runtime performances when running the scenarios
7. Compare the collected measures and recommend a protocol/library

1. Survey and select different libraries of the protocols

CoAP
Constrained **A**pplication **P**rotocol

Name	Programming Language	Client/Server	License	Latest update date
Californium	Java	Client + Server	EPL+EDL	Feb 14, 2023
~~CoAP Shell~~	~~Java~~	~~Client~~	~~Apache License 2.0~~	~~Jun 1, 2021~~
java-coap	Java	Client + Server	Apache License 2.0	Jan 12, 2022
~~jcoap~~	~~Java~~	~~Client + Server~~	~~Apache License 2.0~~	~~June 12, 2012~~
~~nCoap~~	~~Java~~	~~Client + Server~~	~~BSD~~	~~Mar 20, 2018~~
~~Sensinode Java Device Library~~	~~Java SE~~	~~Client + Server~~	~~Commercial~~	~~Unkown~~
~~Sensinode NanoService Platform~~	~~Java SE~~	~~Client + Server~~	~~Commercial~~	~~Unkown~~

1. Survey and select different libraries of the protocols

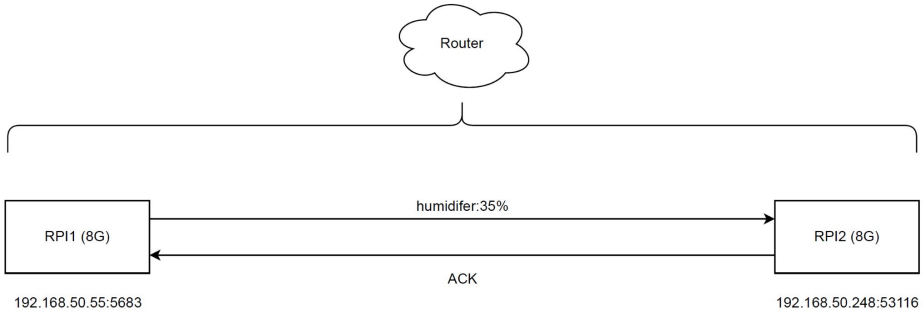
Name	Programming Language	Type	License	Latest update date
~~PubSub+~~	~~C, C#/.Net, Java, JavaScript (NodeJs), Python, Go~~	~~Broker~~	~~Commercial license, free version~~	~~Jan 14, 2021~~
Paho MQTT	C, C++, C#, Go, Java, JavaScript, Python, Rust	Client	Eclipse Public License 1.0, Eclipse Distribution License 1.0 (BSD)	Aug 6, 2022
~~Thingstream~~	~~C, C++, Java, JavaScript, Python, Go~~	~~Client, Broker~~	~~Commercial licence version 2.0~~	~~Mar 14, 2019~~
HiveMQ MQTT Client	Java	Client	Apache License version 2.0	Feb 15, 2023
~~HiveMQ Community Edition~~	~~Java~~	~~Broker~~	~~Apache License 2.0~~	~~Mar 1, 2023~~
~~HiveMQ~~	~~Java~~	~~Broker~~	~~Commercial license~~	~~Feb 7, 2023~~
~~JoramMQ~~	~~Java~~	~~Broker~~	~~Commercial~~	~~June 7, 2022~~
~~moquette~~	~~Java~~	~~Broker~~	~~Apache License version 2.0~~	~~Feb 26, 2023~~
~~OpenRemote MQTT Broker~~	~~Java~~	~~Broker~~	~~AGPLv3~~	~~Feb 27, 2023~~
~~OpenHAB MQTT binding~~	~~Java~~	~~Client~~	~~Eclipse Public License~~	~~Apr 21 2020~~

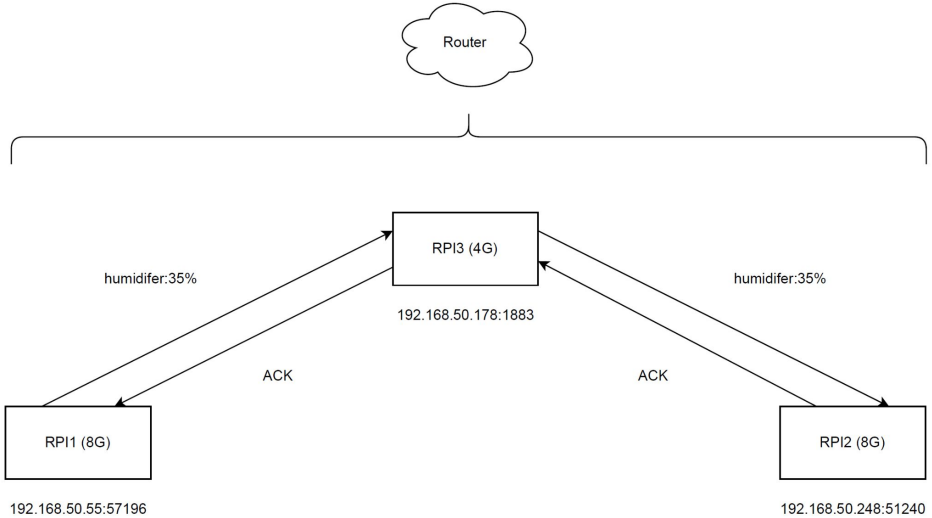
Approach

1. Survey and select different libraries of the protocols
2. **Set up an IoT infrastructure**
3. Collect, devise, and implement scenarios using the chosen libraries
4. Collect static metrics on the implemented senders and receivers
5. Analyse packet sizes theoretically and practically
6. Collect runtime performances when running the scenarios
7. Compare the collected measures and recommend a protocol/library

2. Set up an IoT infrastructure

CoAP
Constrained **A**pplication
Protocol





Approach

1. Survey and select different libraries of the protocols
2. Set up an IoT infrastructure
3. **Collect, devise, and implement scenarios using the chosen libraries**
4. Collect static metrics on the implemented senders and receivers
5. Analyse packet sizes theoretically and practically
6. Collect runtime performances when running the scenarios
7. Compare the collected measures and recommend a protocol/library

3. Collect, devise, and implement scenarios using the chosen libraries

Collect scenario

```
connOpts.setUserName("IamPublisherOne");
connOpts.setPassword("123456".getBytes());
```

		CoAP		MQTT	
		Californium	java-coap	HiveMQ	PahoMQTT
Authentication	Server (CoAP) / Broker (MQTT)	OSCORE	N/A	N/A	N/A
	Client (CoAP) / Pub (MQTT) / Sub (MQTT)	OSCORE	N/A	usernames/passwords	
Transport-layer Security	DTLS directly	YES	NO	?	?
	DTLS with SSLContext	NO	NO	?	?
	TLS directly	?	NO	NO	NO
	TLS with SSLContext	?	YES	YES	YES

```
1 SSLContext context = SSLContext.getInstance("TLSv1.3");
2 context.init(null, tmf.getTrustManagers(), new java.security.SecureRandom());
```

Repository	Solution
Californium	DTLS v1.2 directly
java-coap	TLS v1.3 with SSLContext
PAHO MQTT	TLS v1.3 with SSLContext
HiveMQ MQTT Client	TLS v1.3 with SSLContext

ref:https://www.ssl.com/faqs/faq-what-is-ssl/
ref:https://stackoverflow.com/questions/31634258/what-is-ssl-context

3. Collect, devise, and implement scenarios using the chosen libraries

Devise scenario

		CoAP	MQTT
No Security		NON	QoS0
		CON	QoS1
Security	Californium DTLS v1.2 directly java-coap TLS v1.3 with SSLContext PAHO MQTT TLS v1.3 with SSLContext iveMQ MQTT Client TLS v1.3 with SSLContext	NON	QoS0
		CON	QoS1

ref:https://www.ssl.com/faqs/faq-what-is-ssl/
ref:https://stackoverflow.com/questions/31634258/what-is-ssl-context

3. Collect, devise, and implement scenarios using the chosen libraries

Implement Scenarios

```
1  this.setObserveType(Type.NON);          // Californium
2  this.setConNotifications(false);        // java-coap
```

Code 5.2: Set observing resource to send NON to the client

```
1  // HiveMQ MQTT Client
2  client1.connect(connectMessage);
3  while(client1.getState().isConnected()==false) {
4      //do nothing, just wait for connected
5  }
6
7  // Paho MQTT
8  client1.connect(connOpts);
```

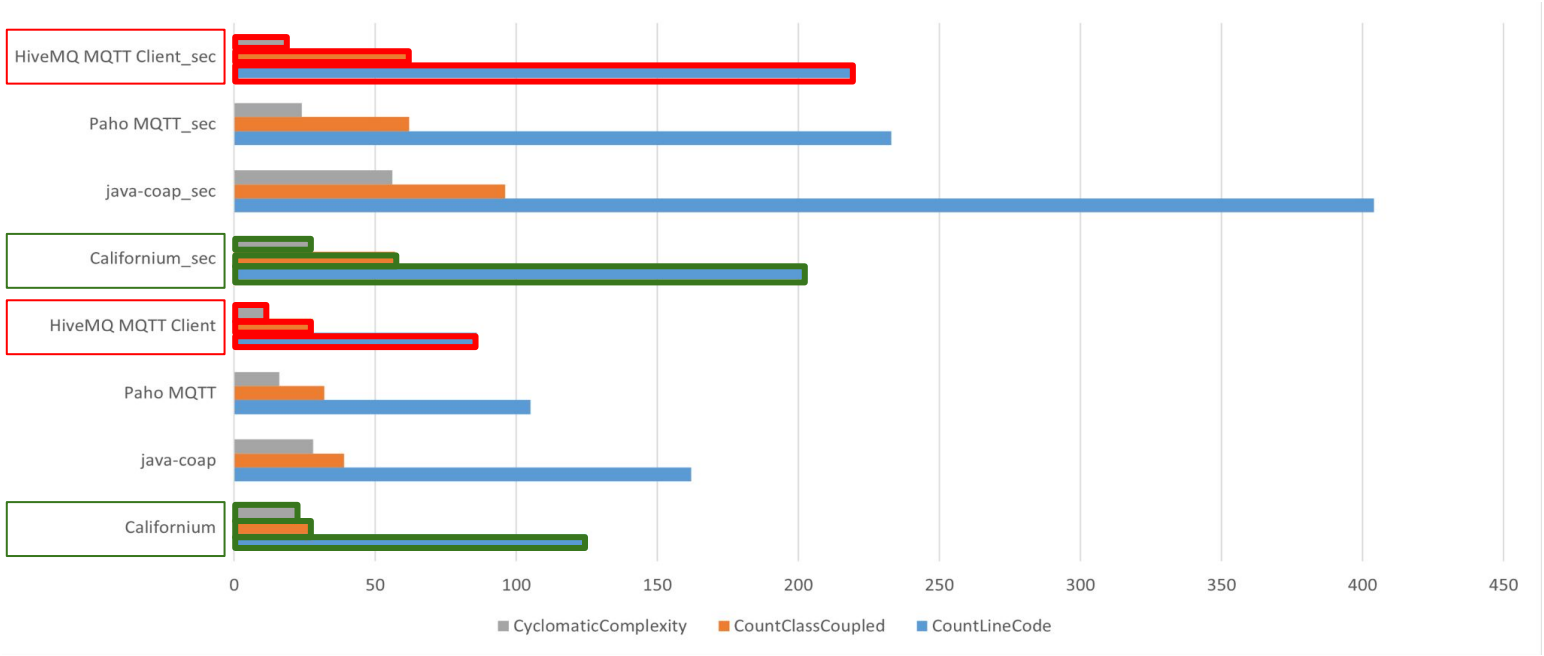
ref:https://github.com/KeithLaiKB/java_learn_coap_comparison
ref:https://github.com/KeithLaiKB/java_learn_mqtt_comparison

Approach

1. Survey and select different libraries of the protocols
2. Set up an IoT infrastructure
3. Collect, devise, and implement scenarios using the chosen libraries
4. **Collect static metrics on the implemented senders and receivers**
5. Analyse packet sizes theoretically and practically
6. Collect runtime performances when running the scenarios
7. Compare the collected measures and recommend a protocol/library

4. Collect static metrics on the implemented senders and receivers

Smaller is better



ref:https://support.scitools.com/support/solutions/articles/70000582223-what-metrics-does-understand-have-2

Approach

1. Survey and select different libraries of the protocols
2. Set up an IoT infrastructure
3. Collect, devise, and implement scenarios using the chosen libraries
4. Collect static metrics on the implemented senders and receivers
5. **Analyse packet sizes theoretically and practically**
6. Collect runtime performances when running the scenarios
7. Compare the collected measures and recommend a protocol/library

5. packet sizes theoretically and practically

Example for CoAP result (practically):

By default, Token just uses 2 bytes in java-coap , but 8 bytes in Californium
If Token length is ignored, **Californium** (left) is slightly smaller than **java-coap** (right)

CoAP
Constrained Application
Protocol

Library		Californium	java-coap
CON	Content is ""	$15 + m$	$10 + m$
	With content	$16 + y + m$	$11 + y + m$

Example for MQTT result (practically):

By default, **Paho MQTT** (left) is slightly smaller than **HiveMQ MQTT Client** (right)

Library			Paho MQTT	HiveMQ MQTT Client
CONNECT			$19 + msgl + p_1 + m$	Same
CONNACK			$3 + msgl + p_1 + m$	Same
PUBLISH_QoS0	Pub	1st Msg	$6 + msgl + p_1 + tpn + m$	Same
		2nd Msg	$6 + msgl + p_1 + m$	Same
	Sub	1st Msg	$3 + msgl + p_1 + tpn + m$	$3 + msgl + p_1 + tpn + m + (1 + sbid)$
		2nd Msg	$3 + msgl + p_1 + tpn + m$	$3 + msgl + p_1 + tpn + m + (1 + sbid)$
PUBLISH_QoS1	Pub	1st Msg	$8 + msgl + p_1 + tpn + m$	Same
		2nd Msg	$8 + msgl + p_1 + m$	Same
	Sub	1st Msg	$5 + msgl + p_1 + tpn + m$	$5 + msgl + p_1 + tpn + m + (1 + sbid)$
		2nd Msg	$5 + msgl + p_1 + tpn + m$	$5 + msgl + p_1 + tpn + m + (1 + sbid)$

5. packet sizes theoretically and practically

Theoretical packet sizes and adjusted practical packet sizes

CoAP
Constrained Application
Protocol

		Theoretical	Adjusted practical
CON	**Content is ""**	$4+e$ $+(1+(opn_1*op_1+opn_2*op_2+opn_3*op_3))*opgn$ $+y$	$7+e+m$
Explanation	$e \in [0,8]$ means token $opn_1 \in [0,1]$ means the number of OptionDelta (Extended) $op_1 \in [0,2]$ means OptionDelta (Extended) $opn_2 \in [0,1]$ means the number of OptionLength (Extended) $op_2 \in [0,2]$ means OptionLength (Extended) $opn_3 \in [0,1]$ means the number of OptionValue $op_3 \in [0,a)$ means OptionValue a means the maximum length of OptionValue is limited by the protocol $opgn$ means the number of group of the Option, it is undefined number limited by the protocol, starting at 0. $y \in [0,i)$ means payload i means maximum length is limited by the protocol m means the number of bytes of other unnecessary attributes and unnecessary value z is undefined number, starting at 0		

	Theoretical	Adjusted practical
PUBLISH_QoS0	$3+msgl+tpn+p_1$ $+n_1*(1+4)+n_2*(1+2)+n_3*(1+1)$ $+n_4*(upgn*(1+(2+k_1)+(2+k_2)))$ $+n_5*(1+(2+k_1))+n_6*(1+sbid)$ $+y$	$4+msgl+p_1+m$
Explanation	$k1 \in [0,65535]$, $k2 \in [0,65535]$, k1 and k2 means the content of the specific attribute $y \in [0,i)$ means payload i means maximum length is limited by the protocol $upgn$ means the number of group of the User Properties, it is limited by the protocol $msgl \in [1,4]$ means message length $p_1 \in [1,4]$, $p_2 \in [0,4]$, p_1 and p_2 both mean property length $tpn \in [0,a)$ means the topic name $sbid \in [1,4]$ means the value of Subscription Identifier $n_1 \in [0,1]$ means Message Expiry Interval $n_2 \in [0,1]$ means Topic Alias $n_3 \in [0,1]$ means Payload Format Indicator $n_4 \in [0,1]$ and $n_5 \in [0,3]$ both mean different groups of attributes with same bytes $n_6 \in [0,1]$ means Subscription Identifier $n_7 \in [0,1]$, means reason code $n_8 \in [0,1]$, means reason string a means the maximum length of topic(topic name) is limited by the protocol m means the number of bytes of other unnecessary attributes and unnecessary value	

5. packet sizes theoretically and practically

Example for adjusted practical for QoS1/CON scenario

	MQTT			CoAP
	Pub to Broker	**Sub to Broker**	**Total**	**Server to Client**
PUB/CON	$6 + msgl[1,4] + p1[1,4] + m$	$6 + msgl[1,4] + p1[1,4] + m$	$(6+1+1)+(6+1+1) = 16$	$7 + e1[0,8]$
ACK	$3 + msgl[1,4] + p2[0,4] + m$	$3 + msgl[1,4] + p2[0,4] + m$	$(3+1+0)+(3+1+0) = 8$	$4 + e1[0,8] + z$
Total	$(6+1+1)+(3+1+0) = 12$	$(6+1+1)+(3+1+0) = 12$	24	$7 + 4 =$ 11

Approach

1. Survey and select different libraries of the protocols
2. Set up an IoT infrastructure
3. Collect, devise, and implement scenarios using the chosen libraries
4. Collect static metrics on the implemented senders and receivers
5. Analyse packet sizes theoretically and practically
6. **Collect runtime performances when running the scenarios**
7. Compare the collected measures and recommend a protocol/library

6. Collect runtime performances when running the scenarios

CoAP
Constrained Application Protocol

		Goodput	Throughput	Overhead
Reliability	**NON**	negligible	irregular	irregular
	CON	java-coap	irregular	irregular
Security	**Non-sec**	java-coap	Californium	java-coap (slightly)
	Sec	negligible	java-coap	Californium

		Goodput	Throughput	Overhead
Reliability	**QoS0**	irregular	HiveMQ MQTT Client	negligible
	QoS1	HiveMQ MQTT Client	HiveMQ MQTT Client	negligible
Security	**Non-Sec**	irregular	HiveMQ MQTT Client	Paho MQTT (slightly)
	Sec	HiveMQ MQTT Client	HiveMQ MQTT Client	HiveMQ MQTT Client (slightly)

Approach

1. Survey and select different libraries of the protocols
2. Set up an IoT infrastructure
3. Collect, devise, and implement scenarios using the chosen libraries
4. Collect static metrics on the implemented senders and receivers
5. Analyse packet sizes theoretically and practically
6. Collect runtime performances when running the scenarios
7. **Compare the collected measures and recommend a protocol/library**

7.Compare the collected measures and recommend a protocol/library

Static metrics

HiveMQ MQTT Client is recommended

1. LOC in non-secure scenario of **HiveMQ MQTT Client** is much less than **Californium**
2. CC in non-secure and secure scenarios of **HiveMQ MQTT Client** is less than **Californium**

Packets metrics

Californium is recommended

1. Eight bytes of tokens by default, but the library is actively maintained, it is possible to change the token, compared to **java-coap**

7.Compare the collected measures and recommend a protocol/library

Runtime metrics

CoAP and its Californium are recommended

1. **CoAP** is faster than **MQTT**
2. Overhead metric in **CoAP** is lower than **MQTT**
3. Among the four libraries in this experiment, in most scenarios, **java-coap** is the fastest, followed by **Californium**

Summary

1. In all, based on our experiment, in pursuit of fast communication, we highly recommend using **Californium**
2. While in pursuit of the advantage at the coding level refers to the static metrics, we highly recommend using **HiveMQ MQTT Client**

Attention: Throughput is higher but since the goodput is lower, it is not an improvement

Discussion

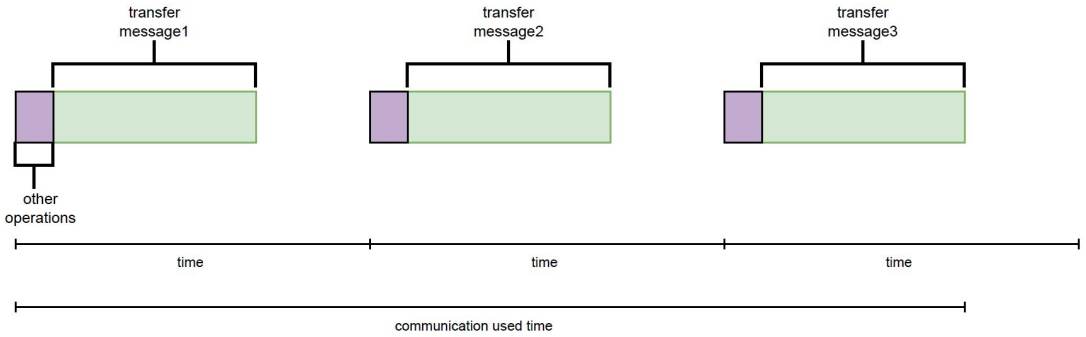
NON/QoS0 vs. CON/QoS1

Comparison of goodput is similar based on the **non-continuous traffic**.

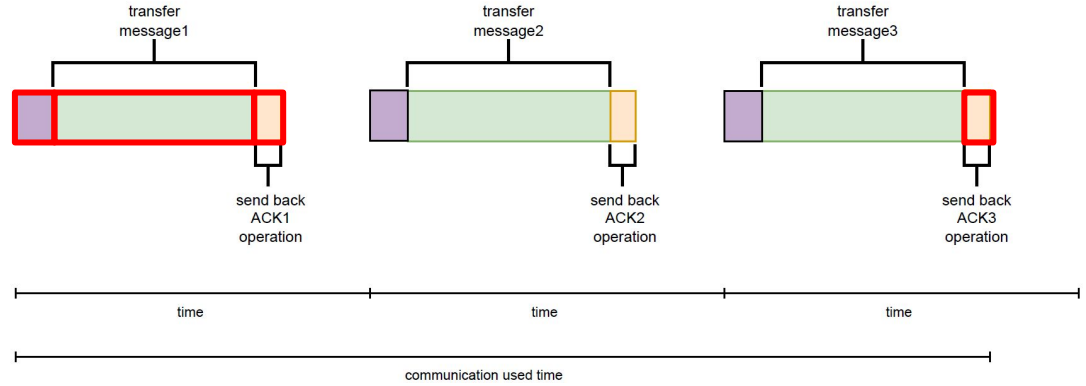
However, lots of IoT devices lead to network congestion

- Overhead metric in CON/QoS1 is greater than in NON/QoS0

NON



CON



Discussion

Non-secure scenario vs. Secure scenario

Non-secure scenario is faster

1. because creating the connection securely will take time
2. the secure cases need more overhead in each packet

However, the decision to employ a secure mode depends on the specific user requirements

Conclusion

Approach

We selected the protocols and libraries, collected, devised, and implemented scenarios
Collected static metrics: HiveMQ MQTT Client
Collected packets metrics (Recommended): Californium
Collected runtime metrics (Recommended): Californium

HiveMQ MQTT Client is recommended

1. Easy to configure
2. Libraries is actively in maintained in the community

Future Work

1. Select more libraries even with different programing languages and different protocols
2. Set a better infrastructure to try more scenarios such as different delays in network
3. Apply more metrics

A Comparison of IoT Communication Libraries: APIs and Performances

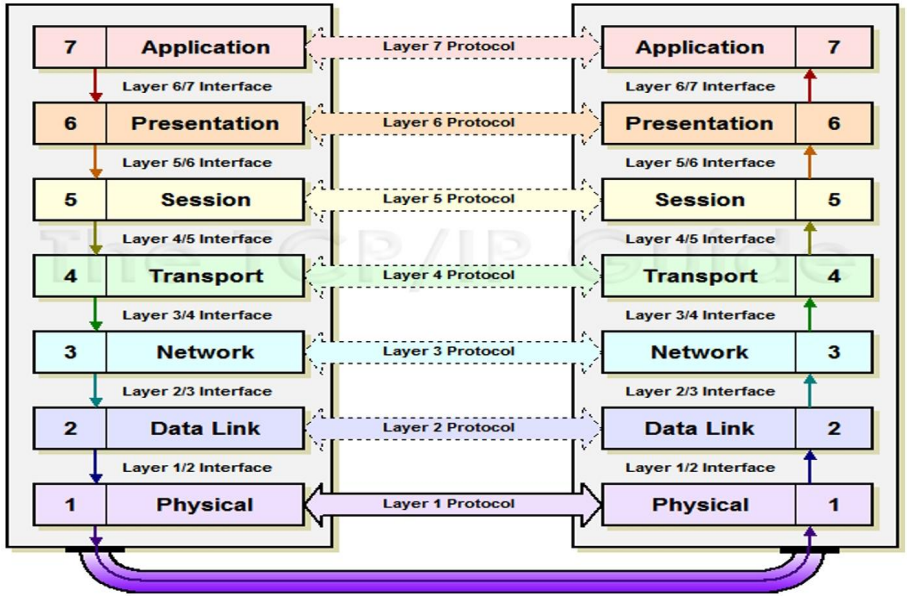
Thank you!

Jianbin Lai
Concordia University

Supervisor: Sandra Céspedes,Yann-Gaël Guéhéneuc

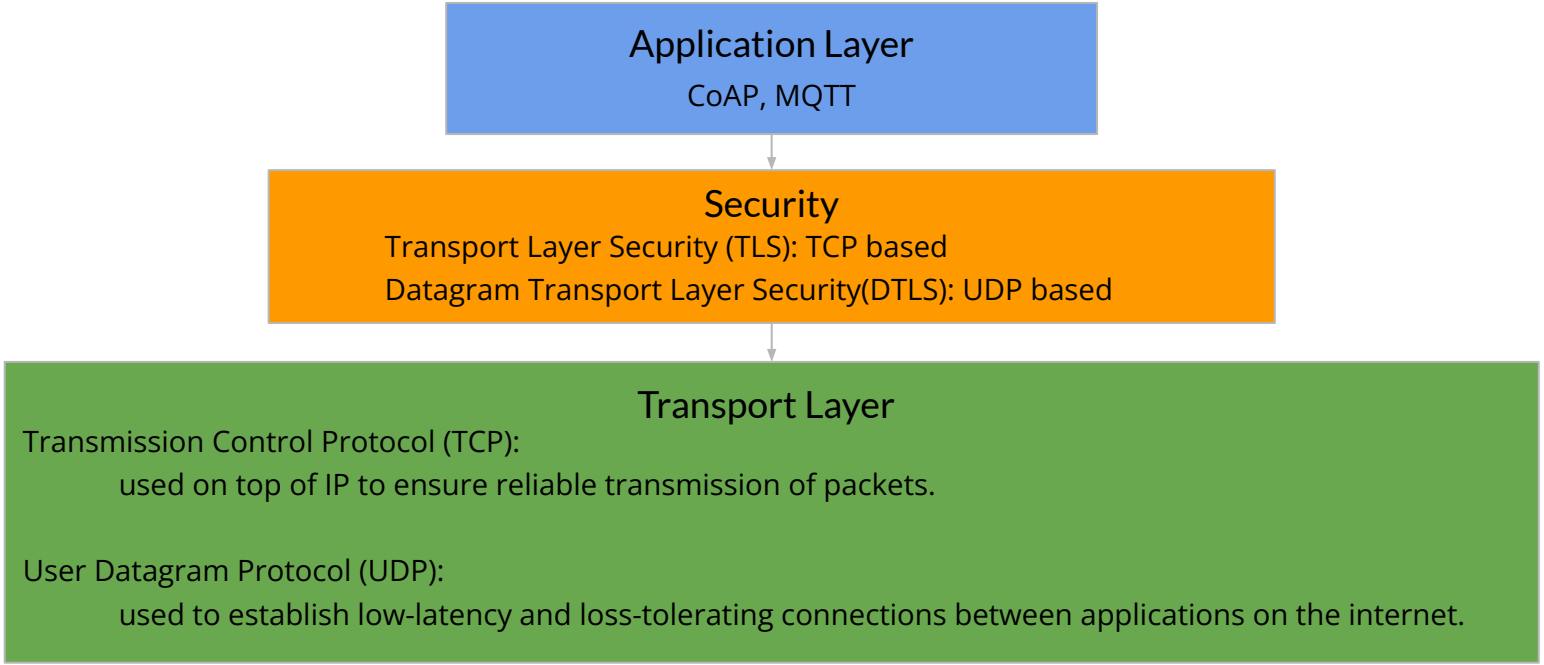
Background

OSI Model



Kumar, Sumit and Dalal, Sumit and Dixit, Vivek. The OSI model: Overview on the seven layers of computer networks. International Journal of Computer Science and Information Technology Research, 2(3):461–466, 2014

Background



ref:https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:the-internet/xcae6f4a7ff015e7d:transporting-packets/a/transmission-control-protocol--tcp
ref:https://www.techtarget.com/searchnetworking/definition/UDP-User-Datagram-Protocol

2. Set up an IoT infrastructure

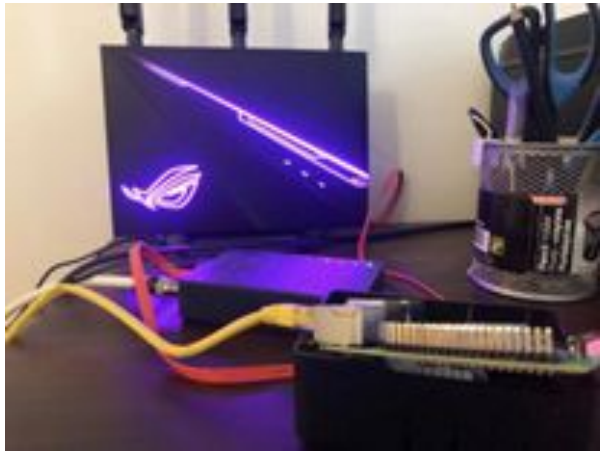
1 Raspberry Pi 4B, 4G RAM, Raspbian OS, 64bit
2 Raspberry Pi 4B, 8G RAM, Raspbian OS, 64bit

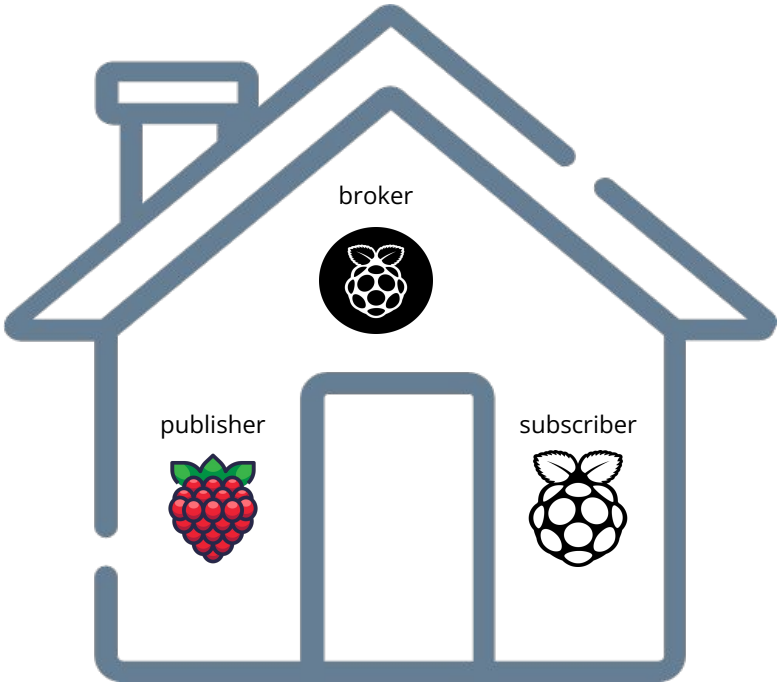
Asus ROG AC2900C router

Before testing throughput, shutdown the Internet.

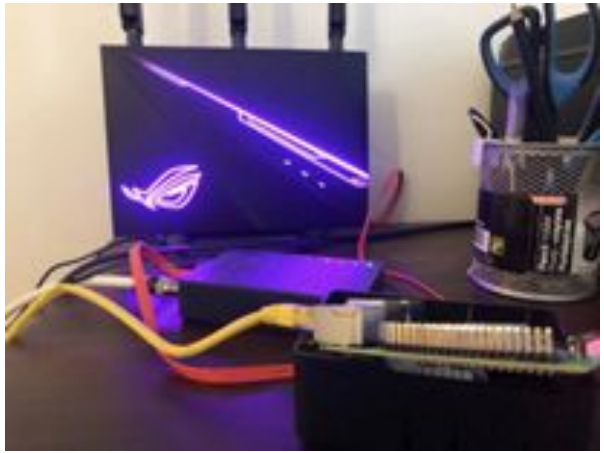
- CoAP
  - one client and one server running on two separate Raspberry Pi boards (wireless).
- MQTT
  - one subscriber and one publisher running on two separate Raspberry Pi boards (wireless).
  - The broker runs on a **third Raspberry Pi wired connected to router**.

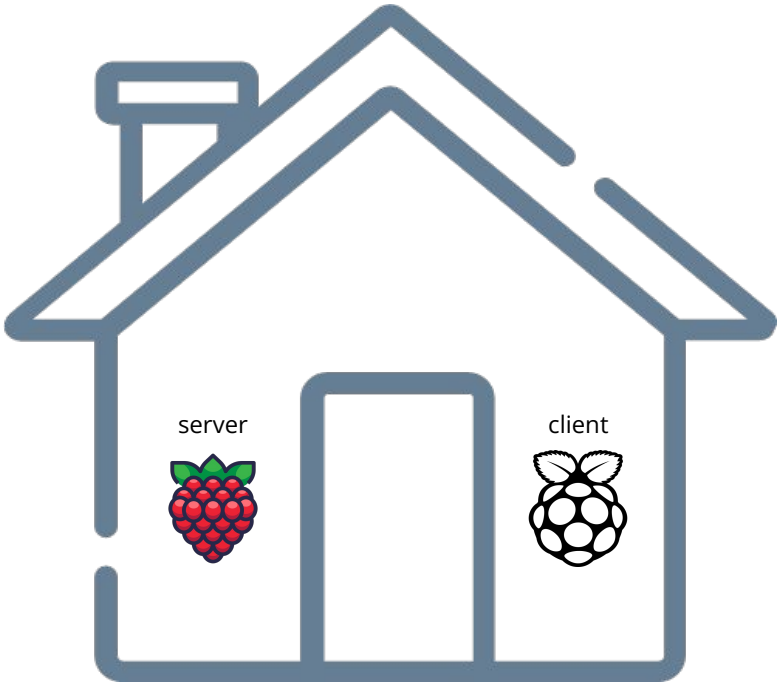
Set up an IoT infrastructure





Set up an IoT infrastructure





Related Work

Comparison of APIs

1. [19] APIDiff: Detecting API breaking changes (2018)
   ⇒*two versions* of same code, but we compare different pieces of code from different projects

2. [20] An Efficient Similarity Comparison Based on Core API Calls(2013)
   ⇒ the evaluation metrics are difficult to apply to our work

Related Work

About CoAP and MQTT

1. [21] Lithe: Lightweight Secure CoAP for the Internet of Things (2013)
   ⇒Compression solution because DTLS is power consuming,
   but we do not compress header, we choose to compare the packet

2. [22] Californium: Scalable cloud services for the Internet of Things with CoAP (2014)
   ⇒present architecture and evaluate the architecture
   but we do not evaluate the architecture, we choose to evaluate the CoAP API

3. [23] Internet of Things: Survey and open issues of MQTT protocol (2017)
   ⇒This paper presents the basic information of the Message Queuing Telemetry Transport (MQTT) protocol.
   But we focus more on comparison API in MQTT.

4. [24] IoT real time data acquisition using MQTT protocol (2017)
   ⇒ an implementation on MQTT(Wemos D1 Mini, PC as broker)
   We use 3 raspberry pi, to imitate the scenarios

Related Work

Comparison of CoAP and MQTT

1. [5] A Comparative analysis of MQTT and IoT application protocols (2021)
2. [13] Comparing energy consumption of application layer protocols on IoT devices (2021)
3. [7] Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP (2017)
4. [25] Performance evaluation of MQTT and CoAP via a common middleware (2014)
5. [26] IoT network protocols comparison for the purpose of IoT constrained networks (2017)

⇒made the comparison related to CoAP and MQTT, some of them talk about network condition.
However, they did not compare the code of the different implementations in different protocols

Overall

1. API not related to IoT [19][20]
2. Some of them just survey or compare the protocols but not related to APIs [23][5][7]
3. Some of them just use **one implementation** in MQTT to compare **with one implementation** in CoAP[13][25][26]
   `Lack of thesis about the comparison of **implementation** of different **protocols** related to **APIs**'

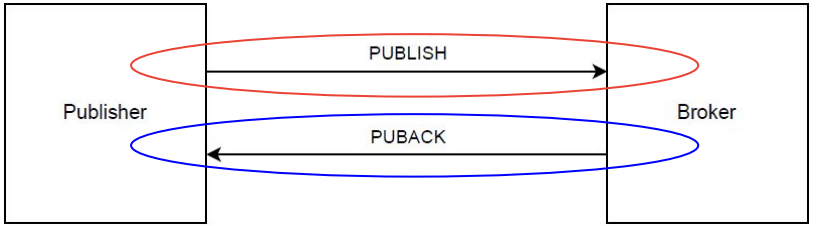
Approach

1. Survey and select different libraries of the protocols
2. Set up an IoT infrastructure
3. Collect, devise, and implement scenarios using the chosen libraries
4. Collect static metrics on the implemented senders and receivers
5. Analyse packet sizes theoretically and practically
6. Collect runtime performances when running the scenarios
7. Compare the collected measures and recommend a protocol/library

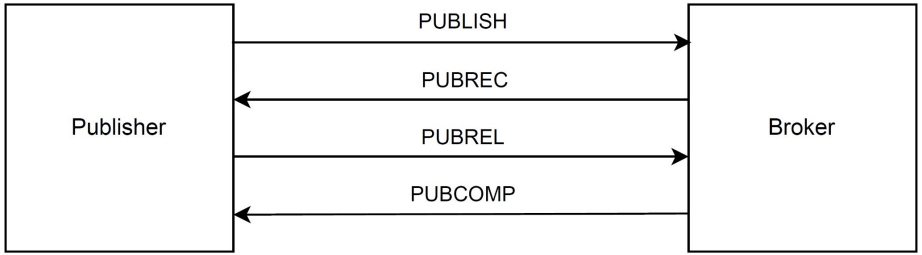
Background

MQTT-QoS0

MQTT-QoS1



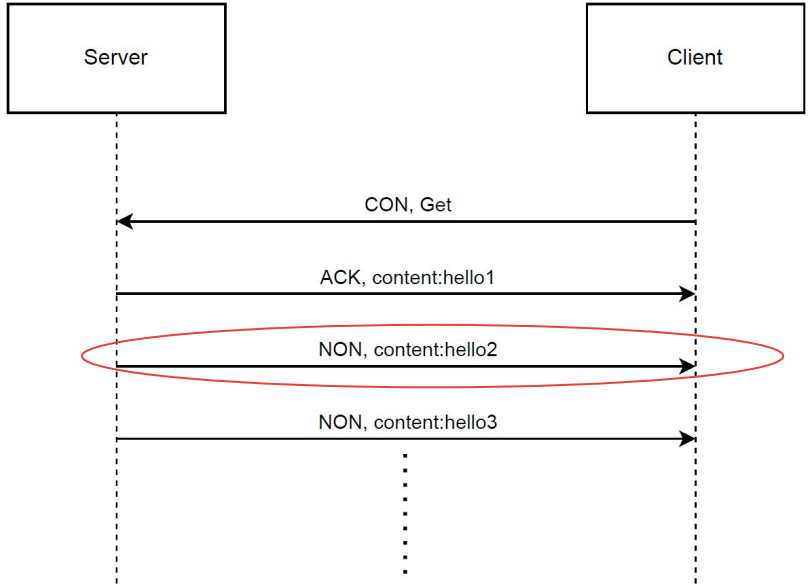
MQTT-QoS2



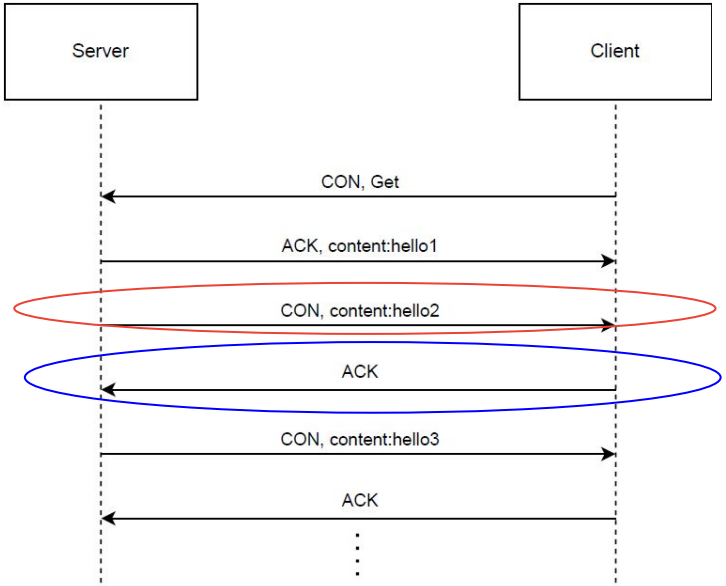
ref:https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

Background

CoAP observing Non-confirmable(NON) resource



CoAP observing Confirmable(CON) resource



ref:https://www.rfc-editor.org/rfc/rfc7252

Collect static metrics on the implemented senders and receivers

Library	Metric	Sever	Client	Total
Californium	LOC	92	33	125
	CBO	17	10	27
	CC	16	6	22
java-coap	LOC	102	60	162
	CBO	20	19	39
	CC	18	10	28
HiveMQ MQTT Client	LOC	40	46	86
	CBO	13	15	28
	CC	4	7	11
Paho MQTT	LOC	35	70	105
	CBO	9	23	32
	CC	4	12	16

(a) Non-secure Scenario

Library	Metric	Sever	Client	Total
Californium	LOC	135	68	203
	CBO	33	24	57
	CC	19	8	27
java-coap	LOC	282	122	404
	CBO	62	34	96
	CC	42	14	56
HiveMQ MQTT Client	LOC	107	113	220
	CBO	31	31	62
	CC	8	11	19
Paho MQTT	LOC	99	134	233
	CBO	24	38	62
	CC	8	16	24

(b) Secure Scenario

Table 5.8: LOC, CBO, and CC metrics in non-secure and secure scenarios

Library	Side	Class	LCOM
Californium	Server	Launcher	0
		Resource	35
		UpdateTask	50
	Client	Launcher	50
java-coap	Server	Launcher	0
		Resource	35
		UpdateTask	50
	Client	Launcher	50
HiveMQ MQTT Client	Server	Launcher	0
	Client	Launcher	66
Paho MQTT	Server	Launcher	0
	Client	Launcher	66
		MsgCallBack	0

(a) Non-secure Scenario

Library	Side	Class	LCOM
Californium	Server	Launcher	0
		Resource	35
		UpdateTask	50
	Client	Launcher	50
java-coap	Server	Launcher	0
		Resource	35
		UpdateTask	50
		ExtraTransportCfg	62
	Client	Launcher	50
HiveMQ MQTT Client	Server	Launcher	0
	Client	Launcher	66
Paho MQTT	Server	Launcher	0
	Client	Launcher	66
		MsgCallBack	0

(b) Secure Scenario

Table 5.9: LCOM in non-secure and secure scenarios

Analyse packet sizes theoretically and practically

Example for CoAP result (practically):

By default, Token just uses 2 bytes in java-coap , but 8 bytes in Californium
If Token length is ignored, **Californium** is slightly smaller than **java-coap**

Library		Californium	java-coap
CON	Content is ""	$15 + m$	$10 + m$
	With content	$16 + y + m$	$11 + y + m$

Example for MQTT result (practically):

By default, **Paho MQTT**(left) is slightly smaller than **HiveMQ MQTT Client** (right)

Library			Paho MQTT	HiveMQ MQTT Client
CONNECT			$19 + msgl + p_1 + m$	Same
CONNACK			$3 + msgl + p_1 + m$	Same
PUBLISH_QoS0	Pub	1st Msg	$6 + msgl + p_1 + tpn + m$	Same
		2nd Msg	$6 + msgl + p_1 + m$	Same
	Sub	1st Msg	$3 + msgl + p_1 + tpn + m$	$3 + msgl + p_1 + tpn + m + (1 + sbid)$
		2nd Msg	$3 + msgl + p_1 + tpn + m$	$3 + msgl + p_1 + tpn + m + (1 + sbid)$
PUBLISH_QoS1	Pub	1st Msg	$8 + msgl + p_1 + tpn + m$	Same
		2nd Msg	$8 + msgl + p_1 + m$	Same
	Sub	1st Msg	$5 + msgl + p_1 + tpn + m$	$5 + msgl + p_1 + tpn + m + (1 + sbid)$
		2nd Msg	$5 + msgl + p_1 + tpn + m$	$5 + msgl + p_1 + tpn + m + (1 + sbid)$

Analyse packet sizes theoretically and practically

theoretical counting and applied theoretical counting

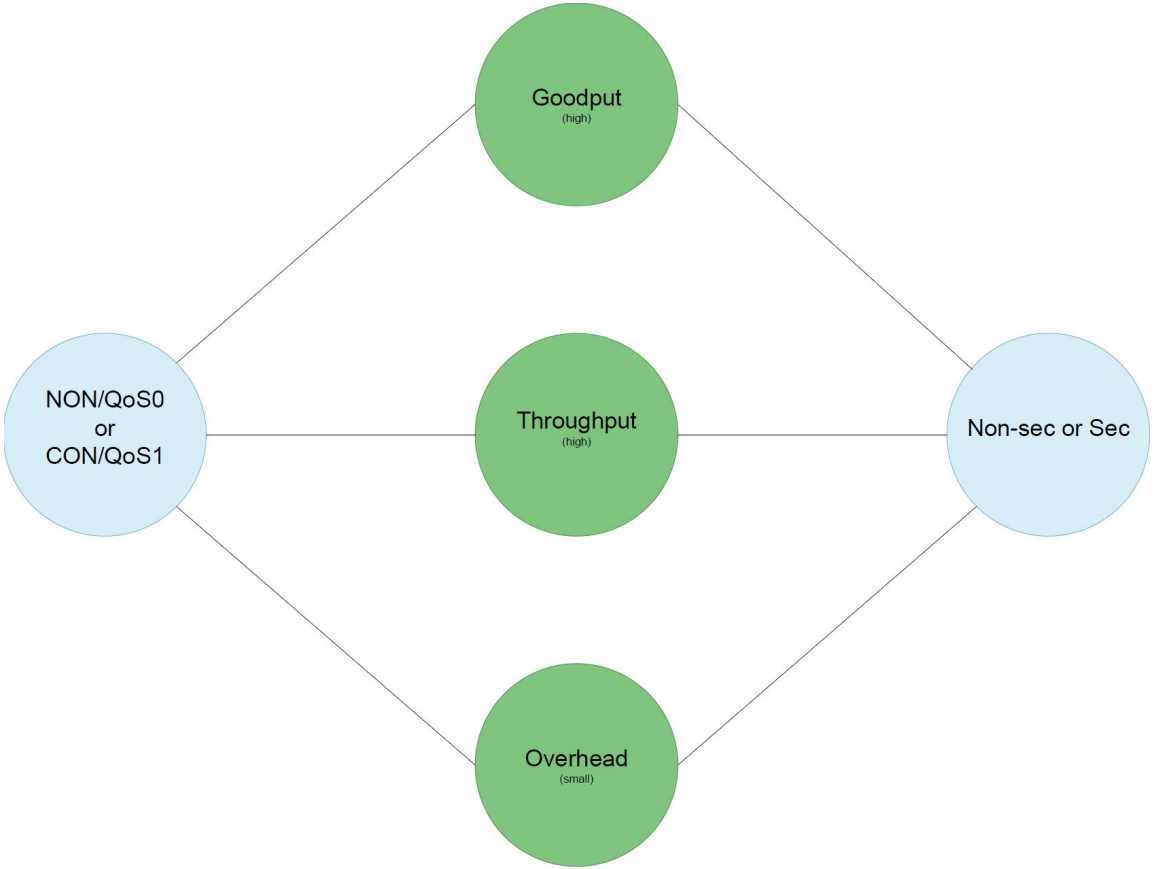
Example for CoAP

		Theoretical	Adjusted practical
CON	**Content is ""**	$4+e$ $+(1+(opn_1 * op_1 + opn_2 * op_2 + opn_3 * op_3)) * opgn$ $+y$	$7+e+m$
Explanation	$e \in [0,8]$ means token $opn_1 \in [0,1]$ means the number of OptionDelta (Extended) $op_1 \in [0,2]$ means OptionDelta (Extended) $opn_2 \in [0,1]$ means the number of OptionLength (Extended) $op_2 \in [0,2]$ means OptionLength (Extended) $opn_3 \in [0,1]$ means the number of OptionValue $op_3 \in [0,a)$ means OptionValue a means the maximum length of OptionValue is limited by the protocol $opgn$ means the number of group of the Option, it is undefined number limited by the protocol, starting at 0. $y \in [0,i)$ means payload i means maximum length is limited by the protocol m means the number of bytes of other unnecessary attributes and unnecessary value z is undefined number, starting at 0		

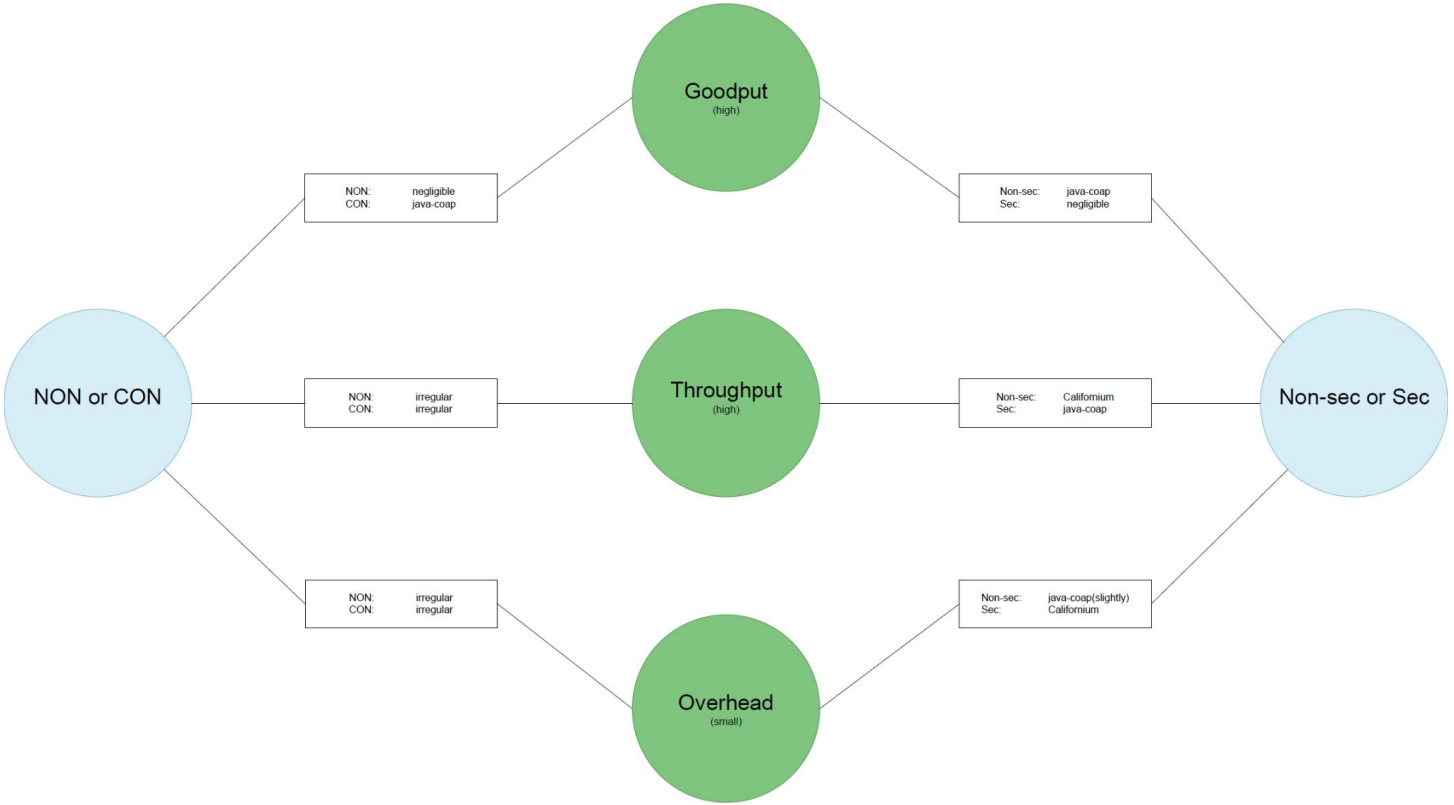
Example for MQTT

	Theoretical	Adjusted practical
PUBLISH_QoS0	$3+msgl+tpn+p_1$ $+n_1*(1+4)+n_2*(1+2)+n_3*(1+1)$ $+n_4*(upgn*(1+(2+k_1)+(2+k_2)))$ $+n_5*(1+(2+k_1))+n_6*(1+sbid)$ $+y$	$4+msgl+p_1+m$
Explanation	$k1 \in [0,65535]$, $k2 \in [0,65535]$, k1 and k2 means the content of the specific attribute $y \in [0,i)$ means payload i means maximum length is limited by the protocol $upgn$ means the number of group of the User Properties, it is limited by the protocol $msgl \in [1,4]$ means message length $p_1 \in [1,4]$, $p_2 \in [0,4]$, p_1 and p_2 both mean property length $tpn \in [0,a)$ means the topic name $sbid \in [1,4]$ means the value of Subscription Identifier $n_1 \in [0,1]$ means Message Expiry Interval $n_2 \in [0,1]$ means Topic Alias $n_3 \in [0,1]$ means Payload Format Indicator $n_4 \in [0,1]$ and $n_5 \in [0,3]$ both mean different groups of attributes with same bytes $n_6 \in [0,1]$ means Subscription Identifier $n_7 \in [0,1]$, means reason code $n_8 \in [0,1]$, means reason string a means the maximum length of topic(topic name) is limited by the protocol m means the number of bytes of other unnecessary attributes and unnecessary value	

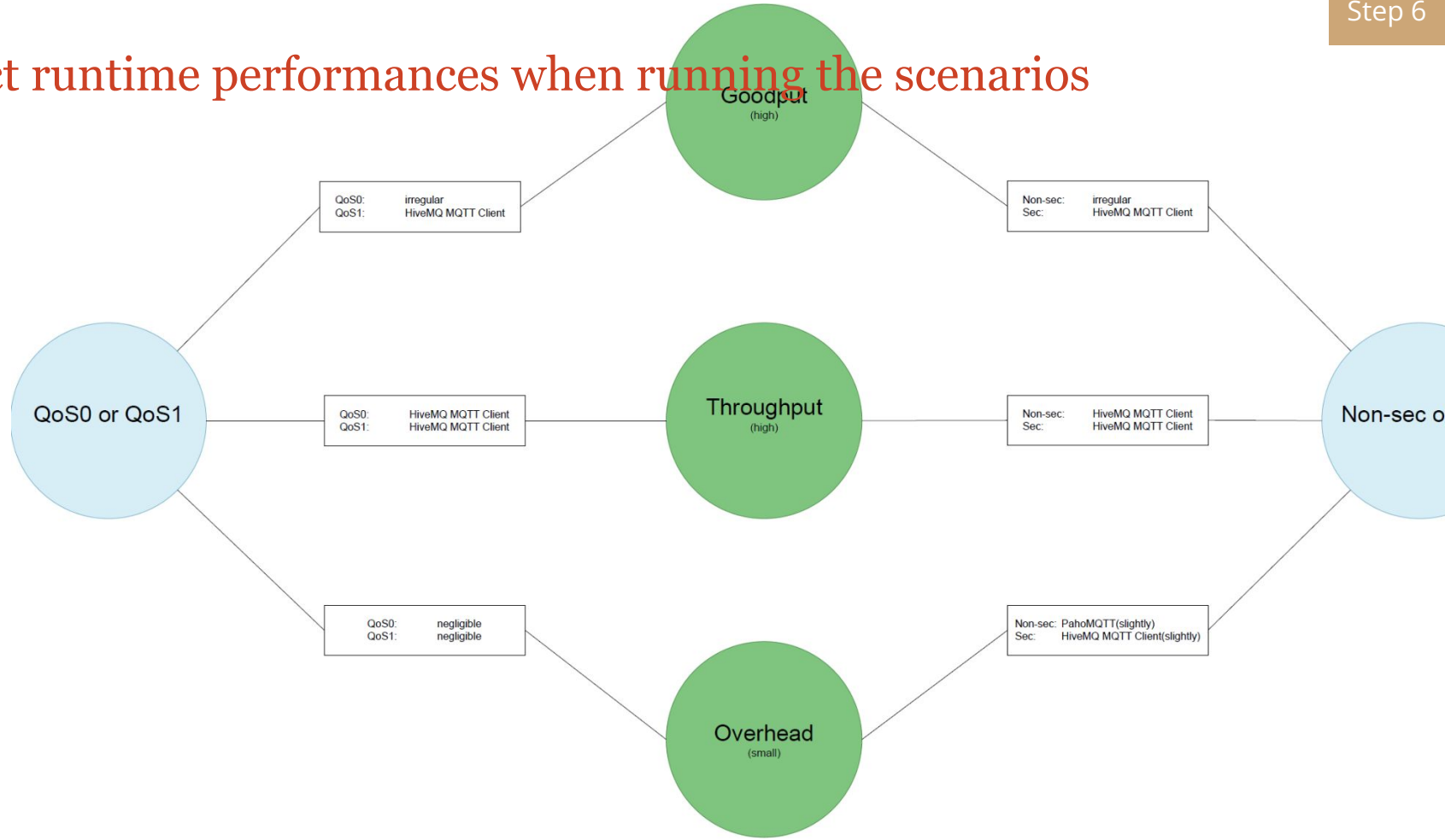
Collect runtime performances when running the scenarios



Collect runtime performances when running the scenarios



Collect runtime performances when running the scenarios



Compare the collected measures and recommend a protocol/library

Static metrics:

- **CoAP**
  - LOC, CC, and CBO in **Californium** is less than **java-coap**

  Conclusion: Californium is better

- **MQTT**
  - LOC, CC in **HiveMQ MQTT Client** is less than **Paho MQTT**

  Conclusion: HiveMQ MQTT Client is better

Overall

HiveMQ MQTT Client is recommended

1. LOC in non-secure scenario of **HiveMQ MQTT Client** shows much less than **Californium**
2. CC in non-secure and secure scenarios of **HiveMQ MQTT Client** show less than **Californium**

Attention: Throughput is higher but since the goodput is lower, it is not an improvement

Compare the collected measures and recommend a protocol/library

Packets metrics:

- The two libraries in **CoAP** use less packets than MQTT
- Because two processes (publisher to broker, and broker to subscriber
- From the tables, it shows that **java-coap** uses the least packet size among the four libraries, followed by **Californium**

Overall

Californium is recommended

1. 8 tokens by default, but the library is active, it is possible to change the token

Attention: Throughput is higher but since the goodput is lower, it is not an improvement

Compare the collected measures and recommend a protocol/library

Runtime metrics:

- **CoAP** is faster than MQTT
- Overhead metric in **CoAP** shows lower in the metric
- Among the four libraries in this experiment, in most scenarios, **java-coap** is the fastest, followed by **Californium**

Summary

1. In all, based on our experiment, in pursuit of fast communication, we highly recommend using **Californium**
2. While in pursuit of the advantage at the coding level refers to the static metrics, we highly recommend using **HiveMQ MQTT Client**

Attention: Throughput is higher but since the goodput is lower, it is not an improvement