OCF-T2TRG-W3C WoT joint meeting

March 23rd, 2018, Prague

OCF-T2TRG-W3C WoT meeting topics

1. Status update of IETF/OCF coordination action items
2. WISHI activities report
3. Representation format of forms for IoT devices
4. Model interoperability: WoT Thing Descriptions with OCF and IPSO models
5. Push Models for Device data (pub/sub, non-traditional responses, ...)
6. Links, Bindings, Interfaces, and Resource Representations
7. "Base" CoAP (New Response Codes, RD alignment)
8. ACE discussion continuation (multipoint security, etc.)
9. Reference implementations and test case (CoAP, CoAP TCP, CBOR, ...)

Repo: https://github.com/t2trg/2018-03-ocf-wot

1 Action Item Review

Action Items 1/2

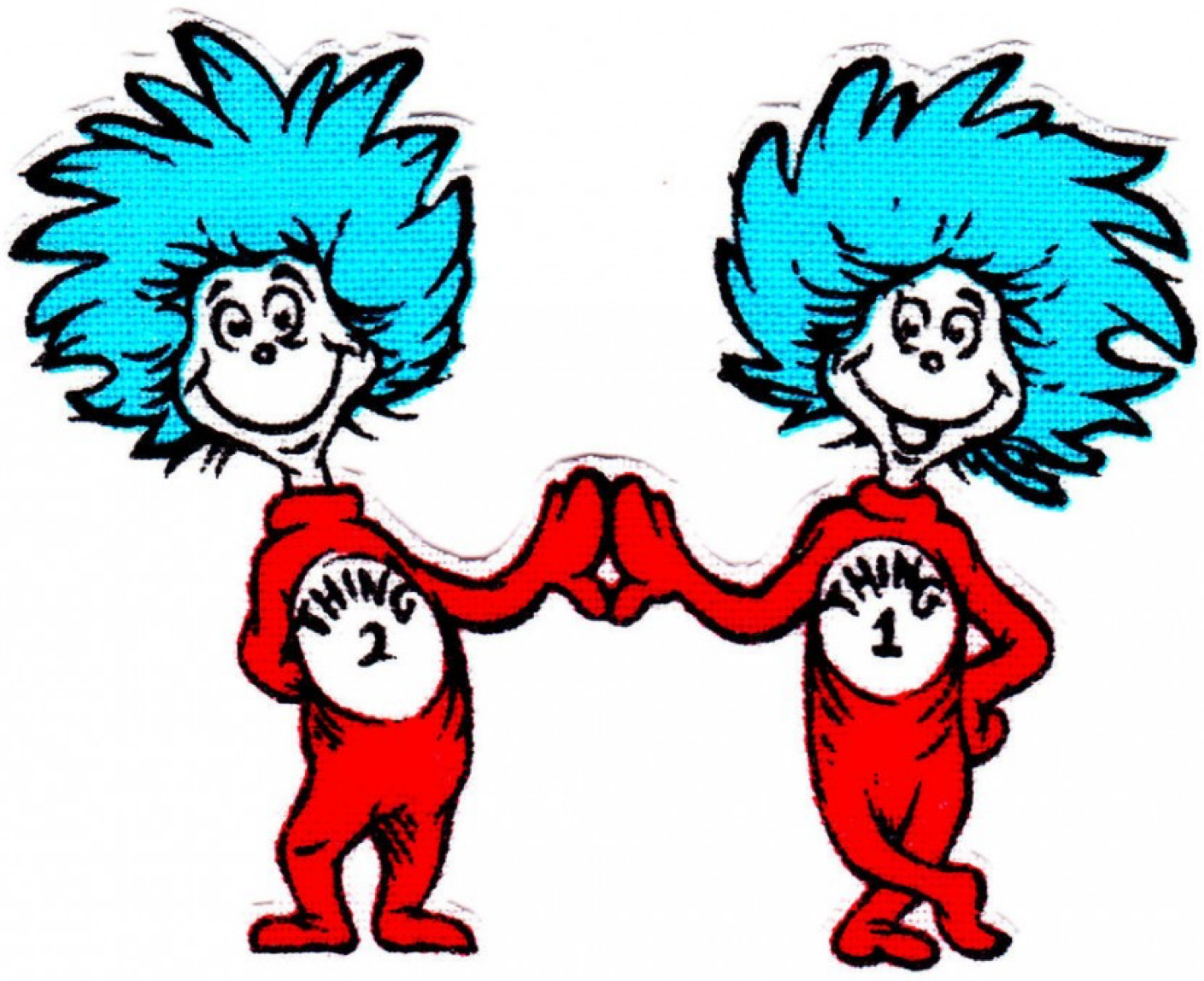
- Setting up monthly calls; topic per call
    - ACE call in January; more TBD
- OCF review of Resource Directory draft
- Design patterns for cloud rendezvous
- Operational guidelines for TCP with TURN
- OneIoTa model notifications for reviews

Action Items 2/2

- Deep-dive on how to use ACE for constrained OCF devices
  - Started in the January call
- Modifying link collections; PATCH format(s)
  - PATCH/FETCH being defined for SenML; more generic considerations planned to be defined
- OSCORE tunnelling to prevent traffic analysis
- Resource read forbidden by definition / forbidden by policy
- Atomic measurements and CoRAL / HSML; bundling
- Re-rendezvous when server knows something broken
- Publish dependent work items at the IETF

2 WISHI

Michael Koster + Chairs


THING 2
THING 1

Workshop on **I**oT **S**emantic/**H**ypermedia **I**nteroperability

- 8 calls after the IETF 100 WISHI hackathon - topics include:
  - Review of existing semantic capabilities - IPSO/LWM2M, OCF, Web links, W3C WoT, CORAL
  - Semantic annotation with RDF ontologies
  - Using third party vocabularies (QUDT, SOSA, SSN, iotschema)
  - Design patterns for semantic metadata integration
  - Layered semantic stack, e.g. SenML + RDF
  - The nature of abstract semantics; high level data and interaction models
  - Binding abstract semantics to concrete protocols
  - Semantic annotation for the context of connected things, features of interest
  - Extend the collaboration with other SDOs around semantic interoperability
  - Planning for IETF Hackathons
  - Interoperability across data types and engineering units

Wishi Hackathon at IETF 101 - High Level Goals

- Bring diverse connected things to interoperate
- Start with directory based discovery using semantic annotation
- Experiment with software adaptation to data models and protocols
- CoAP, HTTP, MQTT protocols
- Simple application scenarios; turn on a light with a motion sensor
- First hands-on experiments

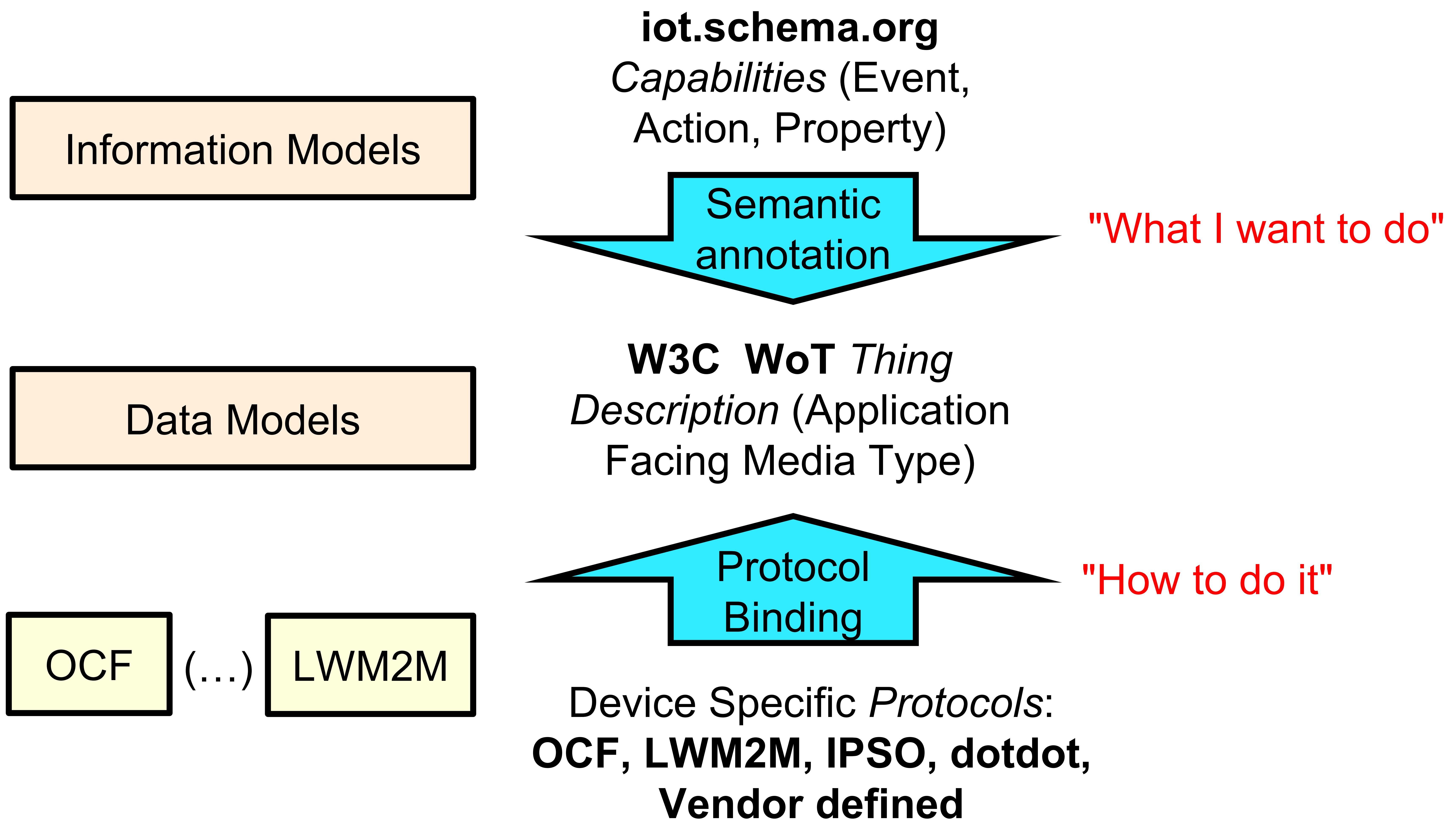
Wishi Hackathon at IETF 101

- What implementations do we have?
  - W3C WoT
  - YANG & CoMI
  - OMA LwM2M Managed Server and Client
  - Ad-hoc device APIs and data models
  - Connected home and automotive domains
- How can we make them work together?
  - Role of W3C Web of Things technology
- Results
  - Thing Descriptions (TD) generated from LwM2M management server
  - CoMI implementation described by YANG re-described with TD
  - TDs stored in Thing Directory, and consumed by WoT implementation
  - WoT implementation communicating directly with all three implementations

W3C Web of Things - Thing Description

- TD is a file format and mediatype of RDF
- Describes abstract Interactions with things
  - Read temperature
  - Lock the door
  - Change the brightness of a light
- Binds to concrete instances that implement the interactions
  - Defines data shape, payload structure
  - Defines data types and range
  - Transfer layer instructions including URI, methods, options
- Applications use abstract interactions to decouple from the underlying implementation
- Any application, any network, any connected thing, using automated software adaptation

Layered Scope in Data Models and Information Models

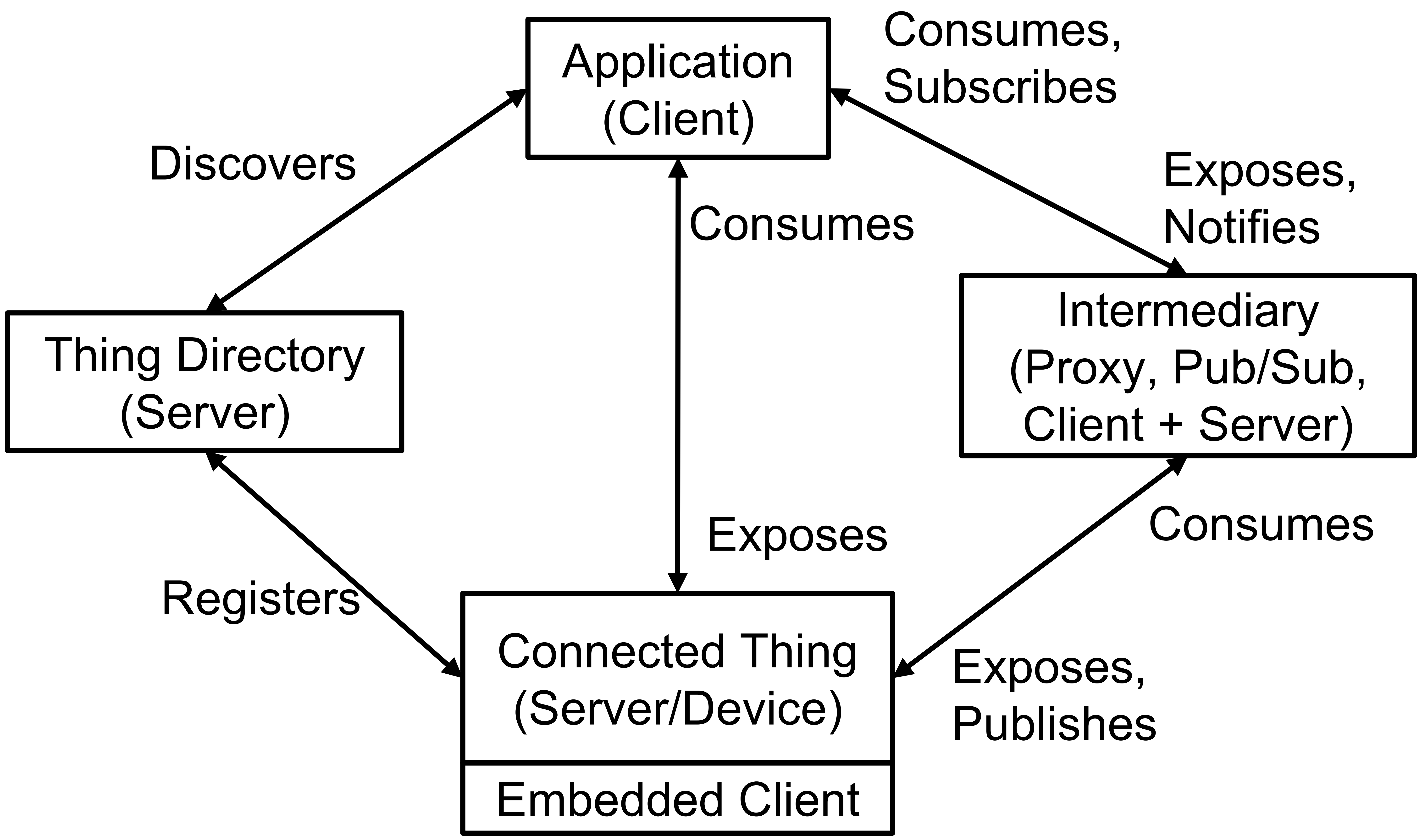


Information Models
Data Models
OCF
(…)
LWM2M
iot.schema.org
Capabilities (Event, Action, Property)
Semantic annotation
"What I want to do"
W3C WoT *Thing Description* (Application Facing Media Type)
Protocol Binding
"How to do it"
Device Specific *Protocols*:
OCF, LWM2M, IPSO, dotdot, Vendor defined

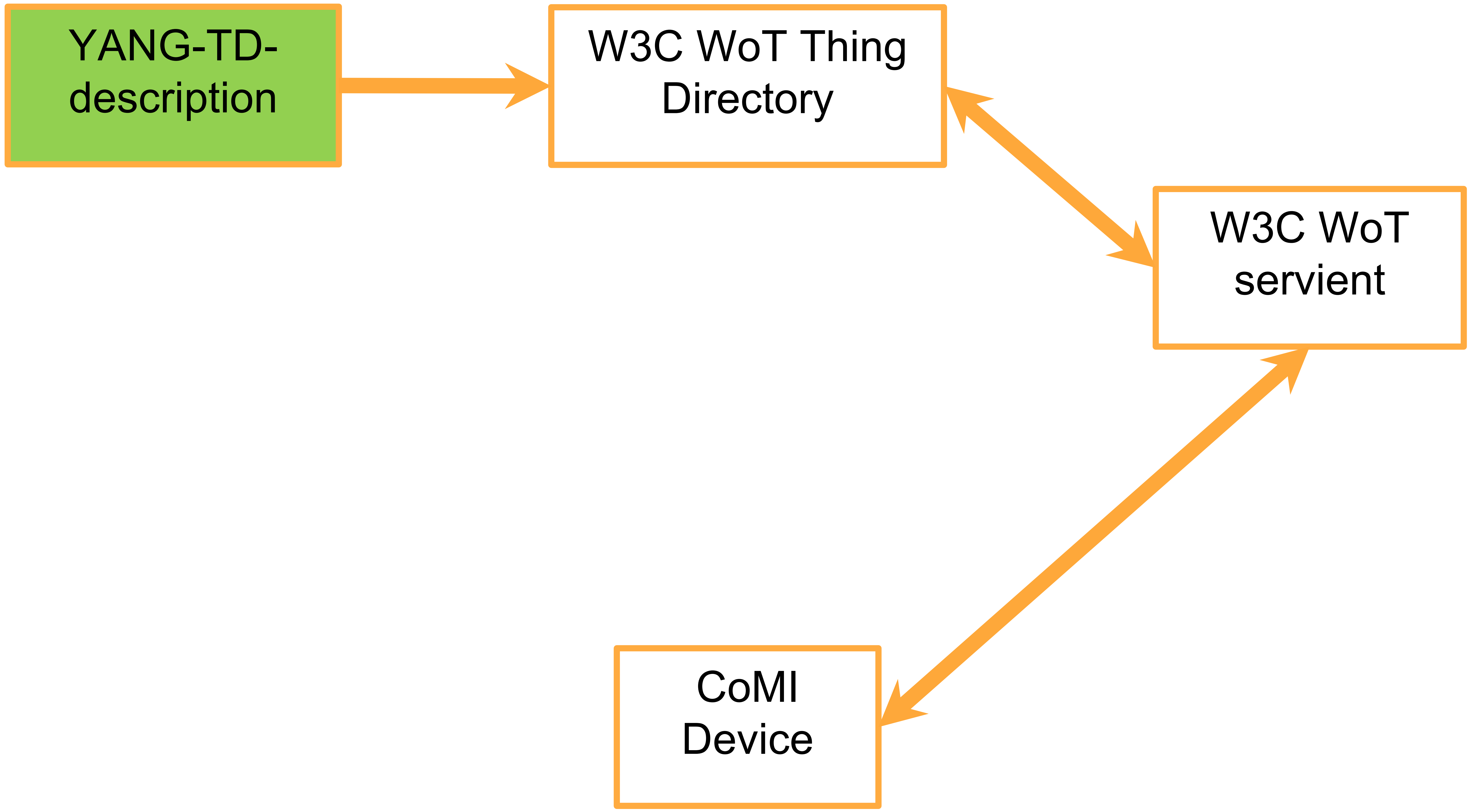
Thing Directory

- Things register their metadata with semantic annotation to a Thing Directory
- Applications can discover the capabilities of registered things based on the semantic annotation
- One or more thing directories with well known entry points
- Semantic discovery and thing integration into the application
  - Submit a semantic query to the Thing Directory indicating the required capability types (temperature measurement, light control, door lock)
  - Retrieve Thing Descriptions from registered things that satisfy the query
  - Select Things to integrate into the application that have the required interactions (e.g. color temperature control for a light)
  - Use the protocol binding part of the TD to construct payloads and perform methods on the desired things

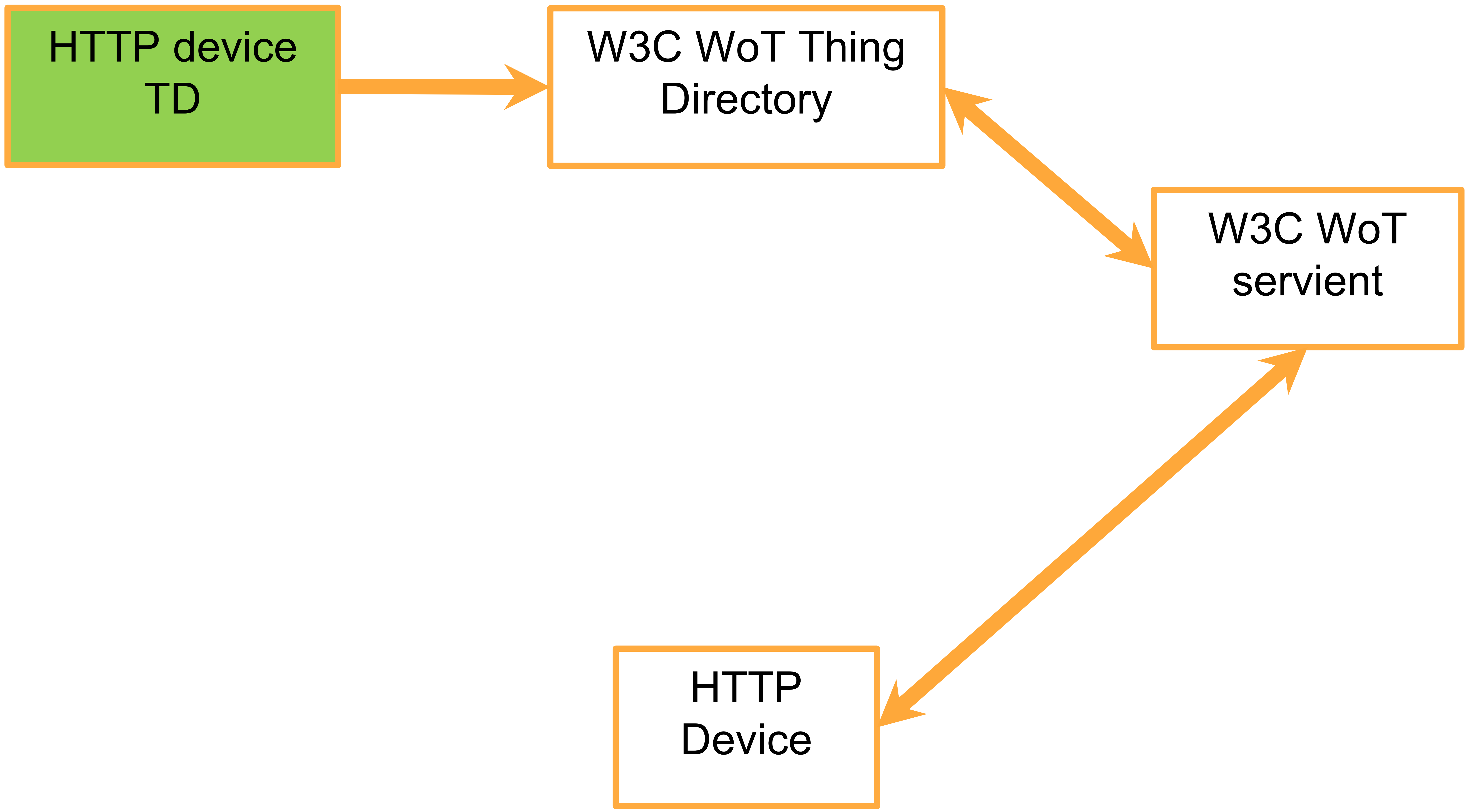
Interop Schematic Diagram – Roles and Interactions



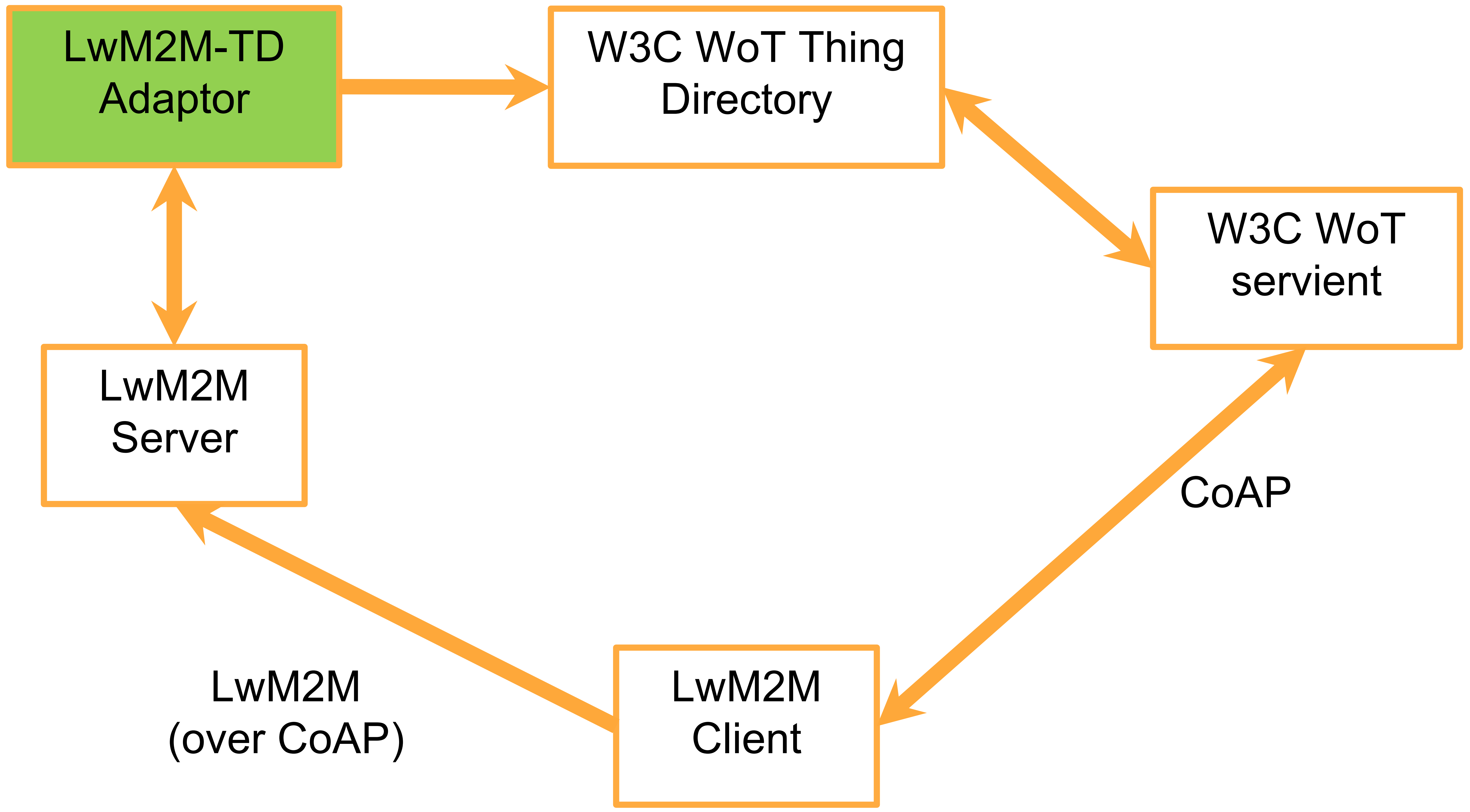
CoMI device integration



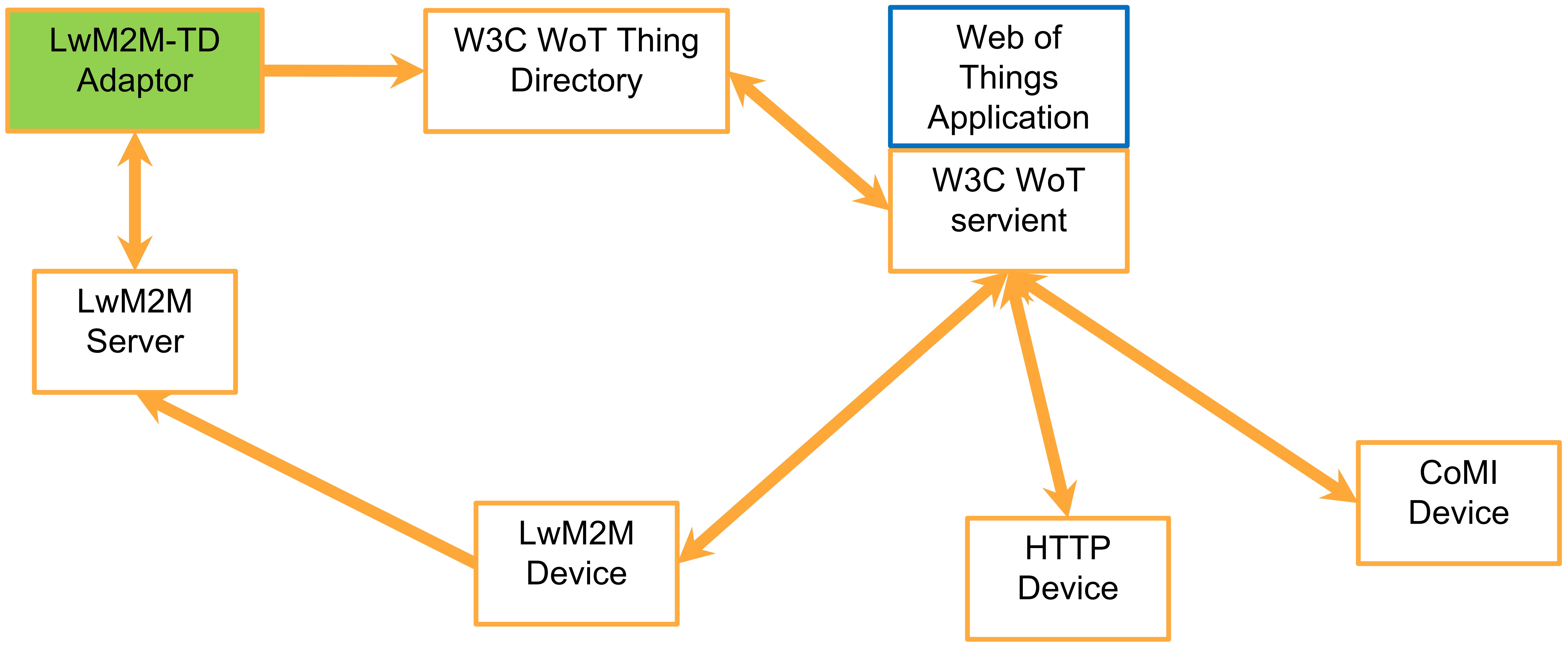
HTTP Device Integration



LwM2M Client & Server integration



Putting things together with



Next Steps

- Experiment with semantic annotation and discovery using CoRE RD and CoRE Link-format
- Experiment with more diverse end device protocols and data models
- More automation of semantic queries - SPARQL syntax from URI-query options
- Automation of protocol bindings using software adaptation or translation
- Binding device capabilities to external Features of Interest (binding an instance of a thing with door lock capability to a specific door of your house) for contextual discovery
- More sophisticated applications, device to device orchestration

3 Forms

- Discuss the concept and implementation of machine forms, for submitting state changes to a device.
- What are some of the emerging patterns and what is the experience to date?
- Can we begin to standardize the design patterns and vocabulary?

Sources of Forms

- Media type with links
    - Link-format (would need target attributes)
    - More general linking format (e.g., CORAL)
    - Specific Application (e.g., via CBOR template tag)

draft-bormann-lpwan-cbor-template

- **variable:** placeholder CBOR data item included in a larger data item (the "CBOR template")

```
  variable<varid> = #6.42(varid)

{ "name": "Carsten Bormann", "place": 42(0) } ➔
{ "name": "Carsten Bormann”, "place": "Bremen" }
```

- Varid can be of any appropriate type (e.g., integer, text string)
- Originally designed for LPWAN SCHC, but can be used in a general way

4 Model Interoperability

- What are the common patterns that we can use for annotation?
- What is the experience of semantic integration to date?
- Discuss Web of Things, Thing Description, and iotschema.

Federated Semantics through

Attaching Foreign Vocabulary

Attaching Foreign Semantics

To open up the closed world of a vocabulary, **foreign vocabulary** can be added in:

— The main vocabulary

— The instance

— The media type of the instance

— The self-description of the Thing

— A **semantic style**

Foreign Semantics in a Vocabulary

Can be easily added in the "T-Box":

— An IPSO temperature **is a** QUDT Celsius Temperature

— An IPSO temperature **is a** UCUM Cel

This requires a place in the vocabulary where that T-Box can be added.
➔ Vocabularies need to be able to refer to other vocabularies.
➔ Can only be done at vocabulary design time

Foreign Semantics in an Instance

Add semantics in the structure of the Instance:

— E.g., HTML and Microformats

This requires a place in the instance where extensible information can be added.
➔ Instance developer needs to be aware about all applicable foreign semantics
➔ Instance may get large if the developer embraces many of these (optimizations?)

Foreign Semantics in the Media Type

Pretty much equivalent to adding foreign semantics to the vocabulary of that media type

Foreign Semantics in the Self-Description

Add semantics to self-description ("type information")

— E.g., in WoT TD

Requires type information in self-description to be able to reach into resource structure as well as the internal structure of resources

Only works for semantic information that changes at the time constants of changes to the self-description ("metadata")

Foreign Semantics in a Semantic Style

Attach information via **selectors** into the instance

— Similar to adding style semantics to HTML via CSS

Requires **selector language** to generically (or specifically!) identify internal structure of the resource

(This could also be used to add foreign semantics to a self-description or even a media type)

Conclusion 2017-12-18

We probably need all of these.

Work on:

— Seamlessly moving between on structure **built out of** resources and structure **within** resources

— Attaching semantic information to structured data; **selector** languages for the IoT

Somewhat related example provided by Klaus Hartke:

```
#using <http://example.org/wishi#>
#using ipso = <http://example.org/ipso/vocabulary/>

ipso:Object3303 </temp1> {
    is-a <http://iot.linkeddata.es/def/wot#Thing>
    is-a <http://iotschema.org/TemperatureCapability>

    ipso:Item5700 </temp1/value> {
        is-a <http://iot.linkeddata.es/def/wot#Property>
        is-a <http://iotschema.org/TemperatureProperty>

        media-type "application/senml+json" {
            field "$[*].v" {  // JSONPath
                is-a <http://iotschema.org/TemperatureData>
                type "number"
                minimum -50
                maximum 100
                unit <http://qudt.org/vocab/unit/DEG_C>
                unit <http://unitsofmeasure.org/ucum.html#Cel>
            }
        }
    }
}
```

Approaches

Transformation language (DSSSL/XSLT)

Augmentation language (CSS)

Rules

selector ➔ effect

```
.warning {color: red}
```

Selector

Selects zero or more structural elements into a "node set"

— HTML Elements in CSS

— XML Elements, Attributes or other parts of the ESIS in XSLT and XPath

Effect is then applied to each selection

Effect

Add attribute to each selection

— kept in a separate namespace by CSS (properties)

Generate additional structure (even in augmentation)

— CSS: `::before`, `::after`…

Can involve computation (e.g., counters)

Value spaces?

Selection components

Navigational

— E.g., XSLT/XPath axes (ancestor ancestor-or-self attribute child descendant descendant-or-self following following-sibling namespace parent preceding preceding-sibling self @ // .. .)

Type-based, Attribute-based

— XML GI, XML classes, ids, other attributes

Computational, inference-based

Computational, inference-based

- `:odd`, `:even`, … ➔ structural computation
- can the effect of other rules be used?
- Full "T-boxes" as in OWL

5 Push Models for Device Data

- Pub/Sub: What is the roadmap for CoRE Pub/Sub? What are the remaining issues from the OCF perspective?
- What are the system design patterns around use of pub/sub? How do server devices publish to a broker?
- How does pub/sub help with the firewall traversal issues?
- Non-traditional response message (draft-bormann-core-responses-00)

Push of what?

- Fresh data for a resource? ✔ (Observe)
- Series of data items?
  - “Events” (discrete, such as sales)
  - “Time series” (where the period is somewhat arbitrary)
- What is important?
  - Do not lose “Events”, vs.
  - Minimize latency

The Series Transfer Pattern (STP): draft-bormann-t2trg-stp

- Model a series as a collection
    - Array of data
    - Array of links (with metadata)
- GET can yield the current collection
- GET more? Where was I… “cursor”
- Result (response, notification) can set MORE bit (i.e., not everything fit)
- Request can set WAIT bit (i.e., don’t just return empty array)

Custody transfer

- Who is responsible for keeping the data?
- Traditional collection: server keeps all
- Things may need to “garbage collect” items in series
  - When can the Thing be sure data has arrived?
  - Could use cursor with a custody flag (= I’ve got all, up to cursor)
  - Custody transfer with multiple clients?

OPEN CONNECTIVITY FOUNDATION™

push model Notification

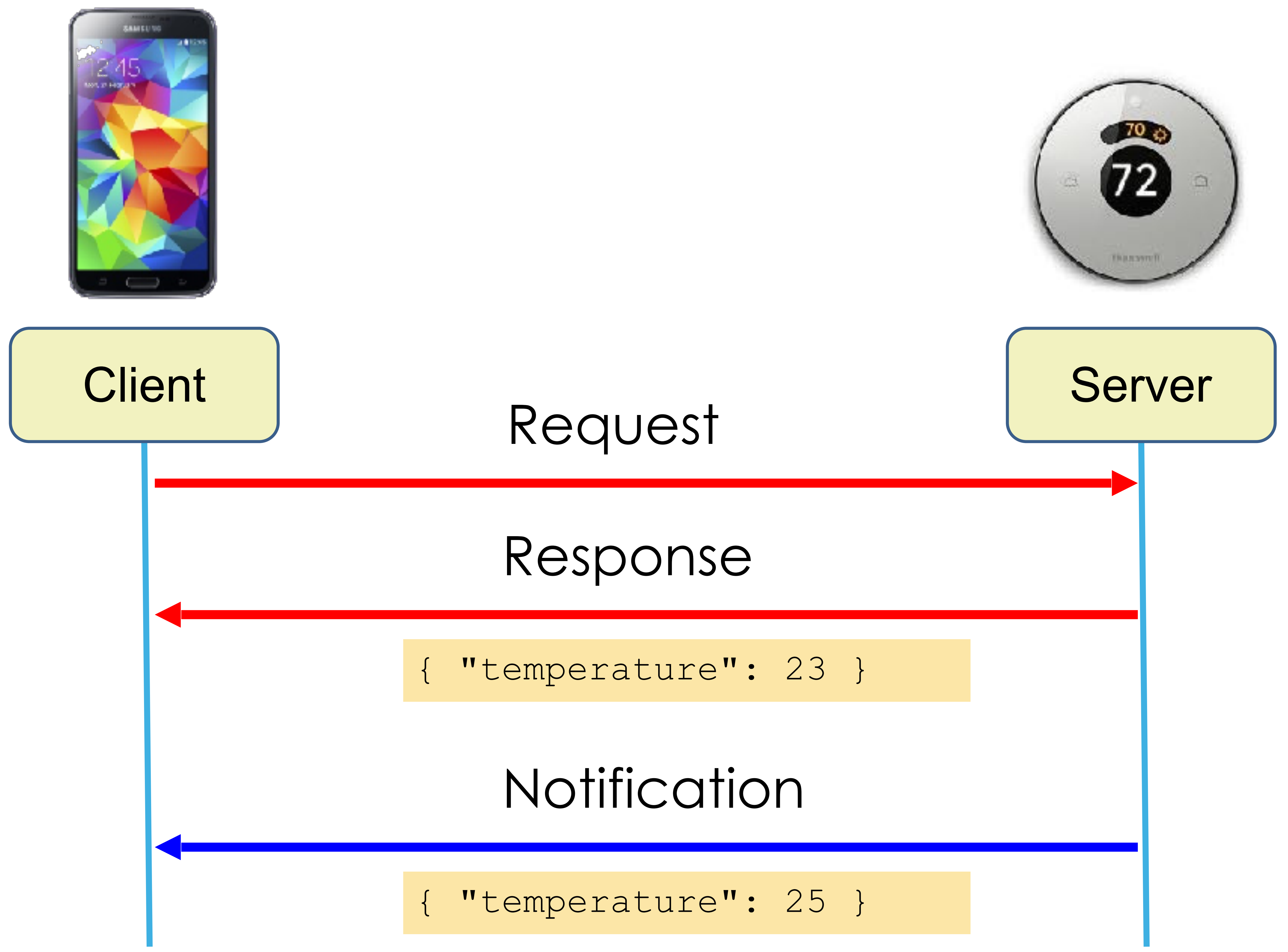
- Non-traditional response (draft-bormann-core-responses)

21th March 2018

JinHyeock Choi

Notification overview

- Notification*
  - A message is sent by a Server without an (exactly matching) Request



```
{ "temperature": 23 }
```

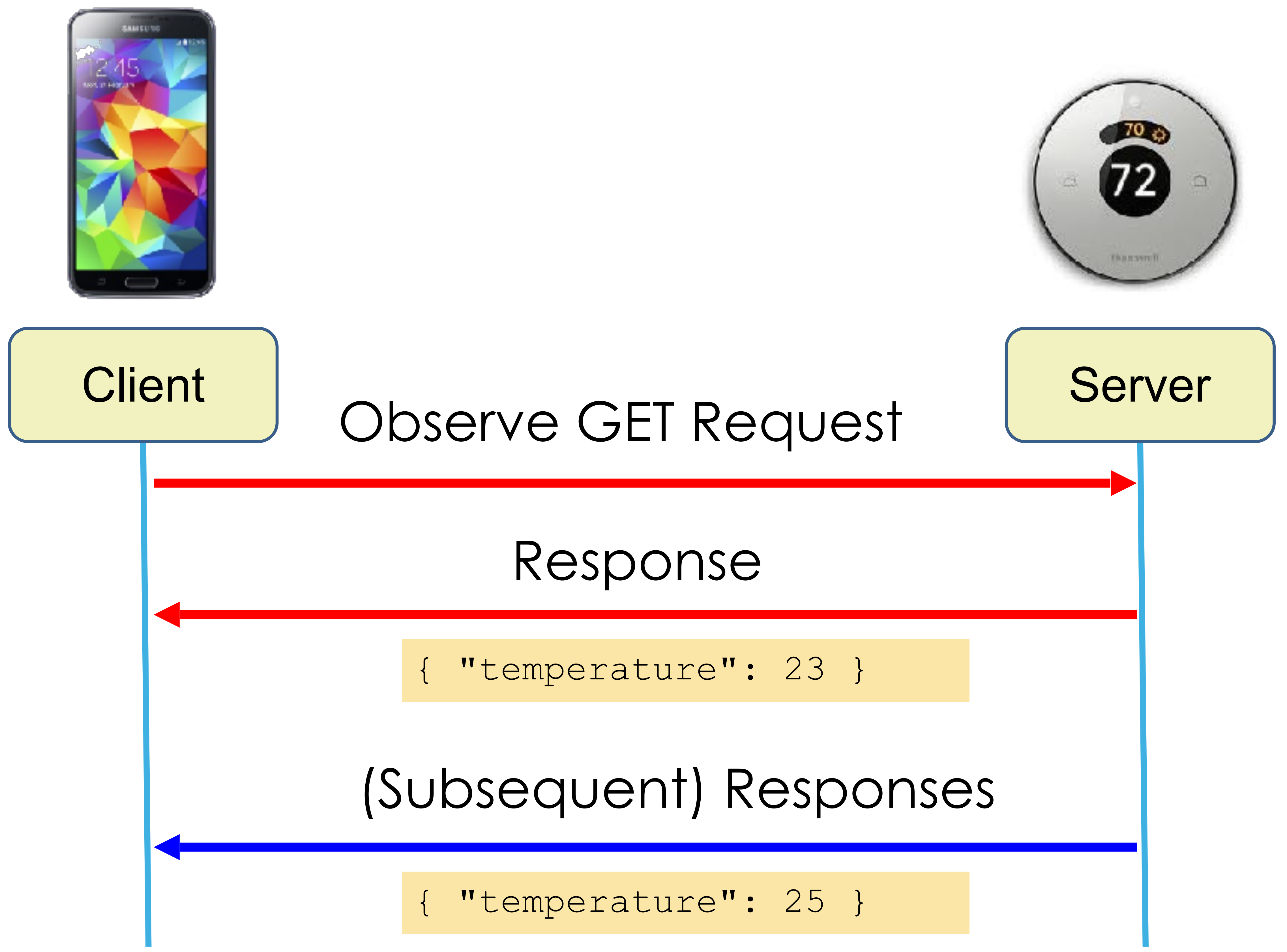
```
{ "temperature": 25 }
```

Notification overview

- 2 kinds of Notification
  - Subscribed notification
    - A client makes a priori subscription for a target resource to the hosting server.
    - e.g.) CoAP Observe (RFC 7641)
  - Unsubscribed notification, aka, push model Notification
    - No a priori subscription for the target resource.
    - Sometimes, it's hard to know where to send a subscription beforehand.
    - Ex) Fire alarm or Vehicular network emergency

Notification overview

- 2 kinds of Notification
  - Subscribed notification

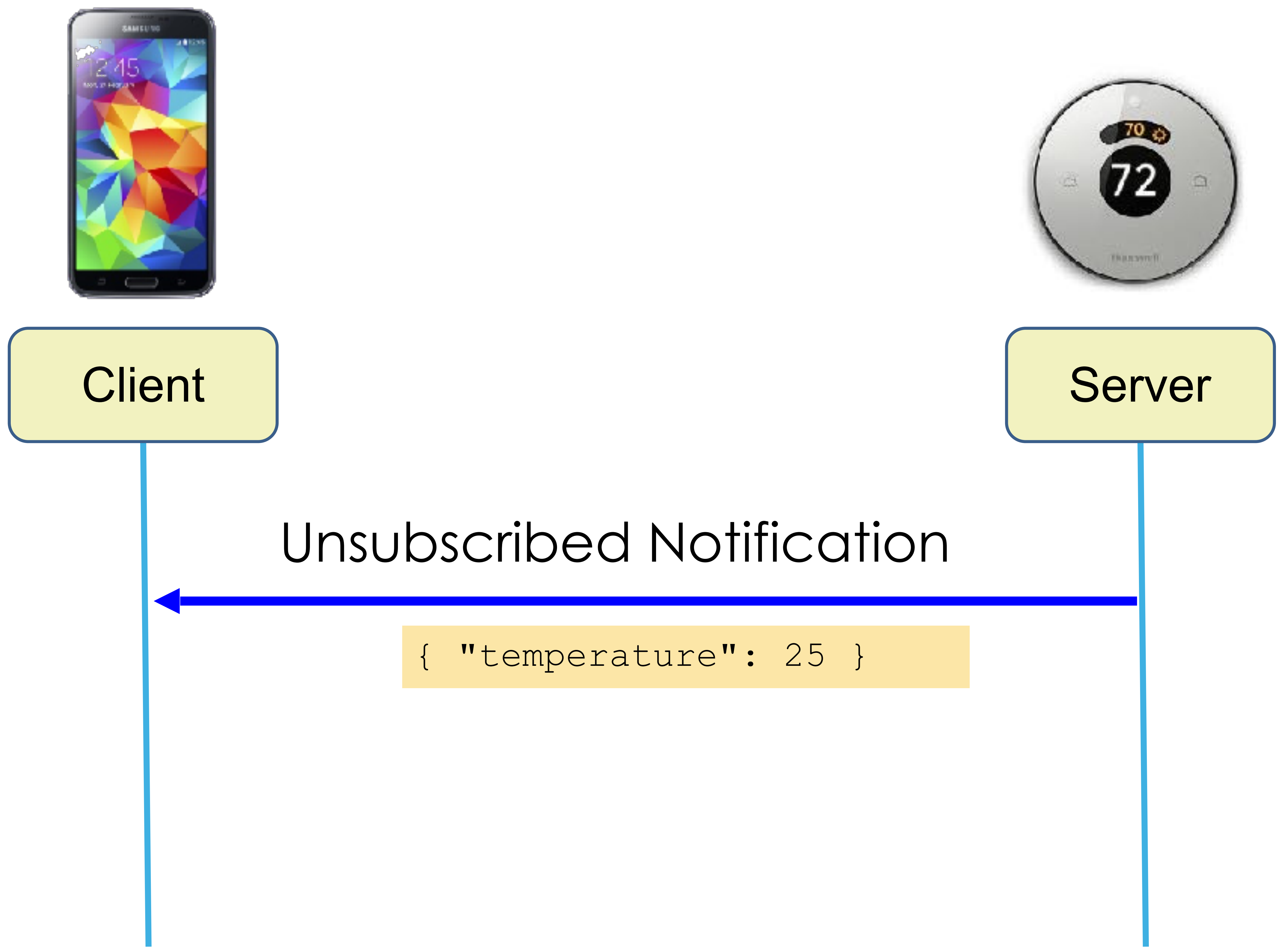


Notification overview

- 2 kinds of Notification*
  - Subscribed notification* (Notification in Mark's presentation)
    - A client makes a priori subscription for a target resource to the hosting server.
    - e.g.) CoAP Observe (RFC 7641)
  - Unsubscribed notification, aka, push model Notification* (Alert in Mark's)
    - No a priori subscription for the target resource.
    - Sometimes, it's hard to know where to send a subscription beforehand.
    - Ex) Fire alarm or Vehicular network emergency

Notification overview

- 2 kinds of Notification
  - Unsubscribed notification, aka, push model Notification

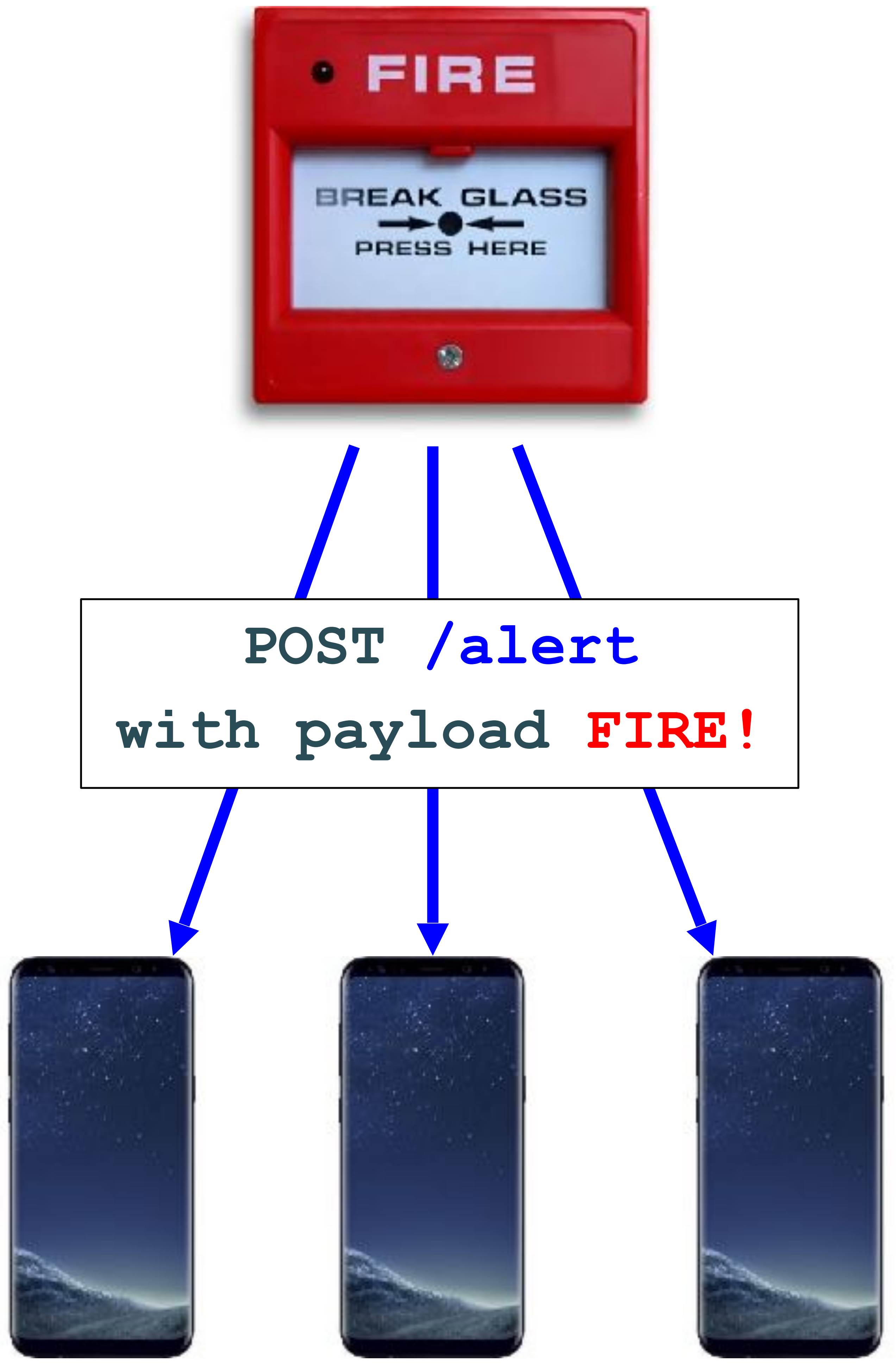


```
{ "temperature": 25 }
```

push model Notification

- Alert or Server push
  - How can a server send a notification without a priori subscription (e.g. CoAP Observe)

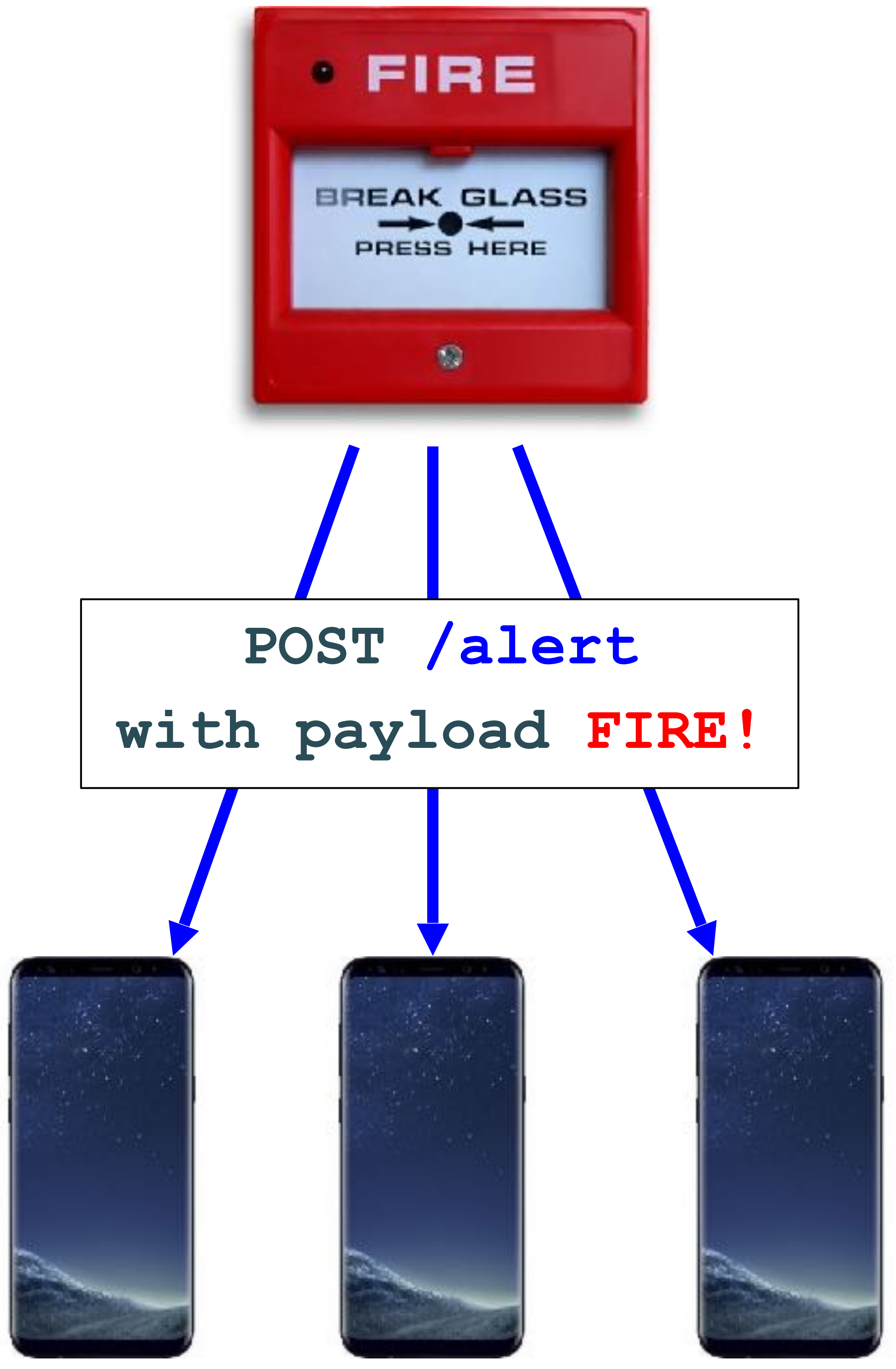
- Two approaches
  - POST based approach: Push with POST Request
    - The Server (as a Client) sends a Request with the alert information (e.g. POST request with a payload FIRE!)



push model Notification

- Alert or Server push
  - How can a server send a notification without a priori subscription (e.g. CoAP Observe)
- Two approaches
  - POST based approach: Push with POST Request
    - The Server (as a Client) sends a Request with the alert information (e.g. POST request with a payload FIRE!)
    - Covered by development by Achim von Neefe
    - A priori knowledge for target Resource for POST is required.
    - i.e.) (transfer protocol) URI for POST request

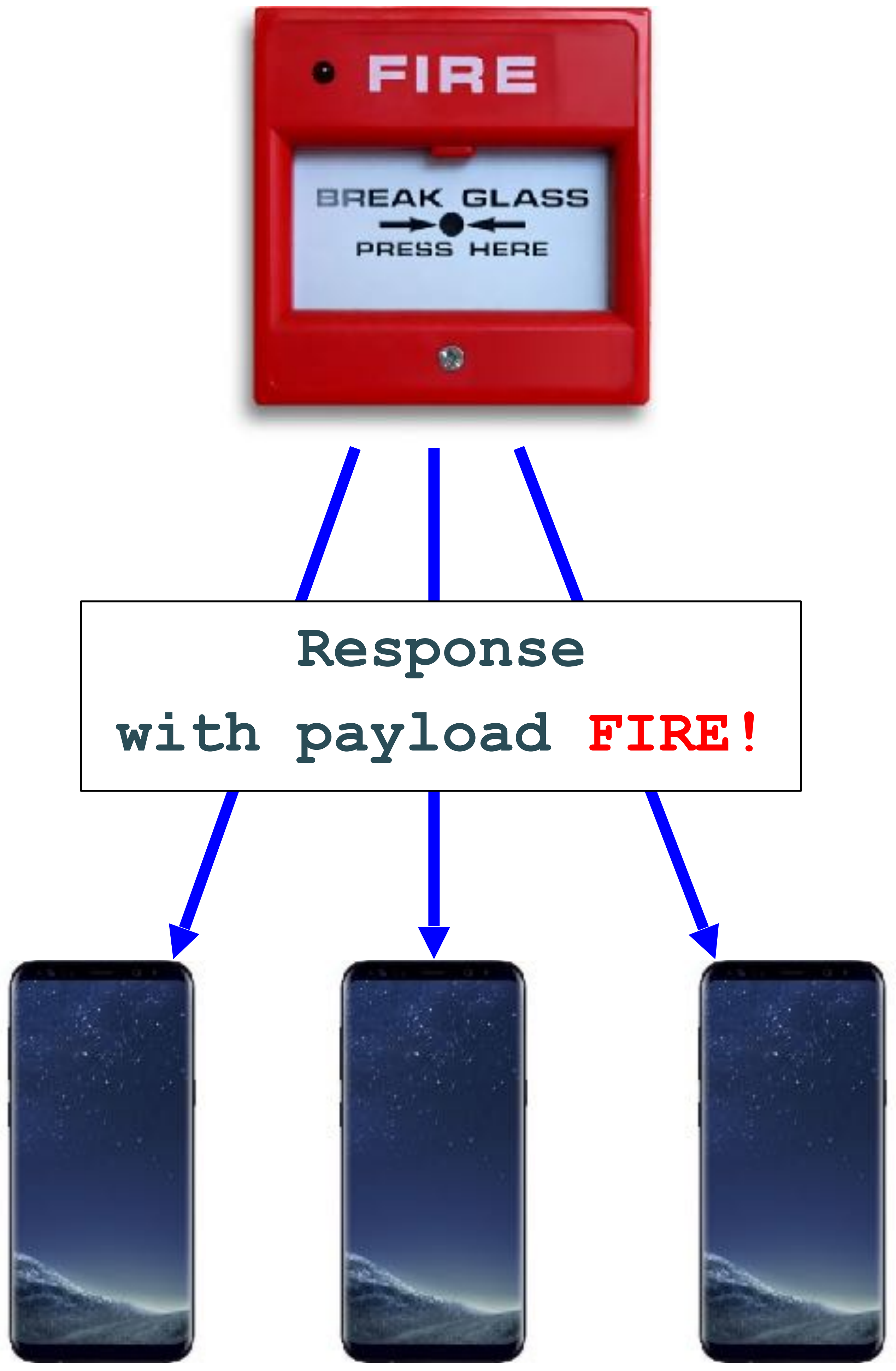
$$\texttt{coap://}\underbrace{\texttt{IPaddress: port\#}}_{\text{Authority Component}}\underbrace{\texttt{/well\_known\_path}}_{\text{Path component}}$$



push model Notification

- Alert or Server push
  - How can a server send a notification without a priori subscription (e.g. CoAP Observe)

- Two approaches
  - Unrequested response based approach: Non-traditional Response
    - The Server sends a unrequested response (e.g. Response with a payload FIRE!)



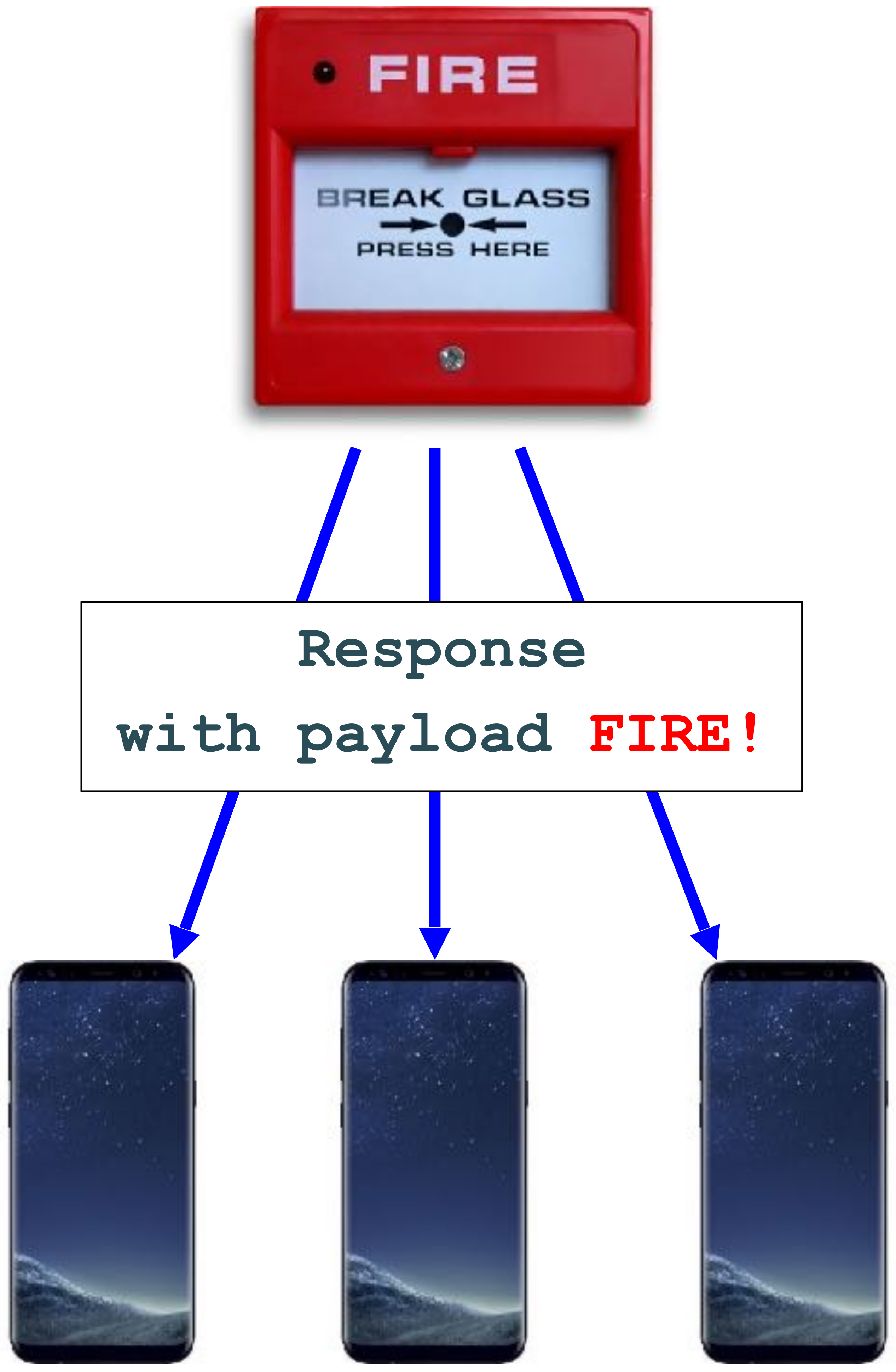
push model Notification

- Alert or Server push
  - How can a server send a notification without a priori subscription (e.g. CoAP Observe)

- Two approaches
  - Unrequested response based approach: Non-traditional Response
    - The Server sends a unrequested response (e.g. Response with a payload FIRE!)
    - Currently illegal? Can we allow such, maybe only for multicast?
    - Some issues need to be resolved (e.g., matching Token)
    - An I-D was submitted for the approach.

      CoAP: Non-traditional response forms

      draft-bormann-core-responses-00



Non traditional Response

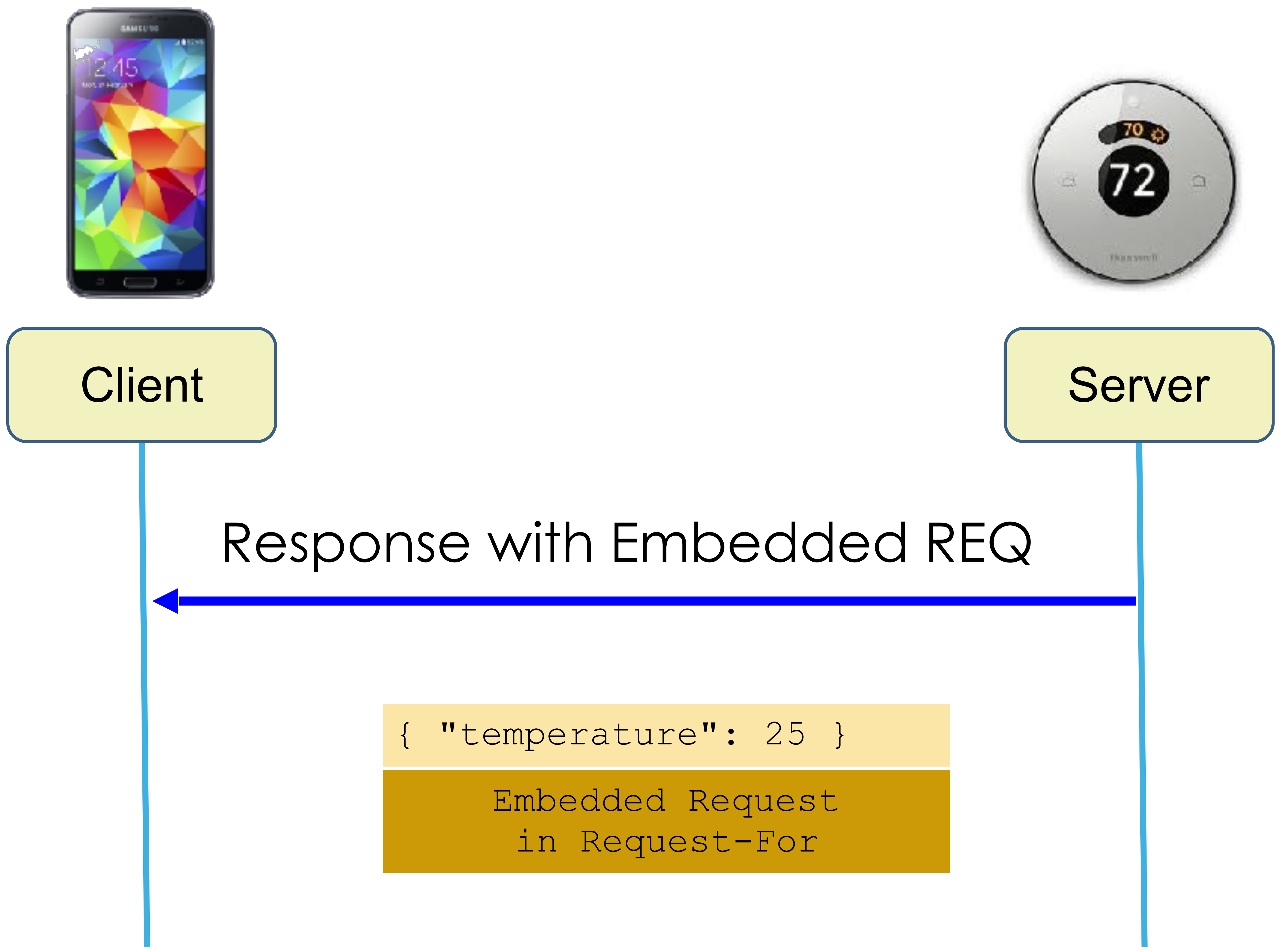
- Issues
  - Destination
    - Where to send such Response? IP address & port number?
  - Token
    - which token value to put in such response. How to satisfy the matching rule?
  - Security
    - How to prevent malignant usage?

Non traditional Response

- Approaches
  - Response with embedded request
    - A server sends a response to a request that it did not actually receive by embedding the request.
    - The option "Response-For" contains a request with zero-length token.
  - Response for configured request
    - A request may reach the server using a different means than that used for the response. For instance, the request may be configured in the server.
    - The client needs to know the configuration information beforehand (e.g., token)
    - E.g.) A Client may request a Server to send a Response to a different Client, i.e., A Client A askes a Server to send a Response to Client B with the option "Response-To".
  - Multicast Response
    - A server MAY send a response to a multicast address, where the token name space is shared by all nodes joined to that multicast address. Some management scheme needed for the token space.

Non traditional Response

- Approaches
  - Response with embedded request in “Request-For” Option



```
{ "temperature": 25 }
```

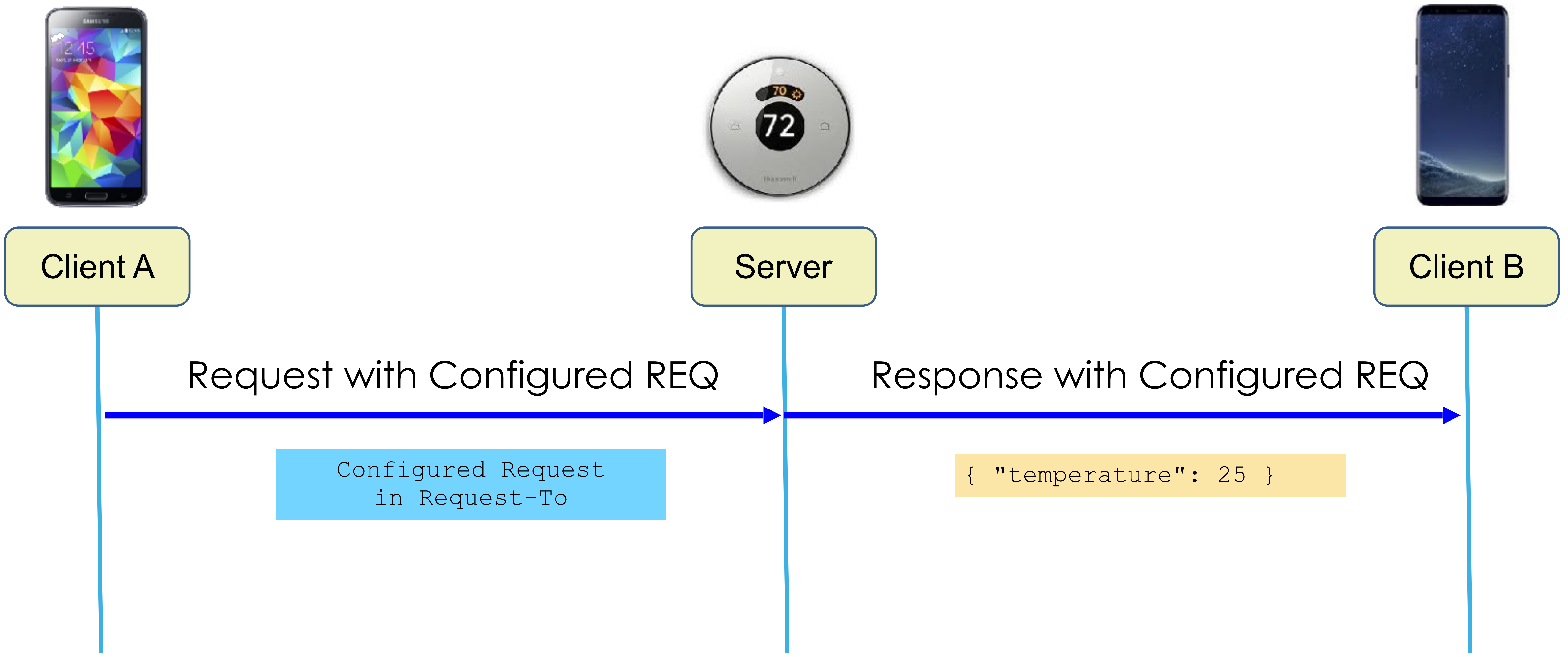
```
Embedded Request
in Request-For
```

Non traditional Response

- Approaches
  - Response with embedded request
    - A server sends a response to a request that it did not actually receive by embedding the request.
    - The option "Response-For" contains a request with zero-length token.
  - Response for configured request
    - A request may reach the server using a different means than that used for the response. For instance, the request may be configured in the server.
    - The client needs to know the configuration information beforehand (e.g., token)
    - E.g.) A Client may request a Server to send a Response to a different Client, i.e., A Client A askes a Server to send a Response to Client B with the option “Response-To”.
  - Multicast Response
    - A server MAY send a response to a multicast address, where the token name space is shared by all nodes joined to that multicast address. Some management scheme needed for the token space.

Non traditional Response

- Approaches
  - Response for configured request in “Request-To” Option

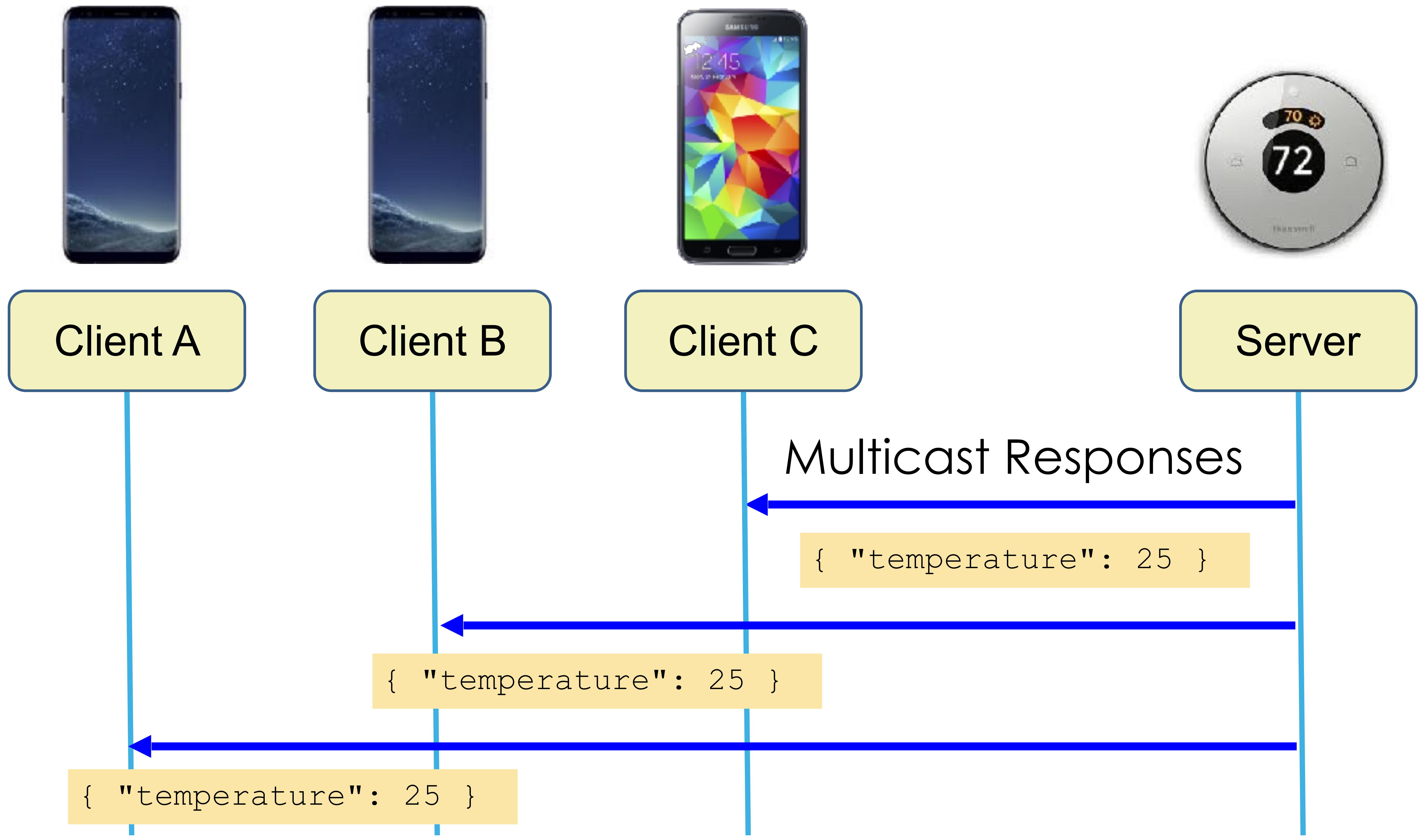


Non traditional Response

- Approaches
  - Response with embedded request
    - A server sends a response to a request that it did not actually receive by embedding the request.
    - The option "Response-For" contains a request with zero-length token.
  - Response for configured request
    - A request may reach the server using a different means than that used for the response. For instance, the request may be configured in the server.
    - The client needs to know the configuration information beforehand (e.g., token)
    - E.g.) A Client may request a Server to send a Response to a different Client, i.e., A Client A askes a Server to send a Response to Client B with the option “Response-To”.
  - Multicast Response
    - A server MAY send a response to a multicast address, where the token name space is shared by all nodes joined to that multicast address. Some management scheme needed for the token space.

Non traditional Response

- Approaches
  – Muticast Response



Follow through

- Summary
  - These descriptions are not intended as advocacy for adopting these approaches immediately, they are provided to point out potential avenues for development that would have to be carefully evaluated.
  - With developer feedback, we’ll investigate a suitable approaches in collaboration with IETF.

OPEN CONNECTIVITY FOUNDATION™

6 Links, Bindings, Interfaces, and Resource Representations

- CoRE Interfaces: Discuss alignment with OCF semantics of interfaces. What extensions are planned or considered?
- Dynlink: Use of dynamic links in new OCF design proposals. What is the roadmap for the Dynlink draft?
- Endpoint representation in a Link (OCF eps approach, draft-silverajan-core-coap-protocol-negotiation-07, draft-thaler-appsawg-multi-transport-uris-02)

7 “Base” CoAP

- Response code improvement (draft-keranen-core-too-many-reqs-00)
- Resource Directories: How to align OCF RD with IETF RD?

Too Many Requests Response Code for CoAP

- CoAP client can cause overload in server with too frequent requests
- How can server tell client to back off
- HTTP error code 429 “Too many requests”
- Proposal: register 4.29 for CoAP
  - With MaxAge to indicate when it’s OK to request again
- Originally part of CoAP Pub/sub Broker draft
- Strong support at IETF 101; working-group adoption call started

8 ACE, Security

T2TRG-OCF ACE WebEx Agenda (Jan 2018)

- Background
  - Overview of ACE; Charter, Documents
  - ACE use cases (RFC 7744) and Problem Statement (-actors)
  - ACE-Oauth "framework"
  - DTLS & OSCORE profiles
  - Group use ("pubsub") & group communications
- Contrasting this to the OCF security model
- Potential uses for ACE-style delegated authorization in OCF context
- Discussion
- Next steps

Conclusions & next steps

- “Lots of potential to use ACE stuff”
  - Devices in different LANs
  - Way for COP and SOP to establish credentials; CMS-CMS interaction was parked
  - Client side aspects; capability for requesting parties and client overseeing principles
- Report to OCF security group
- OCF would like to know when things hit RFC editor queue
- London hackathon results?
- Threats and mitigations for role certificate / access token manipulation by client manager