Constrained RESTful Environments WG (core)

Chairs:

Cullen Jennings <fluffy@cisco.com>

Carsten Bormann <cabo@tzi.org>

Mailing List:

core@ietf.org

Jabber:

core@jabber.ietf.org

- **We assume people have read the drafts**
- **Meetings serve to advance difficult issues by making good use of face-to-face communications**
- **Be aware of the IPR principles, according to RFC 3979 and its updates**

✓Blue sheets
✓Scribe(s)

Milestones (from WG charter page)

http://datatracker.ietf.org/wg/**core**/charter/

Document submissions to IESG:

- **Apr 2010 Select WG doc for basis of CoAP protocol**
- **Dec 2010 CoAP spec with mapping to HTTP REST submitted to IESG as PS**
- **Dec 2010 Constrained security bootstrapping spec submitted to IESG as PS**

- **Jan 2011 Recharter to add things reduced out of initial scope**

77th IETF: core WG Agenda

09:00	Introduction, Agenda	Chairs (5)
09:05	What is REST?	LD (15)
09:20	1 – CoAP reqts and protocol	ZS (55)
10:15	2 – Bootstrap & Security	RS/CO (45)
11:00	0 – Compressed IPfix	LB (15)
11:15	Other Topics	Chairs (20)

77th IETF: core WG Agenda

09:00	Introduction, Agenda	Chairs (5)
09:05	What is REST?	LD (15)
09:20	1 – CoAP reqts and protocol	ZS (55)
10:15	2 – Bootstrap & Security	RS/CO (45)
11:00	0 – Compressed IPfix	LB (15)
11:15	Other Topics	Chairs (20)

REST

Key concepts, terminology,
shared architectural understanding

Architectural Styles and Features

Messaging	Sessions	Transactions
Request/ Response	Store and Forward	Client/Server
Cached	Queued	Pub/Sub
Remote Method Invocation	Remote Procedure Call	Resource-oriented
Connection-oriented	Proxied	Synchronous
Service-oriented	Pipe and Filter	CRUD

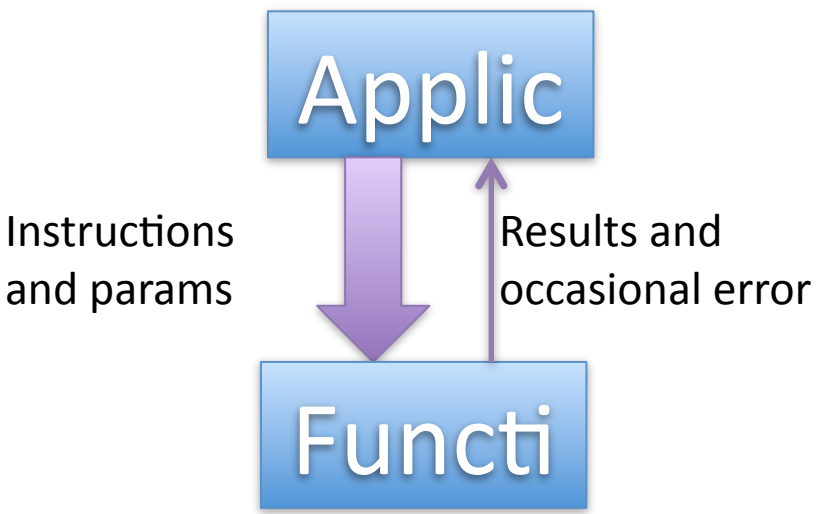
Which ones are REST

Messaging	Sessions	Transactions
Request/ Response	Store and Forward	Client/Server
Cached	Queued	Pub/Sub
Remote Method Invocation	Remote Procedure Call	Resource-oriented
Connection-oriented	Proxied	Synchronous
Service-oriented	Pipe and Filter	CRUD

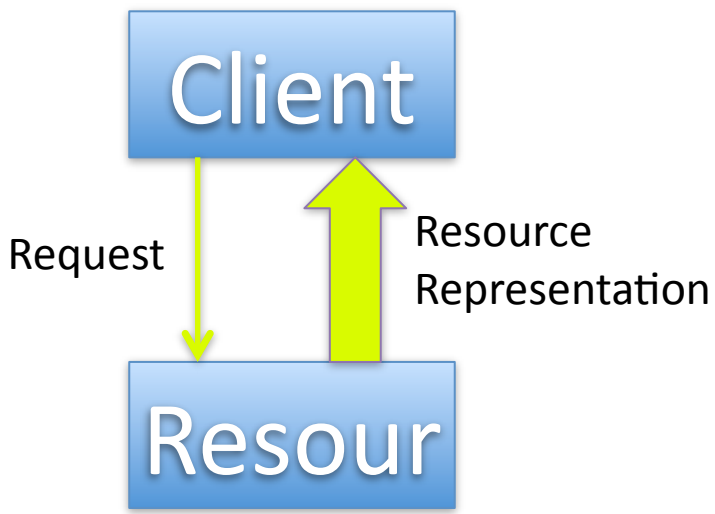
Client, Server, Request, Response

Power relationship is opposite of imperative or functional programming

Imperative programming Application



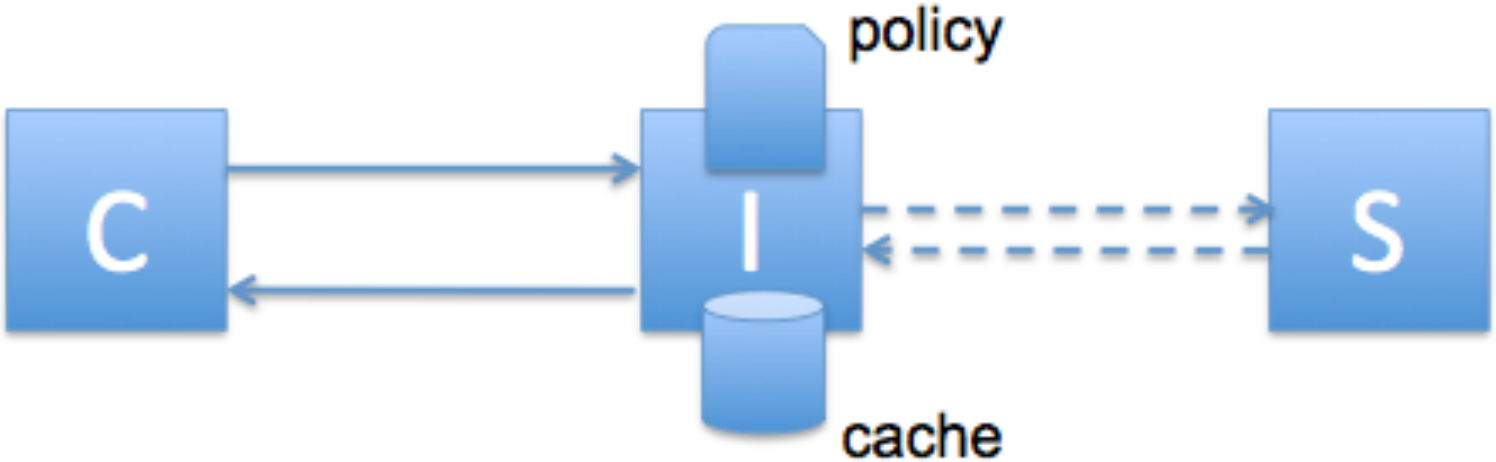
RESTful Network Application



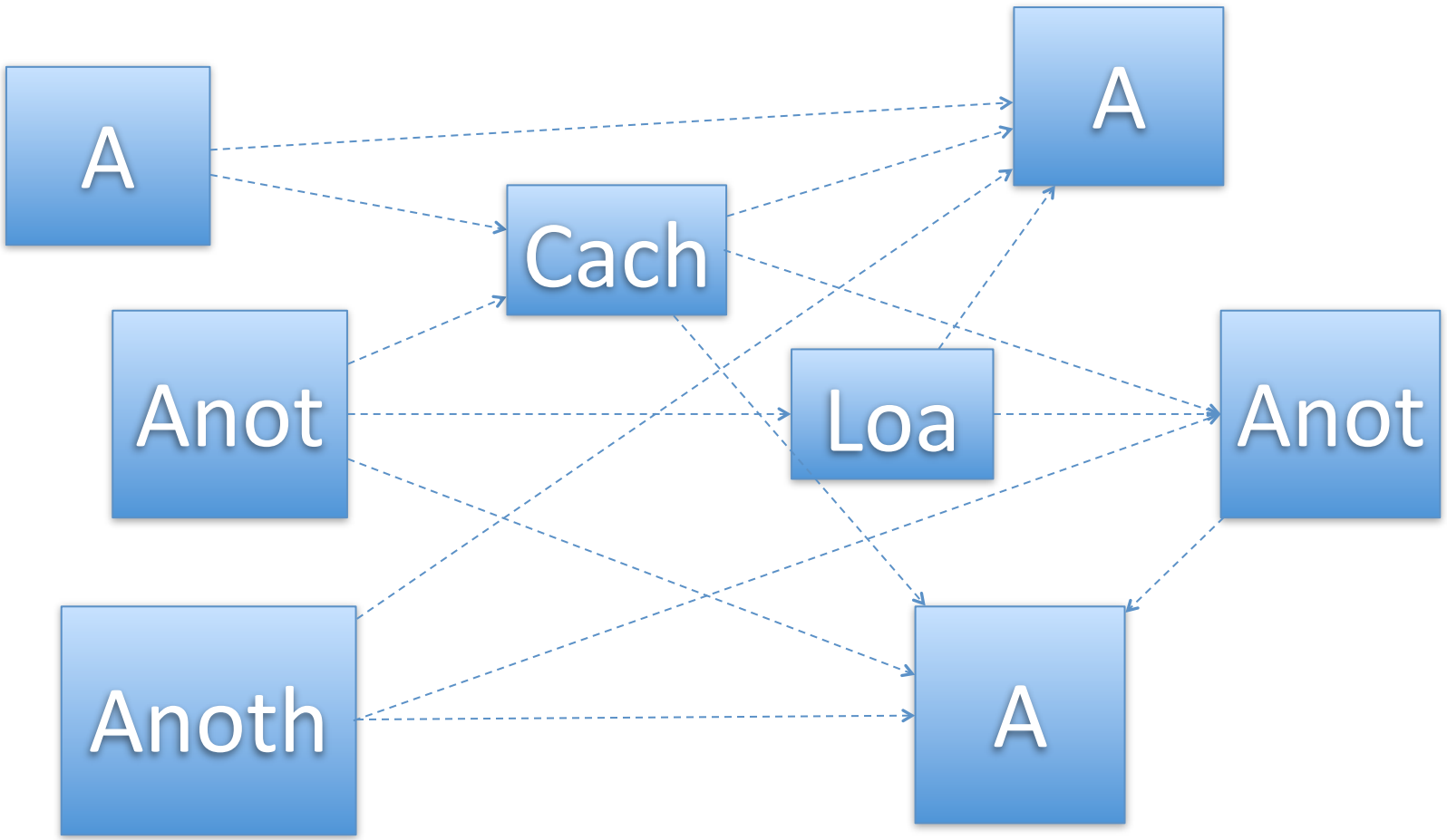
REST vs RPC Styles

REST	RPC
GET /.well-known/user-services … returns URLs to **resources**	*No need to bootstrap*
GET /users	getUsers()
GET /newusers	getUsersSince(date *d*)
GET /users/l	getUsersMatching(string *regexp*)
POST /newuser <body> … returns URL of **new resource**	createUser(*params*)
PUT /data/users2/0037 <body>	updateUser(*params*)

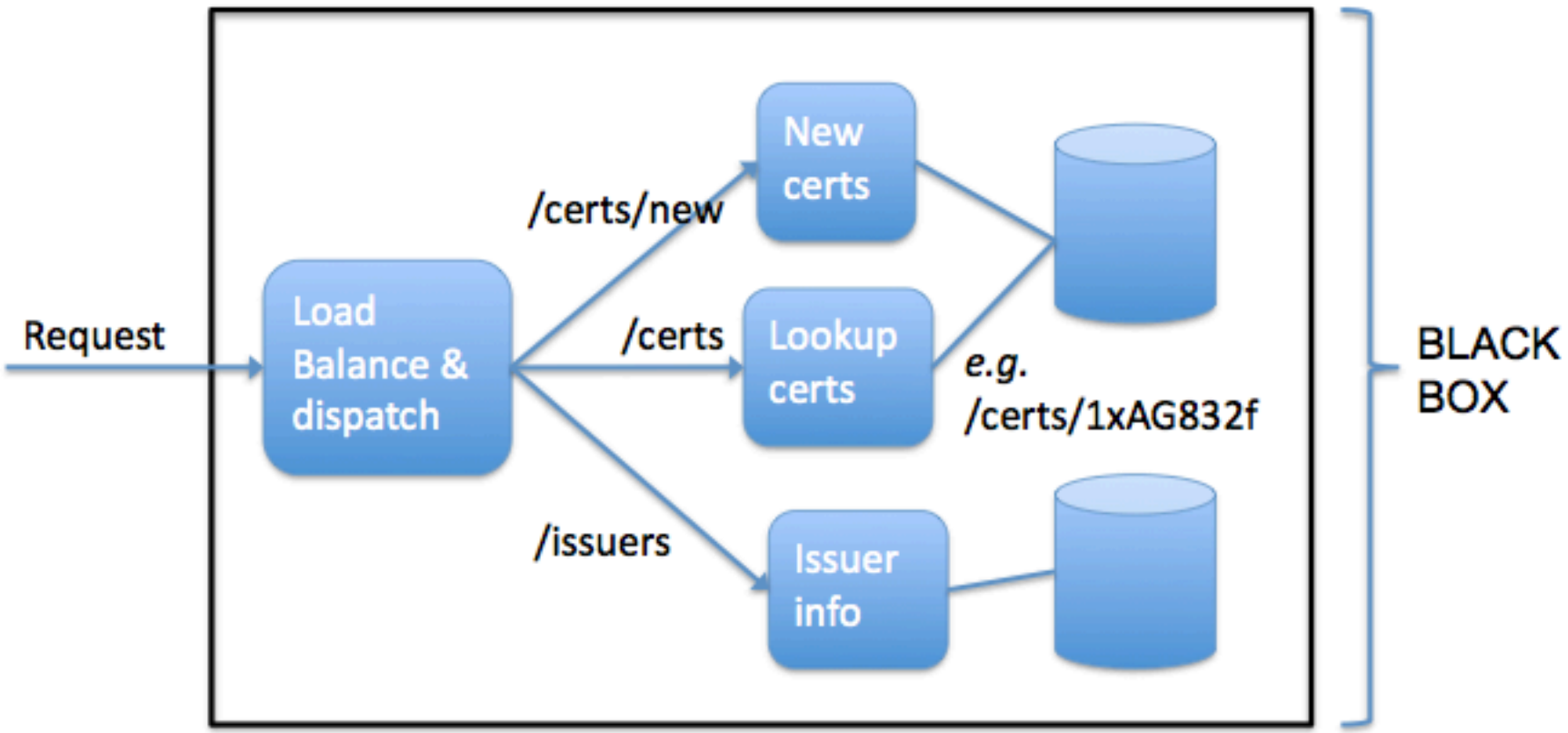
Proxied and Cached



Stateless

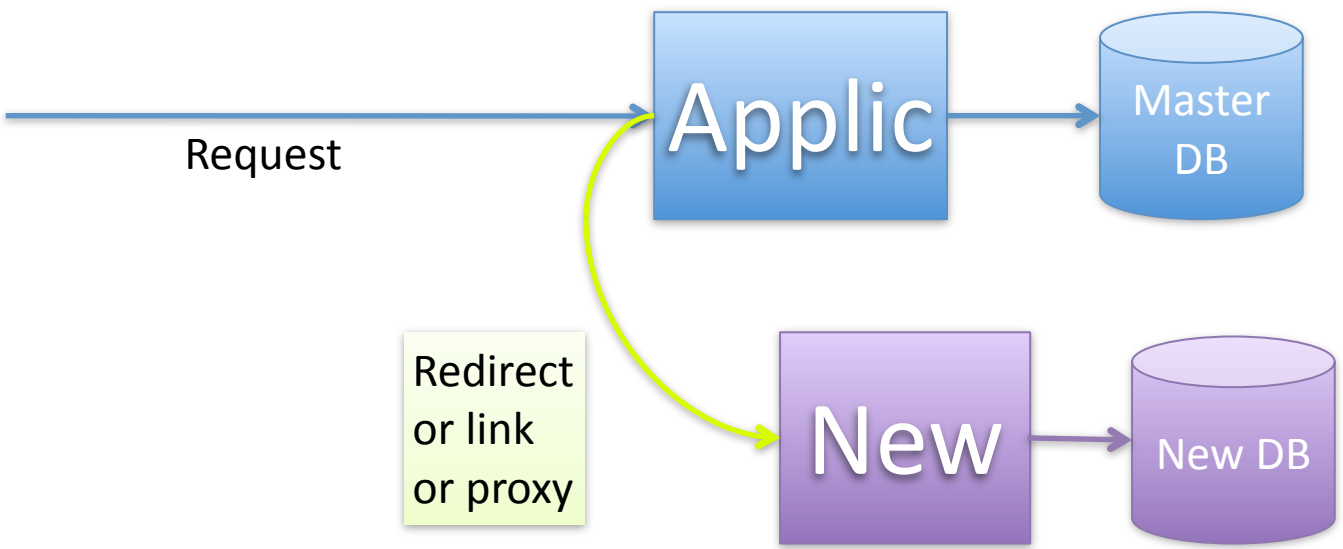


Scalable and Flexible

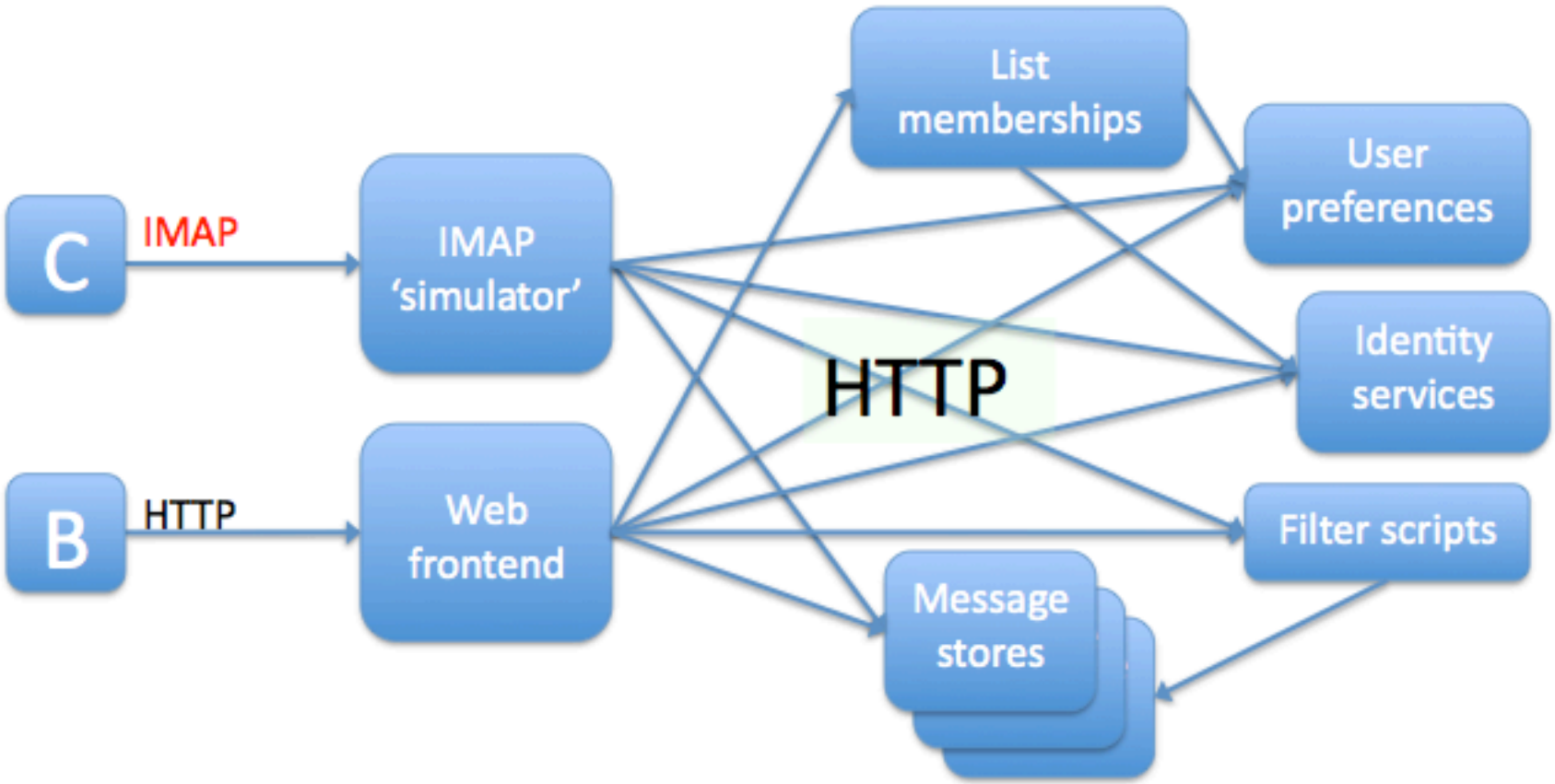


Iterative Development

IF servers really control namespace and data segmentation, server developers can iterate



Loose coupling of independently



Server-controlled Resource Discovery

Yes	Not so much
HTML pages with links chosen by server to offer	URLs with query parameters that the client manipulates
Atom feds with entries and pages chosen by server	WebDAV PROPFIND requests where client forms query
Link relations in documents or document metadata	URLs constructed by clients from fixed template
Bootstrapping from starting point (/ or .well-known/*)	URLs fixed and published e.g. in an interop specification

Good use of Content type

- Application type is often indicated by content type
  - New content type can indicate new version too
- Multiple content types can be supported
  - Dispatching via “Accept” header
  - Dispatching via server URL offered in discovery resources

Reuse

- Reuse of SOFTWARE
- Reuse of Interoperability Features
  - Caching and cache invalidation by resource
  - Conditional requests, both for GET and PUT
  - Compression and delta downloads
  - HTTP PATCH for partial upload
  - WebDAV ACL on resources
  - WebDAV collections or Atom feeds
  - Versioning of resources

Comments Welcome

END OF PRESENTATION

For the Reader

This presentation is pretty heavy on the pictures and talking, not text. For reading this presentation after the fact, here are some links and quotes to provide much more context.

Roy Fielding's PhD Dissertation

"At no time whatsoever do the server or client software need to know or understand the meaning of a URI — they merely act as a conduit through which the creator of a resource (a human naming authority) can associate representations with the semantics identified by the URI. In other words, there are no resources on the server; just mechanisms that supply answers across an abstract interface defined by resources.It may seem odd, but this is the essence of what makes the Web work across so many different implementations. It is the nature of every engineer to define things in terms of the characteristics of the components that will be used to compose the finished product. The Web doesn't work that way. The Web architecture consists of constraints on the communication model between components, based on the role of each component during an application action. This prevents the components from assuming anything beyond the resource abstraction, thus hiding the actual mechanisms on either side of the abstract interface."

Wikipedia entry on “SOAP”

“Most uses of HTTP as a transport protocol are done in ignorance of how the operation would be modeled in HTTP. This is by design—similar to how different protocols sit on top of each other in the IP stack. But this analogy is imperfect; the application protocols used as transport protocols aren't really transport protocols. As a result, there is no way to know if the method used is appropriate to the operation. This makes good analysis at the application---protocol level problematicc with sub--- optimal results—for example, a POST operaiton is used when it would more naturally be modeled as a GET. The REST architecture has become a web service alternaitve that makes appropriate use of HTTP's defined methods.”

Classification of HTTP APIs

1. *"WS-** and *RPC URI-Tunneling* are the worst approaches from a benefits/effort ratio point of view. The lack of best practices makes *RPC URI-Tunneling* considerably worse. Basically, a designer/developer can just go off and go wild.
2. The start-up cost of *HTTP-based Type I* APIs is actually smaller than the one of *RPC URI-Tunneling*, mostly because the former leverages HTTP mechanisms (methods, failure model, caching). […]
3. Depending on the degree to which existing media types apply to the problem domain *HTTP-based Type II* should be considered over *HTTP-based Type I* because the start-up cost is almost identical. A transition from *HTTP-based Type II* to REST at a later point in time, however, is rather easy[...]
4. Turning a *HTTP-based Type II* API into a REST API might be as easy as deleting the API documentation.
5. If you are to any degree concerned with long term maintenance- and evolution cost[…] REST is the best solution despite the start-up cost. The start-up cost will amortize in the long run."

77th IETF: core WG Agenda

09:00	Introduction, Agenda	Chairs (5)
09:05	What is REST?	LD (15)
09:20	1 – CoAP reqts and protocol	ZS (55)
10:15	2 – Bootstrap & Security	RS/CO (45)
11:00	0 – Compressed IPfix	LB (15)
11:15	Other Topics	Chairs (20)

draft-shelby-core-coap-req-00

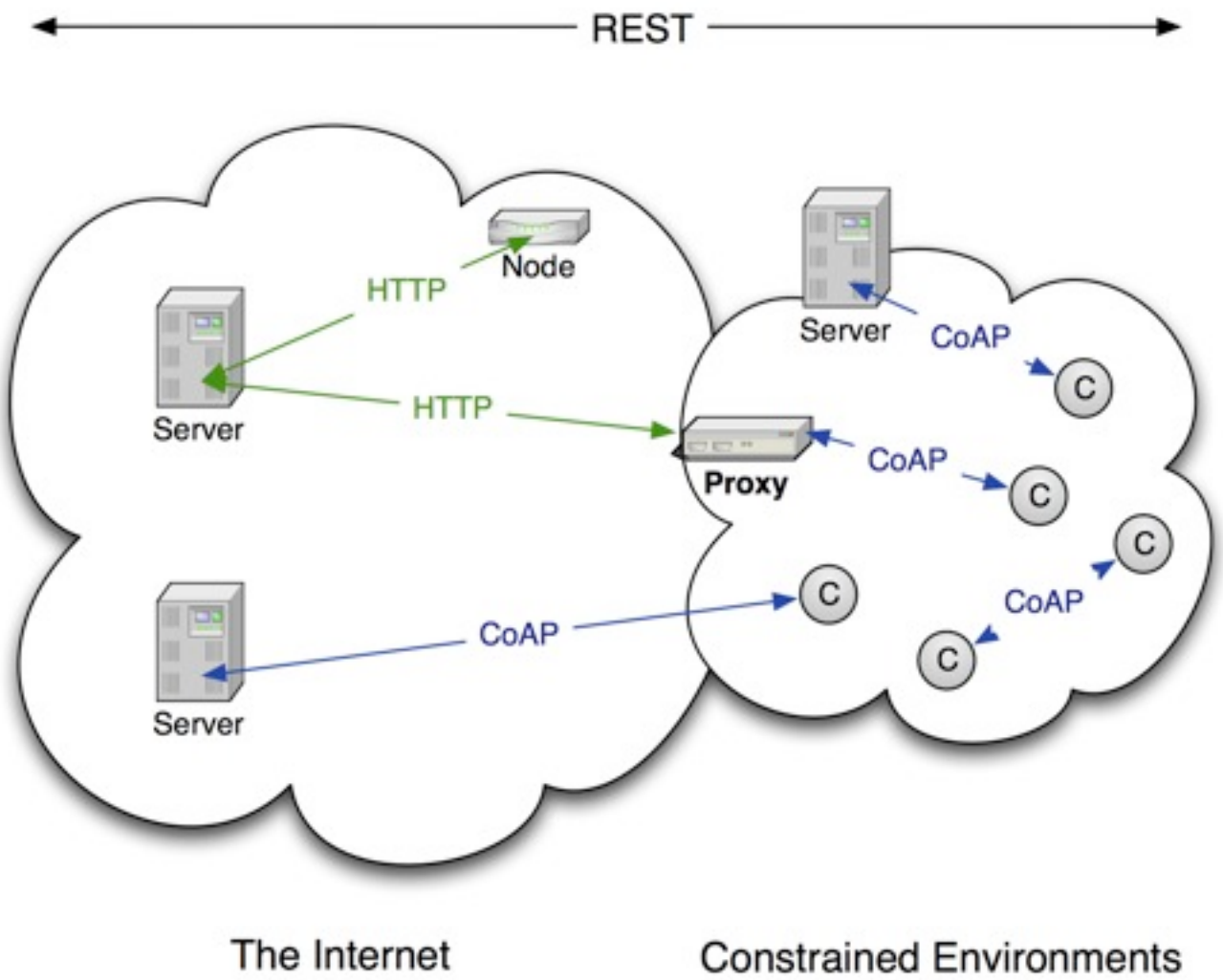
Z. Shelby, M. Garrison Stuber, D. Sturek, B. Frank, R. Kelsey

draft-shelby-core-coap-00

Z. Shelby, B. Frank

CoRE WG, IETF-77 Anaheim

CoRE Architecture



draft-shelby-core-coap-req-00

- Point of this document
- Applications considered
- Requirements in a nutshell
- Summary and next steps

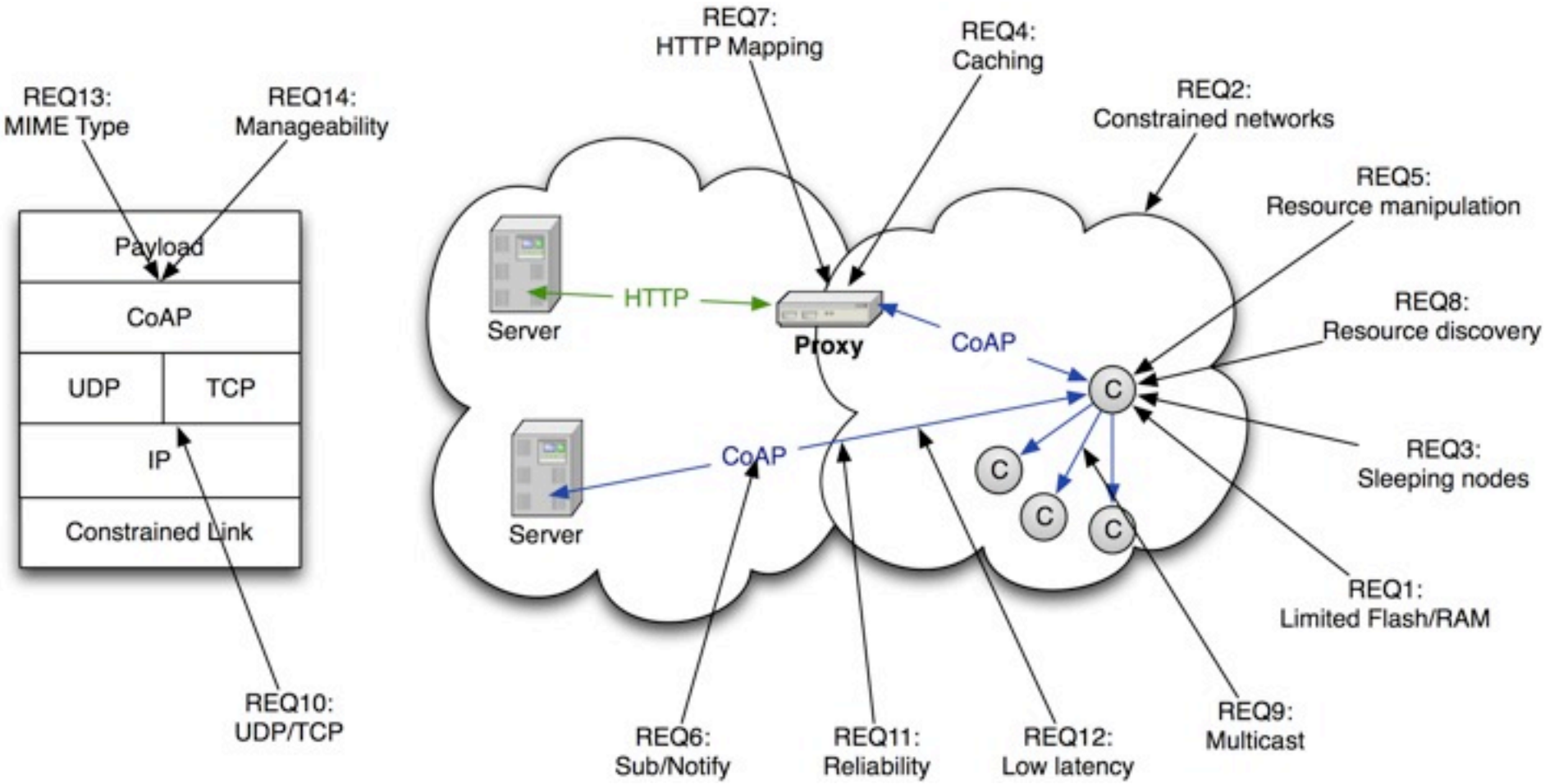
Point of this document

- Summarize related requirements from
  - The approved charter
  - draft-bormann-6lowpan-6lowapp-problem-01
  - draft-sturek-6lowapp-smartenergy-00
  - draft-martocci-6lowapp-building-applications-00
  - draft-gold-6lowapp-sensei-00
- Discuss possible CoAP features
- Develop an applicability analysis
  - Smart Energy, Building Automation, M2M

Applications considered

- Smart Energy
  - ZigBee Smart Energy 2.0
  - CENELEC/ETSI M2M
  - Smart grid applications
- Building Automation
  - Commercial and home automation
  - Open Building Information Exchange (oBIX)
- General M2M Applications

Requirements in a nutshell



Summary and next steps

- This draft summarizes CoAP requirements
- Needed features identified
  - Along with possible solutions
- Applicability analysis just started
- Next step?
  - Reduce to just the requirements
  - Move features and applicability to CoAP ID
  - Submit as -01
  - Leave as documentation...

draft-shelby-core-coap-00

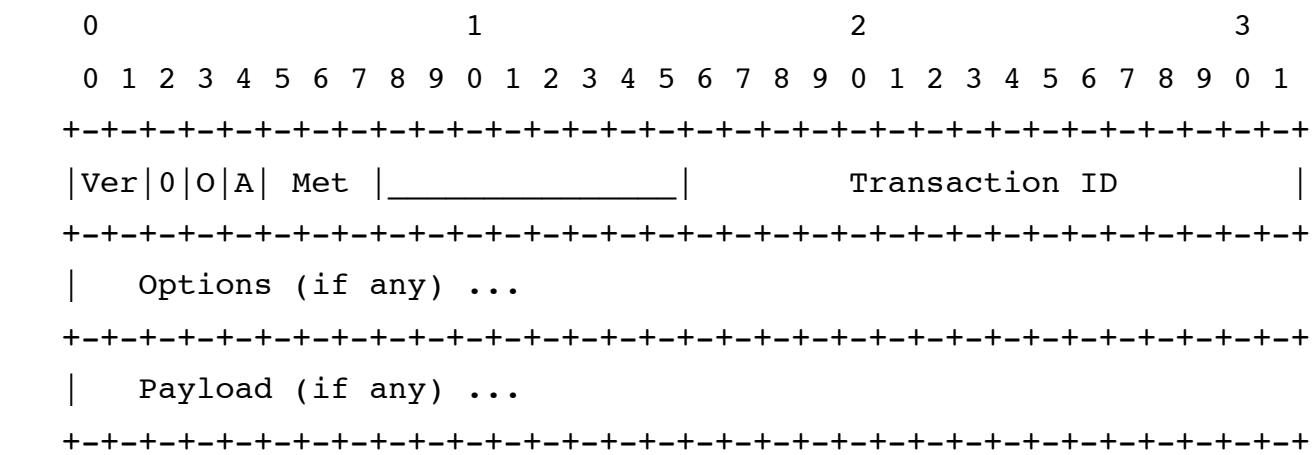
- Protocol overview
- Message header
- Transport binding
- Discovery
- Subscription & notification
- Caching
- HTTP mapping
- Next steps

Protocol overview

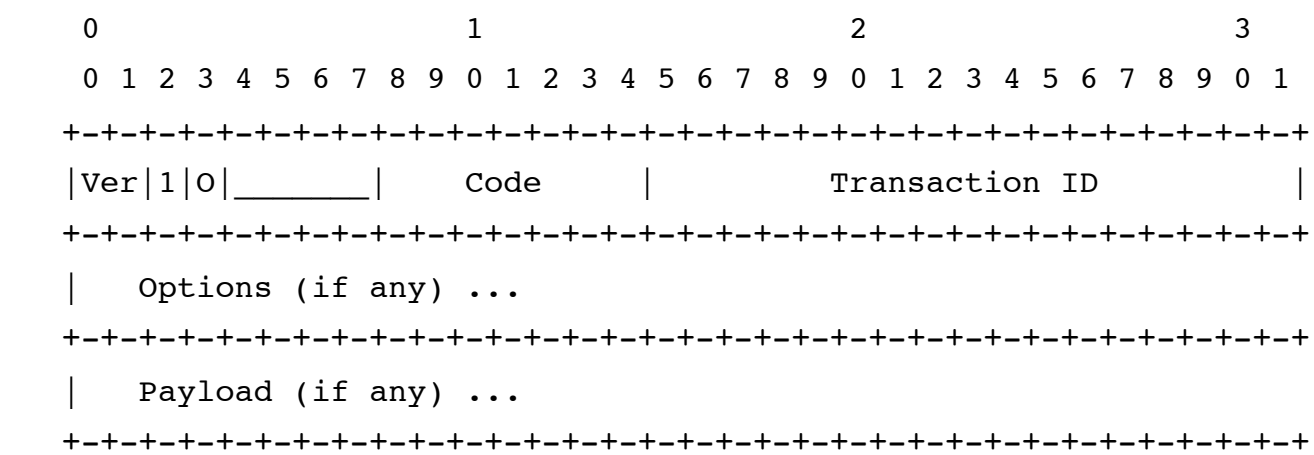
- Simple RESTful protocol transport
  - Create, Read, Update, Delete, Notify
- Small, simple header < 10 bytes
  - 4 byte base header
  - TLV options, typically 3-4 bytes per option
- URI support (string or integer)
- Subset of content types
- Subset of HTTP-compatible response codes
- UDP and TCP bindings
- coap://
- default port = 61616

Message header

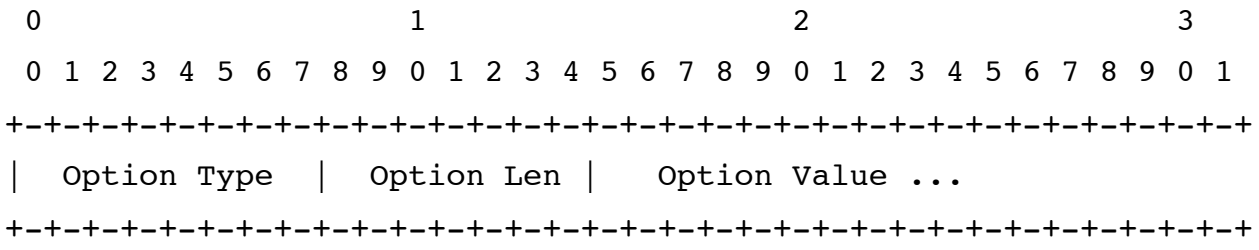
REQUEST HEADER



RESPONSE HEADER



Message header: Option



Type	Name	Data type	Description
0x0	No-more-options	None	Indicates no more options
0x1	Content-type	16-bit unsigned integer (Len = 2)	The content-type of the message-body
0x2	Uri-string	String	The URI of the resource
0x3	Uri-code	8-bit unsigned integer (Len = 1)	The URI of the resource
0x4	Max-age	16-bit unsigned integer (Len = 2)	The maximum age of the resource for use in caching

Transport binding

- UDP Binding
  - Default and required
  - Stop-and-wait reliability
    - Req with A flag - match Res with TID
    - Retransmission (up to x times on timeout)
  - Unicast and multicast support
- TCP Binding
  - Delivering large chunks of data
  - Continuous streams of data
  - When congestion control is needed (across the Internet e.g.)

Discovery

- Service Discovery
  - Finding services in the first place
  - We are not chartered to define a new one
  - We can point to a suggested way of doing this step if needed (e.g. mDNS)
- Resource Discovery (a.k.a Web Discovery)
  - Finding the web resource of a CoAP service
  - Getting and notifying lists of resource links
  - Required feature of CoAP

Discovery - requirements

- Support both user agent and directory agent discovery methods
  - Directory agent support is not popular with HAN device manufacturers (due to cost)
  - User agent method should be supported for all devices that can support it (non sleeping devices)
  - Limited directory agent support is needed for devices which sleep (gas meters, water meters, etc.)

Discovery - requirements

- Multicast support for discovery requests – There is no list stored of devices on the HAN
- Resource discovery with few parsing requirements (and's, or's and other complicated request parsing should not be required)
- Efficient delivery of resource lists in discovery responses (resource IDs, tokens, etc.)

Discovery - requirements

- Limit packet sizes of requests and responses
  - While we don't want to align application support with Layer 2 packet sizes, the truth is, the L2 PDU is only 127 bytes!
  - However, we want to align with common web services (human readable URIs, XML, etc.)
  - Tokens, resource IDs, etc. to reduce request and response payloads are very welcome contributions

Discovery - requirements

- Standard security protocols and methods for delivery of requests and responses (TLS, DTLS, etc.)
- Access Control Lists (ACLs) or other security controls for URI access (only expose resources in discovery responses matching the security level of the requestor)

Discovery - possible solutions

- The apps area works on several solutions
  - Eran Hammer-Lahav gave a great overview in apps area meeting (link??)
- Called Web Discovery in HTTP world
- Well-known URIs /.well-known/
  - draft-nottingham-site-meta
- HTTP link header format (RFC2068)
  - draft-nottingham-http-link-header
- Atom (RFC4287) or HTML link lists
- XRD - XML Resource Description
- LRDD - Link-based Resource Description
  - draft-hammer-discovery

Discovery - a proposal

- RESTful discovery using CoAP
- Discovery pull (unicast or multicast)
  - READ / (default discovery port) **or**
  - READ /.well-known/coap-links (default port)
- Discovery push (unicast or multicast)
  - NOTIFY / (default port)
- Discovery agent registration (unicast)
  - UPDATE / (default port) to agent
- Returns resources listed using subset of:
  - draft-nottingham-http-link-header

  </temperature>; rel=TemperatureC; media=text/xml

  etc....

Subscription & notification

- CoAP also requires a push model
- One possible REST solution:
- Subscribe to a resource URL
  - READ the URL with “subscribe header”
  - Header has call-back URL and timeout
  - Subscription must be refreshed
- Notifications sent upon change
  - NOTIFY sent to call-back URL

Caching

- CoAP will support simple caching
  - Useful for sleeping nodes
  - Decreases load on constrained networks
- Three possible scenarios:
  - An intermediate CoAP proxy may cache resources and answer READ requests using a cached version.
  - An intermediate CoAP proxy may cache subscriptions to a sleeping node.
  - An intermediate CoAP proxy may use notifications from a node to update a resource.

HTTP mapping

- An HTTP mapping is needed for CoAP
- Basic RESTful methods are easy
- Subscription/Notification is more difficult
  - What is the HTTP interface to subscribe to a CoRE resource?
  - How to push resulting notifications to the HTTP client?
    - HTTP polling
    - HTTP long poll (draft-loreto-http-bidirectional-01)

Known Issues

- Length of message body (option?)
  - Pack multiple messages in TCP
- Content transfer encodings not needed
- Do we need a magic byte?

Next steps

- First try at resource discovery section
  - Don Sturek will help write this
- First subscription/notification section
- Write the HTTP mapping section
- Complete caching section
- Fix nits and comments from the list
- -01 submission in April

77th IETF: core WG Agenda

09:00	Introduction, Agenda	Chairs (5)
09:05	What is REST?	LD (15)
09:20	1 – CoAP reqts and protocol	ZS (55)
10:15	2 – Bootstrap & Security	RS/CO (45)
11:00	0 – Compressed IPfix	LB (15)
11:15	Other Topics	Chairs (20)

Bootstrapping

CoRE WG Presentation by Colin O'Flynn, Atmel

Overview

- Definition of Bootstrapping
- Problems Faced
- Existing Solutions
- Proposed Framework
- Fitting In with CoAP

Bootstrapping – What is it?

- The magic that takes a network from a box of nodes to a fully functioning network

- draft-oflynn-core-bootstrapping specifies that:
  - Bootstrapping is complete when settings have been securely transferred prior to normal operation in the network.

Bootstrapping – What is it not?

- Does not replace service or resource discovery
  - Bootstrapping is finished when normal network operation can begin, at which point service or resource discovery can occur

Bootstrapping – Problems

- Merging Networks
  - If a node is already on a network, and the user wishes this node to join another network, what happens?
- Node Mobility
- Resource Constraints
  - Computational, Power, Size, and Price
- User Interface
  - Wide range of nodes: from full graphical LCD to no user interface
- Security

Existing Solutions

- Examples of solutions to these problems exist in several standards, such as :
  - WiFi Protected Setup (WPS)
  - Bluetooth
  - Wireless USB
- Typically defined for too narrow an application-space for CoRE though. As CoRE nodes span the range from:
  - Tiny parasitic power devices to wall-powered nodes
  - 8-bit microcontrollers to 32-bit processors
  - Low to High security requirements (ie: light switch vs. smart meter)

Proposed Framework

- **Communications Channel**: Used during normal network operation (e.g.: 802.15.4)
- **Control Channel**: Used for bootstrapping only (e.g.: IrDA, NFC, 802.15.4)
- **User Interface**: Defines what the user controls the node with (e.g.: pushbutton, keyboard)
- **Bootstrap Profile**: Defines information exchanged during bootstrapping (e.g.: channel settings, encryption keys)
- **Bootstrap Protocol**: Actual messages exchanged for bootstrapping (note: this 'protocol' likely a wrapper on existing protocols)

Fitting in with CoAP

- Bootstrapping requires input from other layers to work!
  - User needs to select networks/nodes to join
  - Node may automatically join networks based on available services
    - Bootstrapping should NOT duplicate service discovery, but work with the proper layers / standards
- Bootstrapping difficult to implement "cleanly"

Next Steps

- Feedback from requirements of different users
- Decide on standards which bootstrapping will use
- Fit bootstrapping and CoAP together
- Finish documentation

77th IETF: core WG Agenda

09:00	Introduction, Agenda	Chairs (5)
09:05	What is REST?	LD (15)
09:20	1 – CoAP reqts and protocol	ZS (55)
10:15	2 – Bootstrap & Security	RS/CO (45)
11:00	0 – Compressed IPfix	LB (15)
11:15	Other Topics	Chairs (20)

Compressed IPFIX for Smart Meters in Constrained Environments

draft-braun-core-compressed-ipfix-01

Lothar Braun, Corinna Schmitt, Benoit Claise, Georg Carle

77th IETF Meeting, Anaheim, 2010

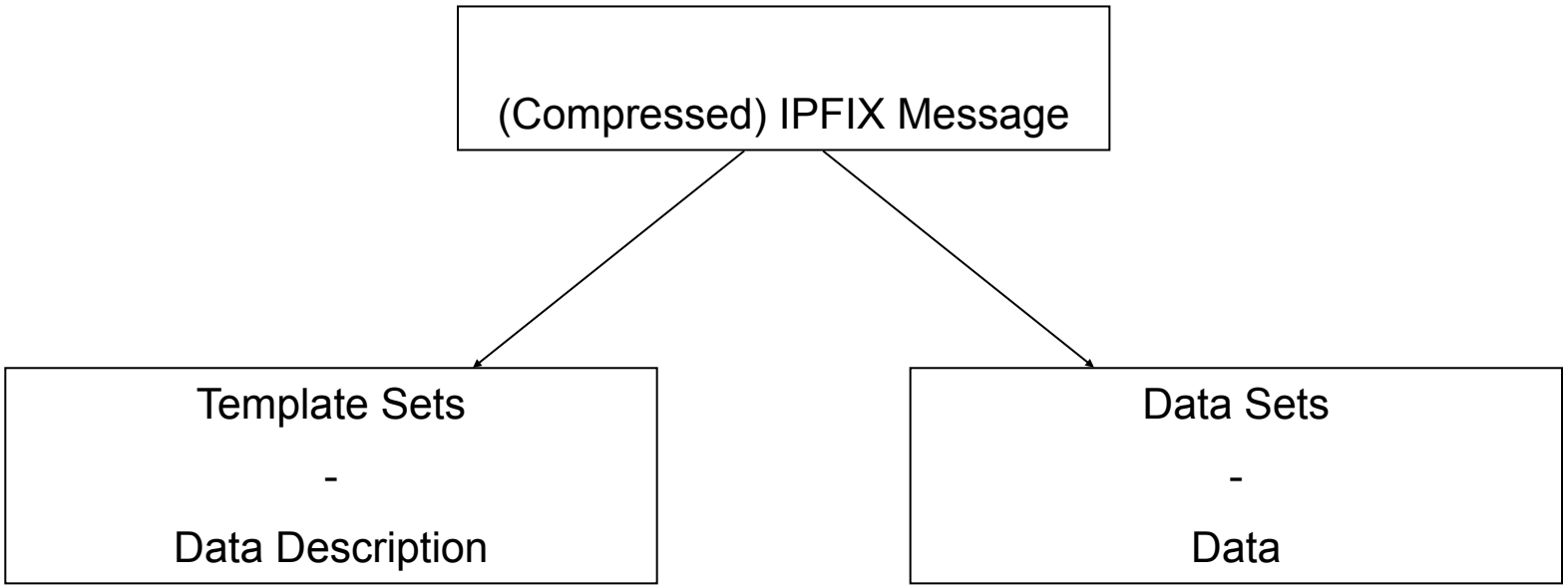
Agenda

- Application Scenarios
- (Compressed) IPFIX
  - Protocol structure
  - Information Model
- Differences between Compressed IPFIX and IPFIX
- Implementation Status
- Future Plans
- Discussion

Application Scenarios

- Wireless sensor networks for **long term measurement** of physical quantities (temperature, humidity, power consumption, …)
  - Target application scenarios accept packet loss
  - Sensors perform periodic measurements
  - Optional: several measurements can be aggregated into a single packet
  - Export of aggregated measurements
- Use existing protocol: IPFIX (IP Flow Information eXport)
  - Simple information model
  - Separation between data and data description
- Problem: Header sizes ➔ Modifications are necessary
  - IPv6 → 6LoWPAN
  - IPFIX → Compressed IPFIX

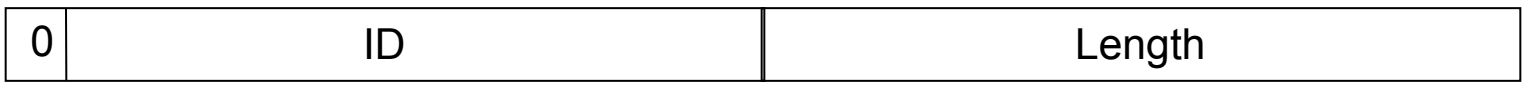
(Compressed) IPFIX – Messages



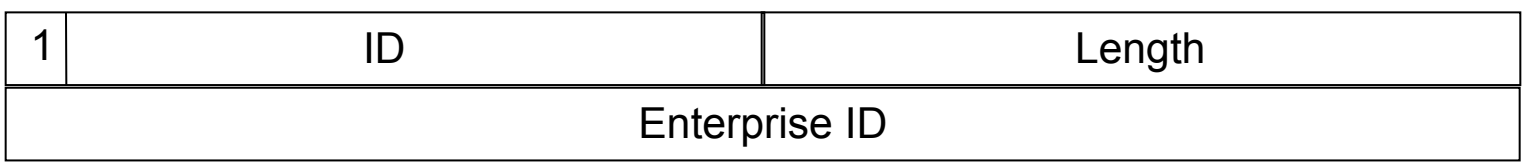
- Template Sets: Announce which type of data is sent by a device
- Data Sets: Contain only data
  - Reference a Template to identify data description
  - ➔ Template Set needs to be transmitted before the Data Set

(Compressed) IPFIX – Templates / Information Model

- Templates carry data description (Template Fields):



- Field ID:
  - ID defines type of data (e.g. temperature, humidity, …)
  - Example: ID = 4711, temperature between 0 and 100 degree Celsius
    - Needs to be defined somewhere
- Custom data types:



- Enterprise ID: Identifies the authority that issued the ID
- Multiple Template Fields form a Template Record
- A Template Record describes a Data Set

(Compressed) IPFIX – Data Sets

- Data Sets contain only data, not meta information
- Templates are needed to decode the data:

Template

Template Set		Length
Template ID		Field Count = 2
0	Temperature	2 Byte
0	Humidity	2 Byte

Data

Data Set	Length
Temperature: **100**	
Humidity: **20**	
Temperature: **97**	
Humidity: **20**	

(Compressed) IPFIX in Constrained Environments

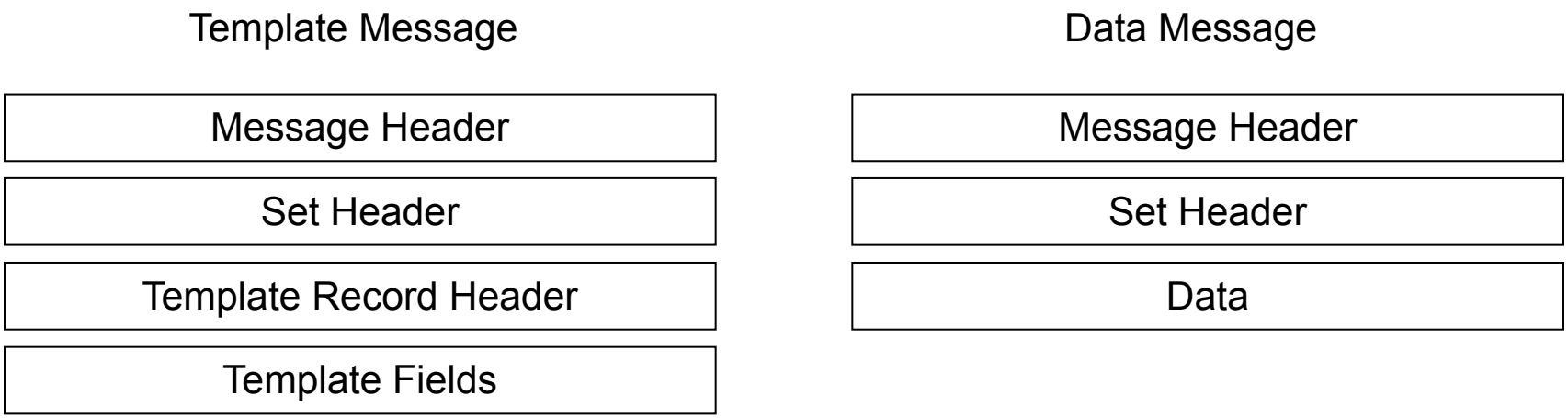
- Usage of (Compressed) IPFIX
  - Sensors perform periodic measurements (no requests)
  - Optional: Sensors aggregate several measurements into a single packet
  - Export packet (power-off wireless transceiver in the mean time)

- Template(s) are pushed first
  - Resend periodically if UDP is used on the transport layer
- Data is pushed afterwards

- Network constraint:
  - IEEE 802.15.4
  - Packet size constraint:
    - 128 Bytes frames
    - ~102 Bytes for application payload

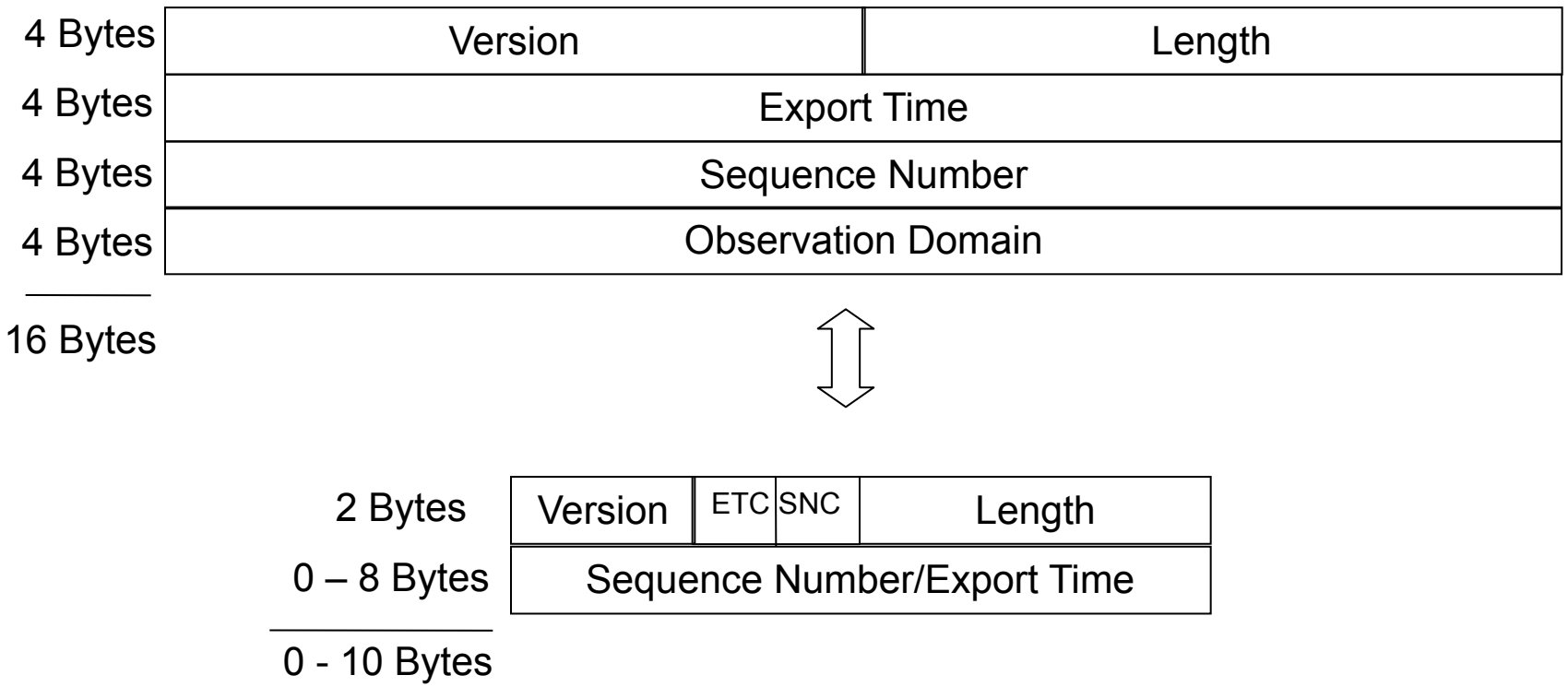
- IPFIX Header sizes too big ➔ Compressed IPFIX

(Compressed) IPFIX – Message Format



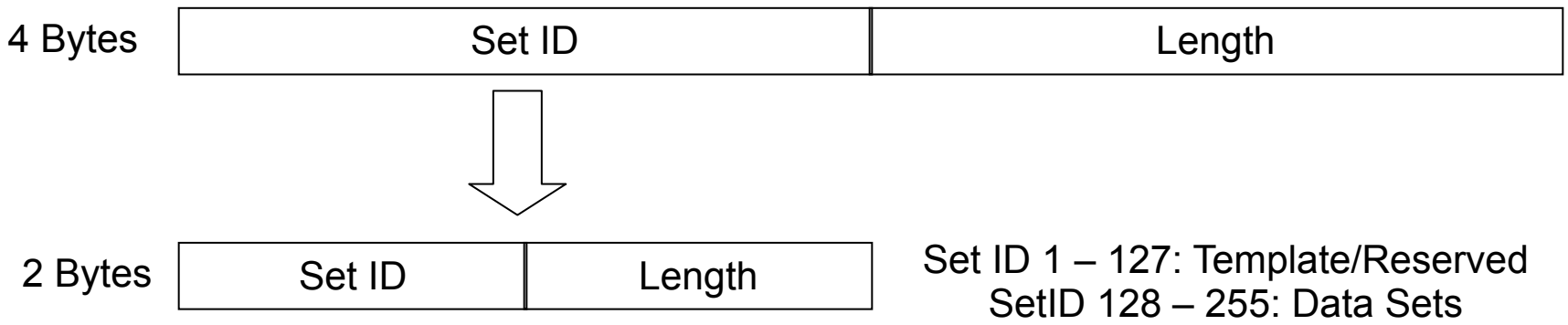
Compressed IPFIX – Header Compression

- Message Header:

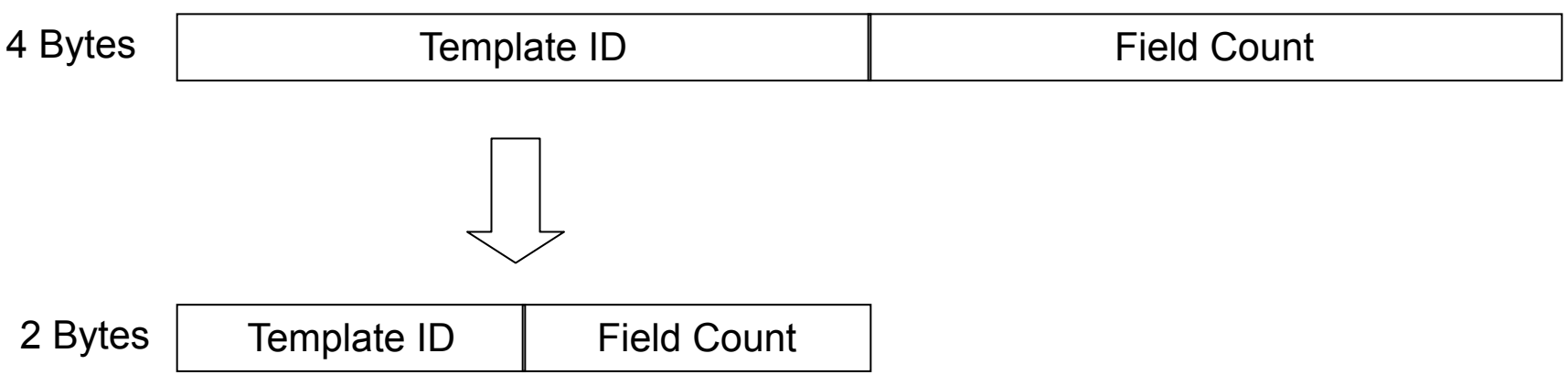


Compressed IPFIX – Set and Template Record Header

- Compressed Set Header

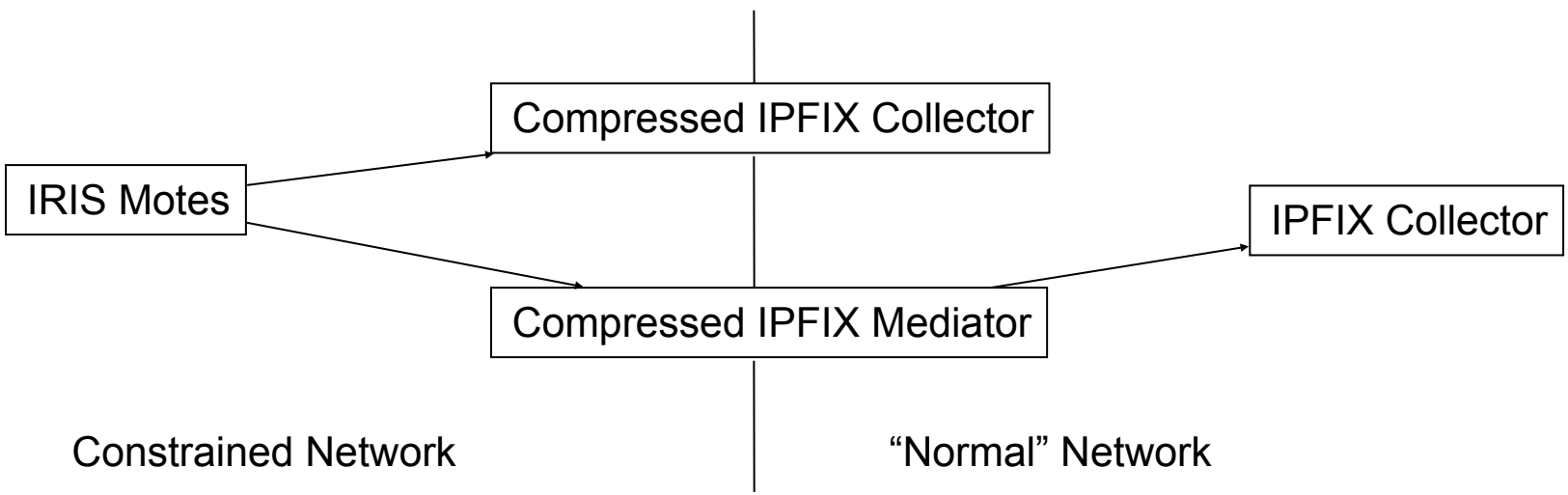


- Compressed Template Record Header



Implementation Status

- Implemented the protocol for Tiny OS and tested it on the IRIS Motes
- Second implementation enhances our standard IPFIX implementation with Compressed IPFIX support
  - Compressed IPFIX Exporter
  - Compressed IPFIX Collector
  - Mediation from Compressed IPFIX → IPFIX



Future Plans

- Introduce (optional) bidirectional communication
  - Sensors may set a bit in the header to indicate that they can receive data
- “Reliable transport” of IPFIX Templates over UDP
  - Templates need to be stored by the Collector to decode Data Sets
    - Templates are therefore now resend periodically (every N Data Sets)
  - Acknowledgements (optional) for these packets
  - Stop resending templates if template has been ACKed
- Use IPFIX to “configure” Templates on the sensor nodes
  - Right now: The template needs to be pre-configured on the device
  - Future: A template can be pushed to the device
  - Right now: Sensor is always sending data
  - Future: Sending can be turned off (similar to publish/subscribe)

Discussion

- Is there a general interest in the protocol?
  - (despite the fact that it is out scope of CoRE right now)
- Should certain features of Compressed IPFIX be part of CoAP?
- Should there be a protocol like Compressed IPFIX in CoRE?

77th IETF: core WG Agenda

09:00	Introduction, Agenda	Chairs (5)
09:05	What is REST?	LD (15)
09:20	1 – CoAP reqts and protocol	ZS (55)
10:15	2 – Bootstrap & Security	RS/CO (45)
11:00	0 – Compressed IPfix	LB (15)
11:15	Other Topics	Chairs (20)