Freie Universität Berlin

Embedded Internet and the Internet of Things WS 12/13

8. Application Layer

Prof. Dr. Mesut Güneş
Distributed, embedded Systems (DES)
Institute of Computer Science
Freie Universität Berlin

Overview

- Introduction
- CoAP
- Query processing
- Summary

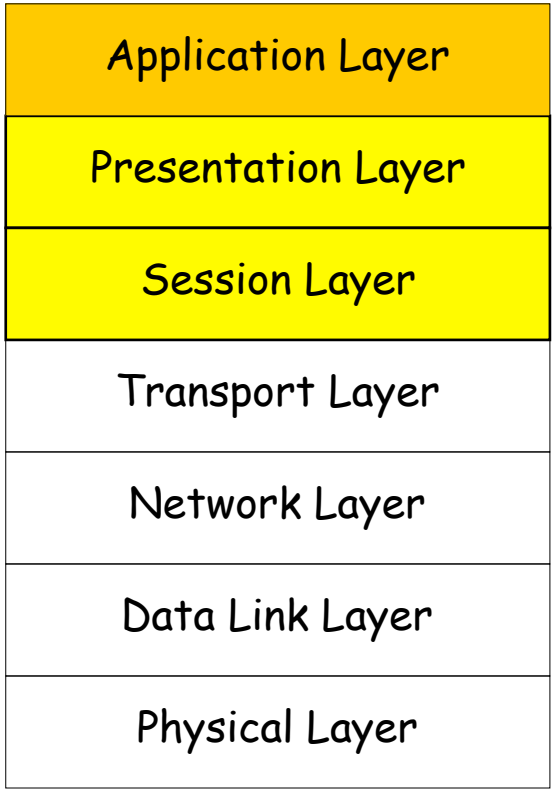
Introduction

Application layer in the Internet

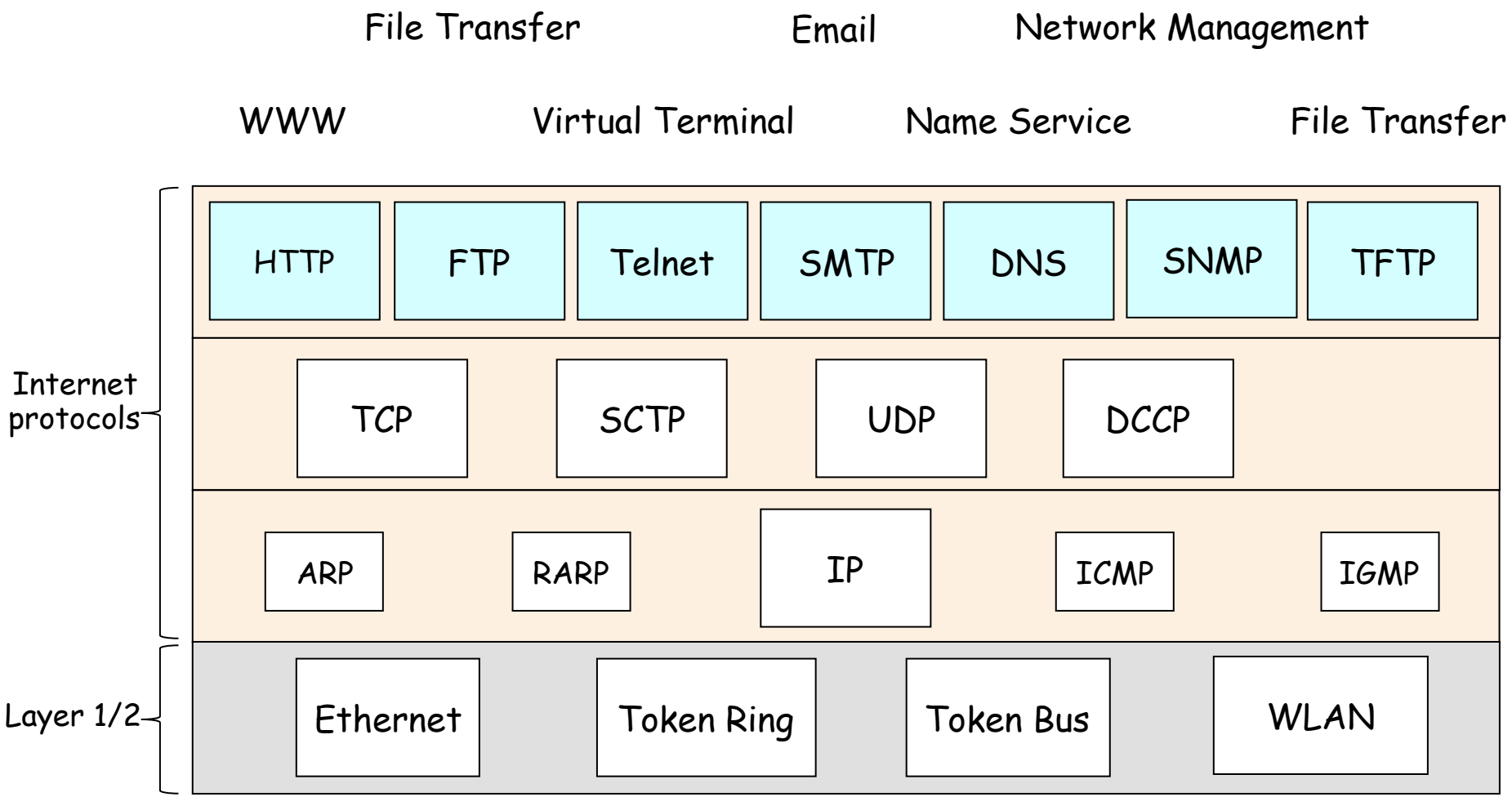

[Telematics]

- Lower layers
  - The meaning of all lower layers is to provide a communication facility for applications
  - Are not really designed for end users
- Application layer
  - Application layer protocols work on top of the transport layer protocols
  - Implement applications for end users
  - A large set of different applications (protocols) with totally different requirements and assumptions
  - According to ISO/OSI three layers, but in the Internet exists only one layer

OSI Reference Model



Application layer in the Internet



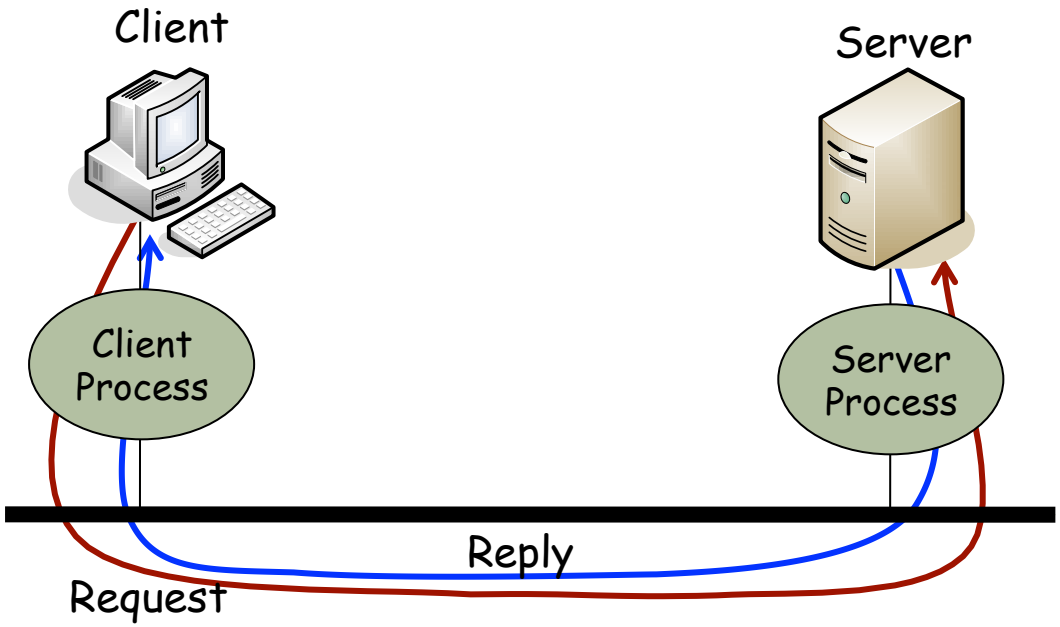
Application layer in the Internet

- Protocols of the application layer are common communication services

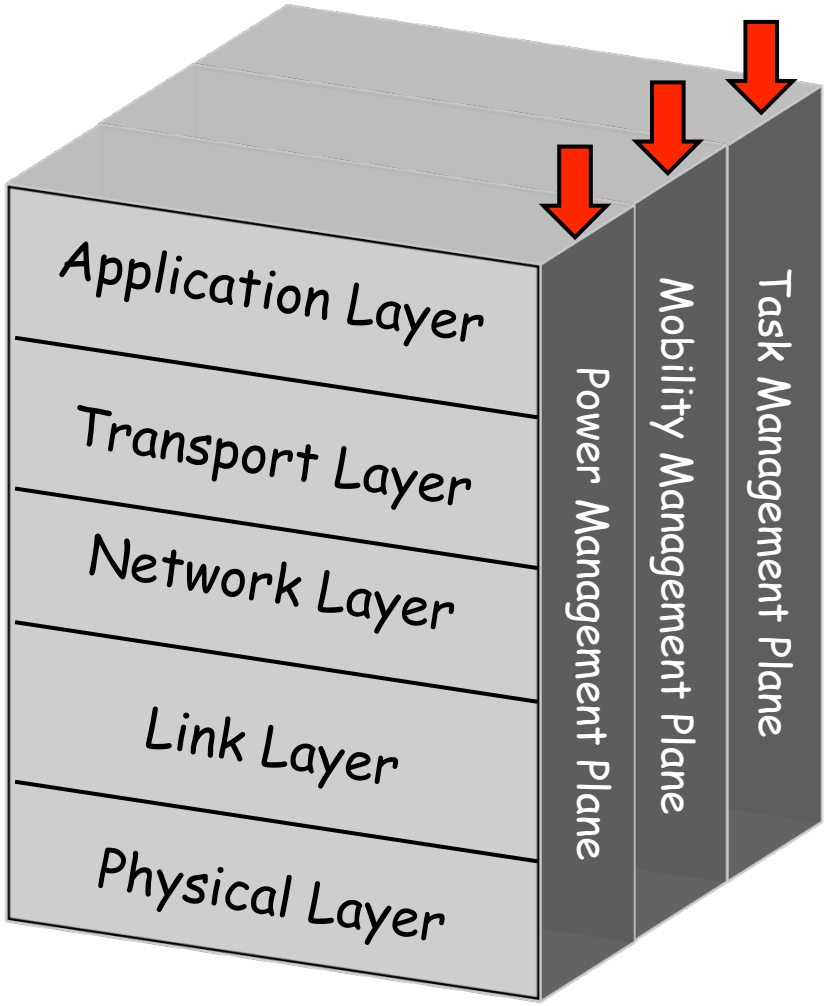
- Protocols of the application layer are defined for special purposes and specify ...
  - the types of the messages
  - the syntax of the message types
  - the semantics of the message types
  - rules for definition, when and how an application process sends a message resp. responses to it

Application layer in the Internet

- Usually **client/server** structure
- Processes on the application layer use TCP(UDP)/IP-Sockets



Protocol stack

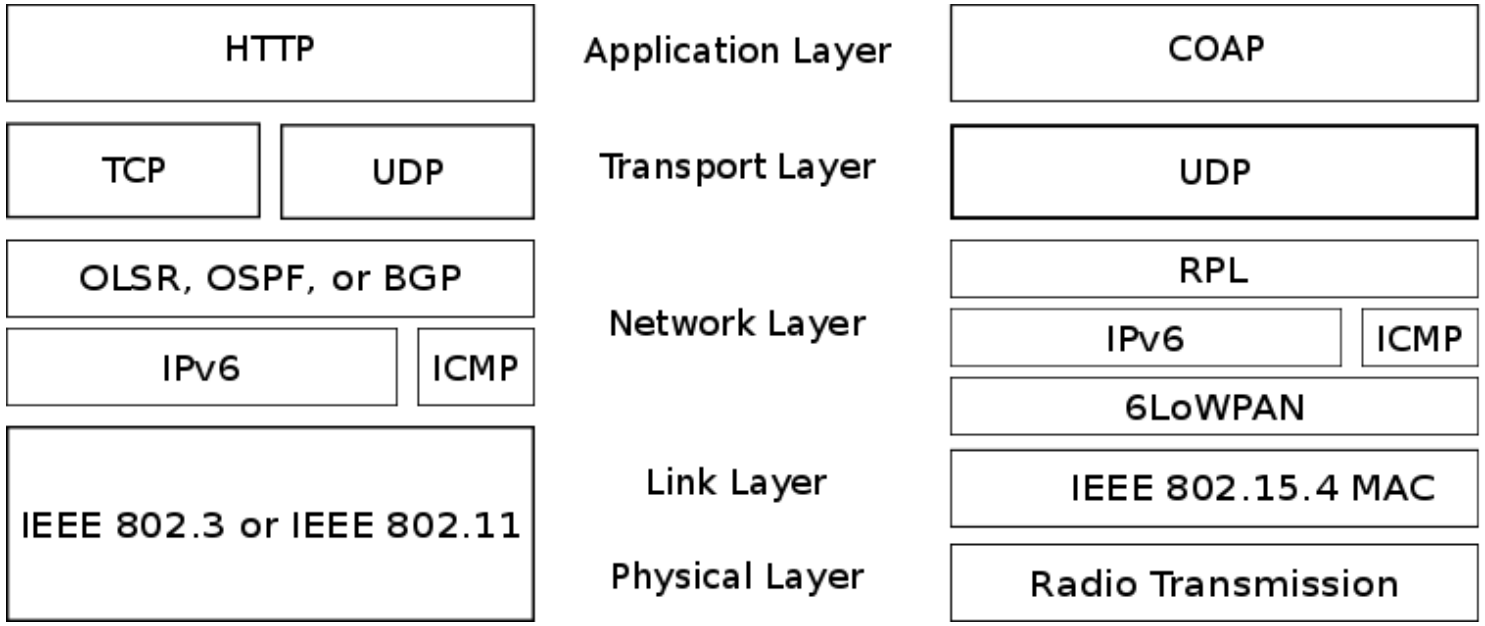


IoT Applications

[Gubbi]

Protocol stack

- Different protocol stack models for the future



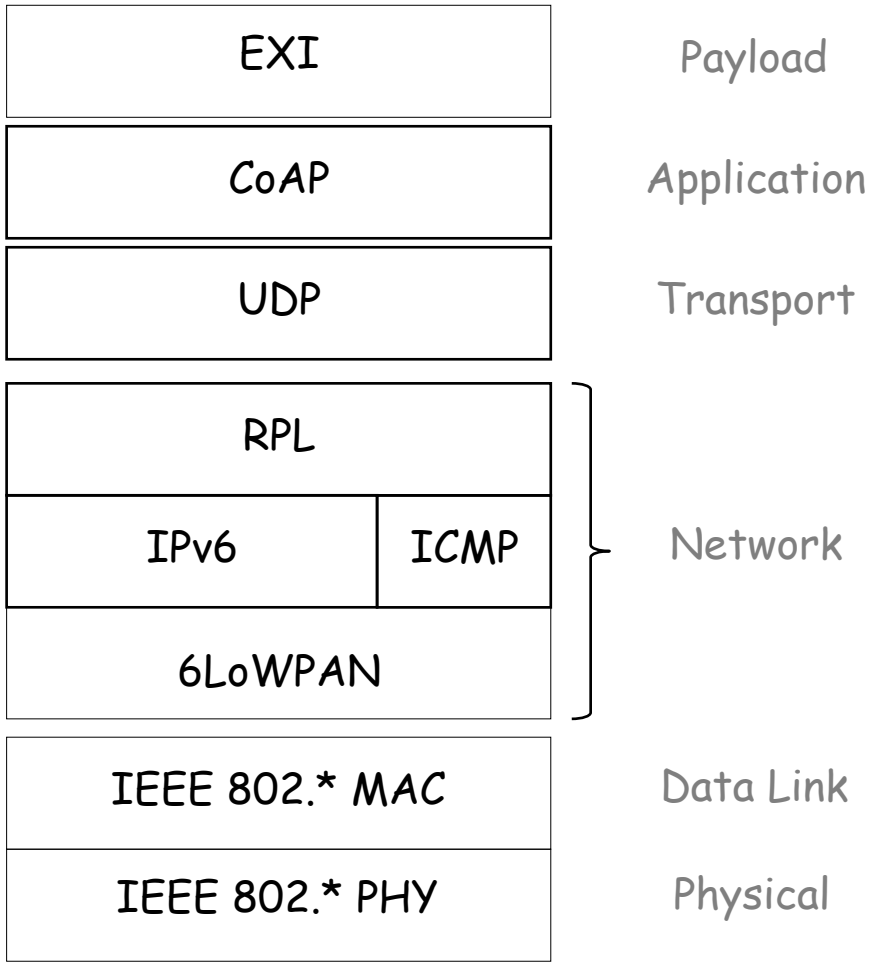
Protocol stack

- Different protocol stack models for the future

XML	Payload	EXI
HTTP	Application	CoAP
TCP	Transport	UDP
IP	Network	6LoWPAN
Ethernet MAC	Data Link	IEEE 802.* MAC
Ethernet PHY	Physical	IEEE 802.* PHY

- W3C Efficient XML Interchange (EXI) format is a binary high performance XML representation for M2M communication.

Protocol stack



CoAP

Constrained Application Protocol (CoAP)

- IETF working group
  Constrained RESTful Environments (core)
- Documents related to CoAP
  - Most specifications are currently work in progress!

- draft-ietf-core-coap-13
  "Constrained Application Protocol (CoAP)", 2012-12-06
- RFC 6690 (draft-ietf-core-link-format)
  "Constrained RESTful Environments (CoRE) Link Format", 2012-08

- See http://datatracker.ietf.org/wg/core/ for a list of all documents.

What is REST?

- REST was defined by Roy T. Fielding in his PhD thesis
  - Co-author of many web protocols

- Key principles of REST (simplified)
  - Give every "thing" (resource) an ID
  - Link things together
  - Use standard methods
  - Resources with multiple representations
  - Communicate statelessly

Constrained Application Protocol (CoAP)

Description of Working Group
(http://datatracker.ietf.org/wg/core/charter/)

"CoRE is providing a framework for resource-oriented applications intended to run on constrained IP networks. A constrained IP network has limited packet sizes, may exhibit a high degree of packet loss, and may have a substantial number of devices that may be powered off at any point in time but periodically "wake up" for brief periods of time.

These networks and the nodes within them are characterized by severe limits on throughput, available power, and particularly on the complexity that can be supported with limited code size and limited RAM per node. More generally, we speak of constrained networks whenever at least some of the nodes and networks involved exhibit these characteristics. Low-Power Wireless Personal Area Networks (LoWPANs) are an example of this type of network. Constrained networks can occur as part of home and building automation, energy management, and the Internet of Things."

Constrained Application Protocol (CoAP)

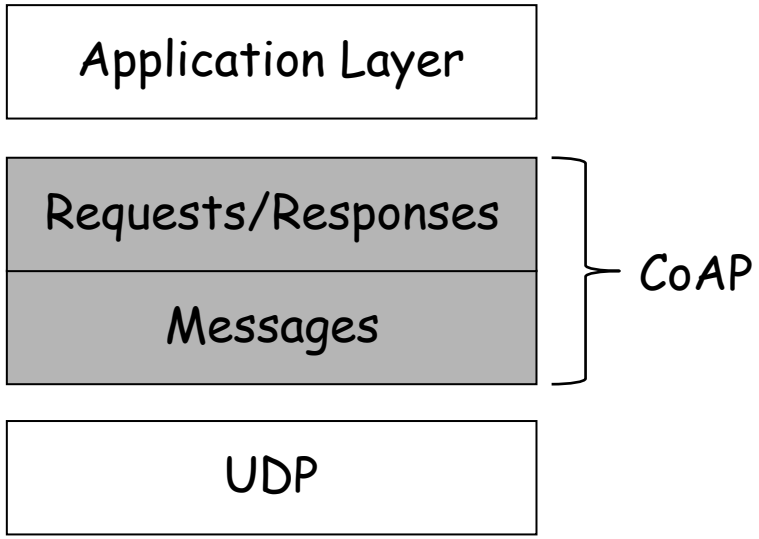
- Features
  - Constrained web protocol fulfilling M2M requirements.
  - UDP binding with optional reliability supporting unicast and multicast requests.
  - Asynchronous message exchanges.
  - Low header overhead and parsing complexity.
  - URI and Content-type support.
  - Simple proxy and caching capabilities.
  - A stateless HTTP mapping, allowing proxies to be built providing access to CoAP resources via HTTP in a uniform way or for HTTP simple interfaces to be realized alternatively over CoAP.
  - Security binding to Datagram Transport Layer Security (DTLS).

CoAP: Message types

Type	Description
CON	Confirmable Message Each confirmable message is acknowledged either by a message type of "Ack" or "Reset".
NON	Non-Confirmable Message For messages that do not require an acknowledgement. This is particularly true for messages that are repeated regularly for application requirements, such as repeated readings from a sensor where eventual success is sufficient.
Ack	Acknowledgement Message An Ack acknowledges that a specific CON message arrived. It does not indicate success or failure of any encapsulated request.
Reset	Reset Message A Reset message indicates that a specific message (CON or NON) was received, but some context is missing to properly process it. This condition is usually caused when the receiving node has rebooted and has forgotten some state that would be required to interpret the message. Provoking a Reset message (e.g., by sending an empty Confirmable message) is also useful as an inexpensive check of the liveness of an endpoint ("CoAP ping").

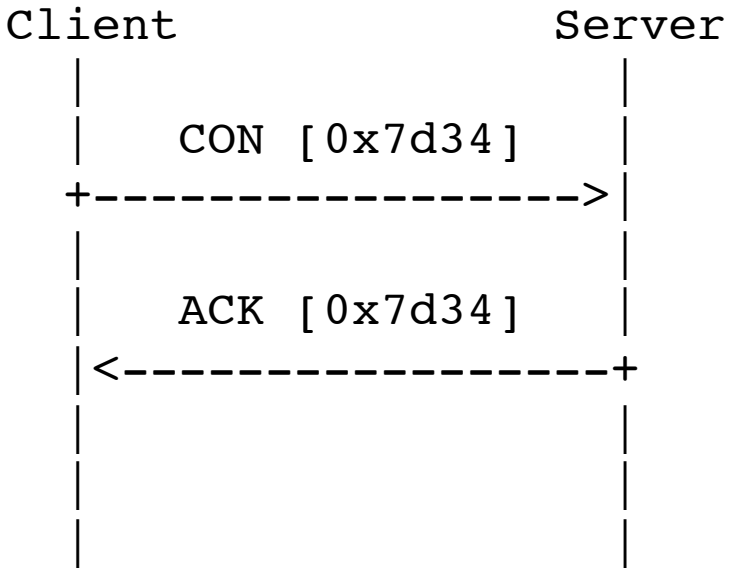
CoAP

- Abstract layering of CoAP

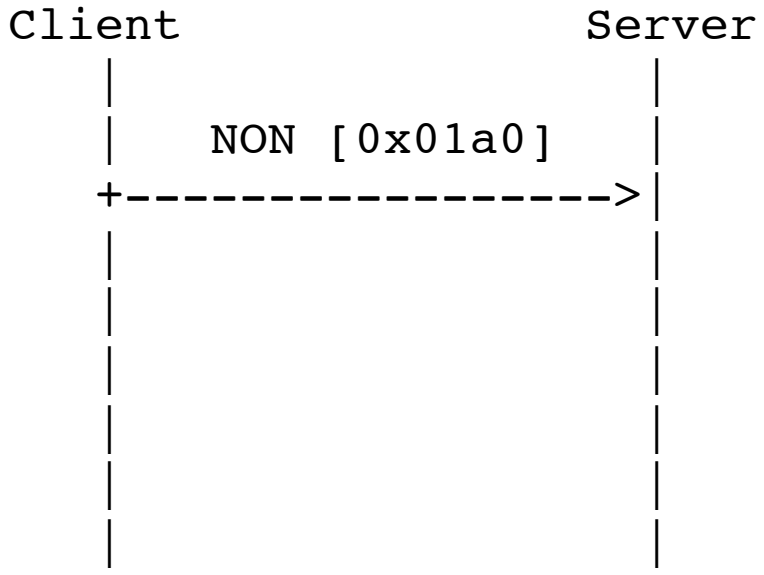


CoAP

- Reliable message transmission



- Unreliable message transmission



CoAP

- Request/Response Model

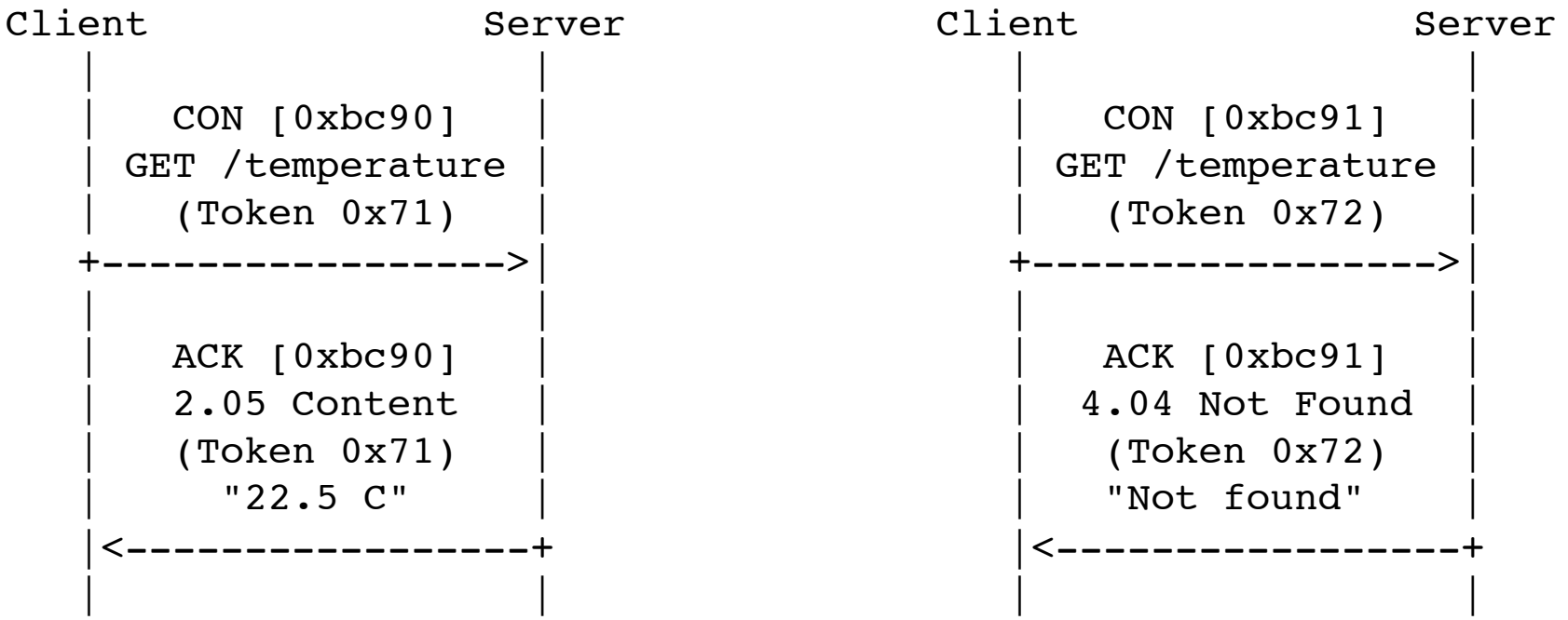


Figure 4: Two GET requests with piggy-backed responses

A Token is used to match responses to requests independently from the underlying messages

CoAP

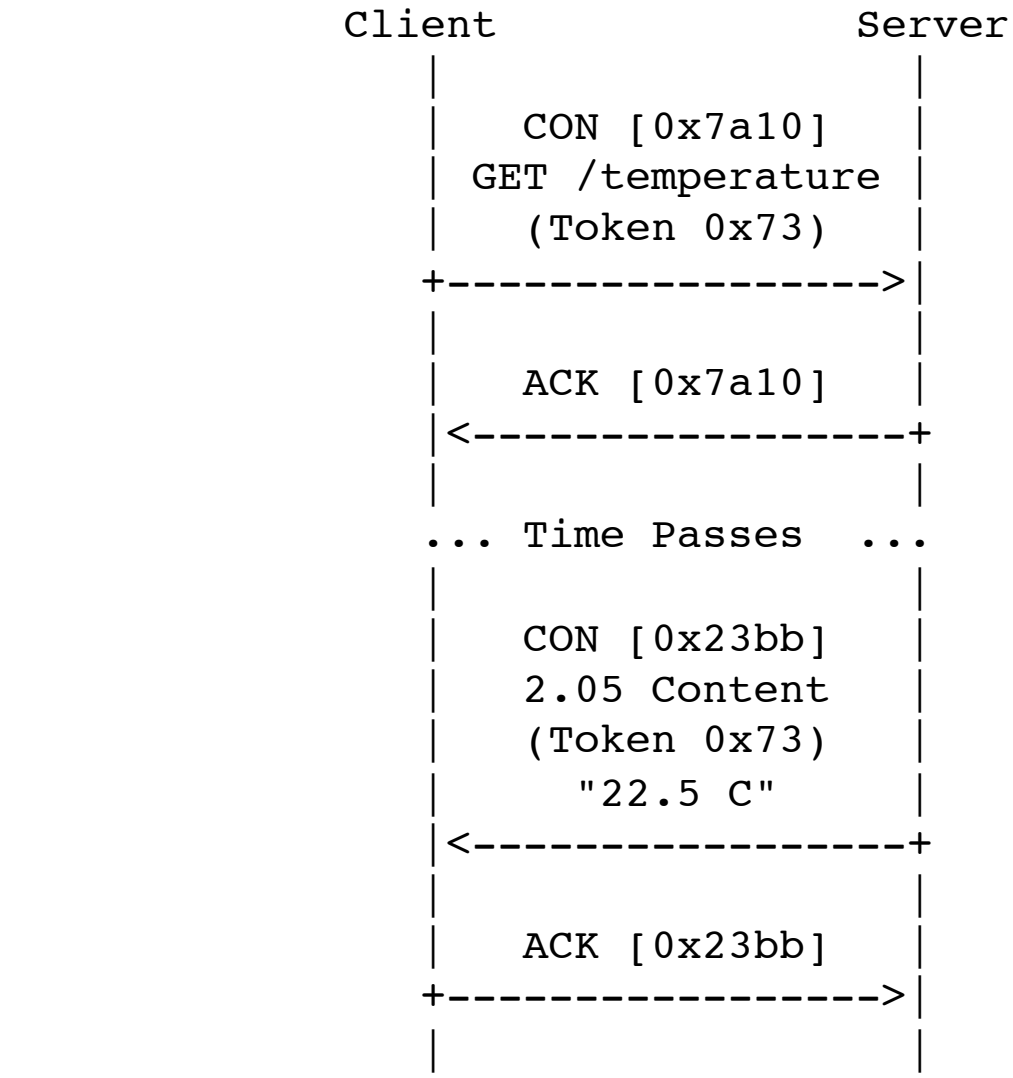


Figure 5: A GET request with a separate response

CoAP

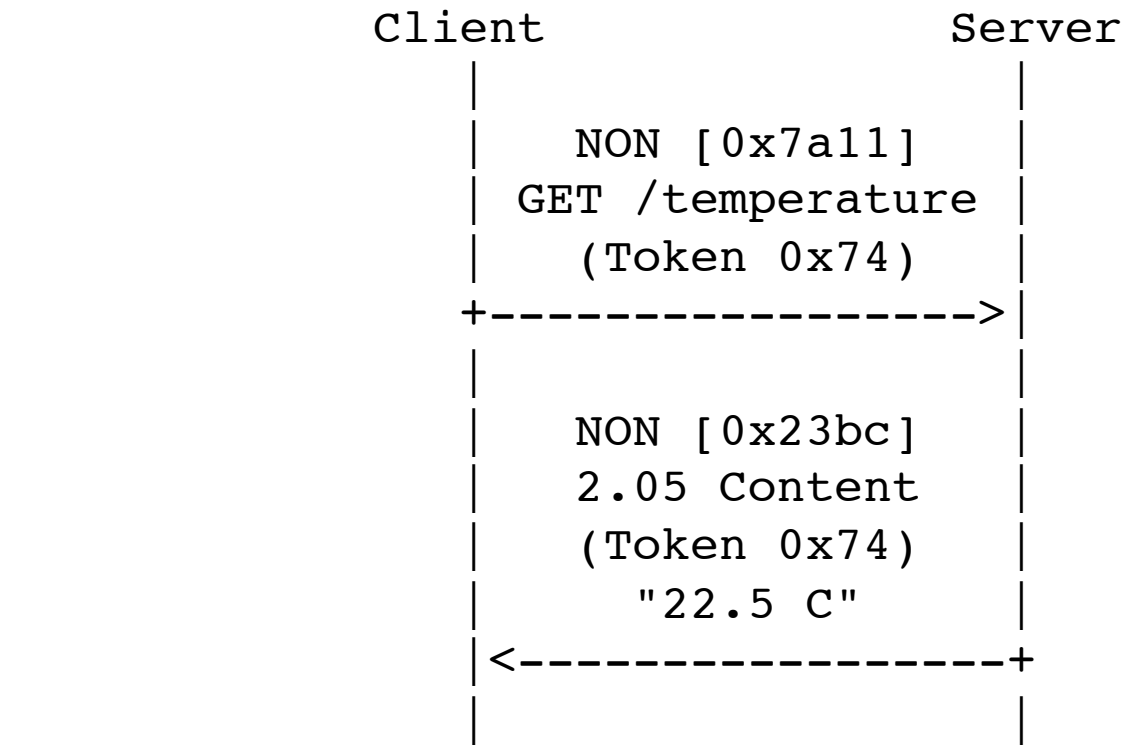


Figure 6: A NON request and response

Message format

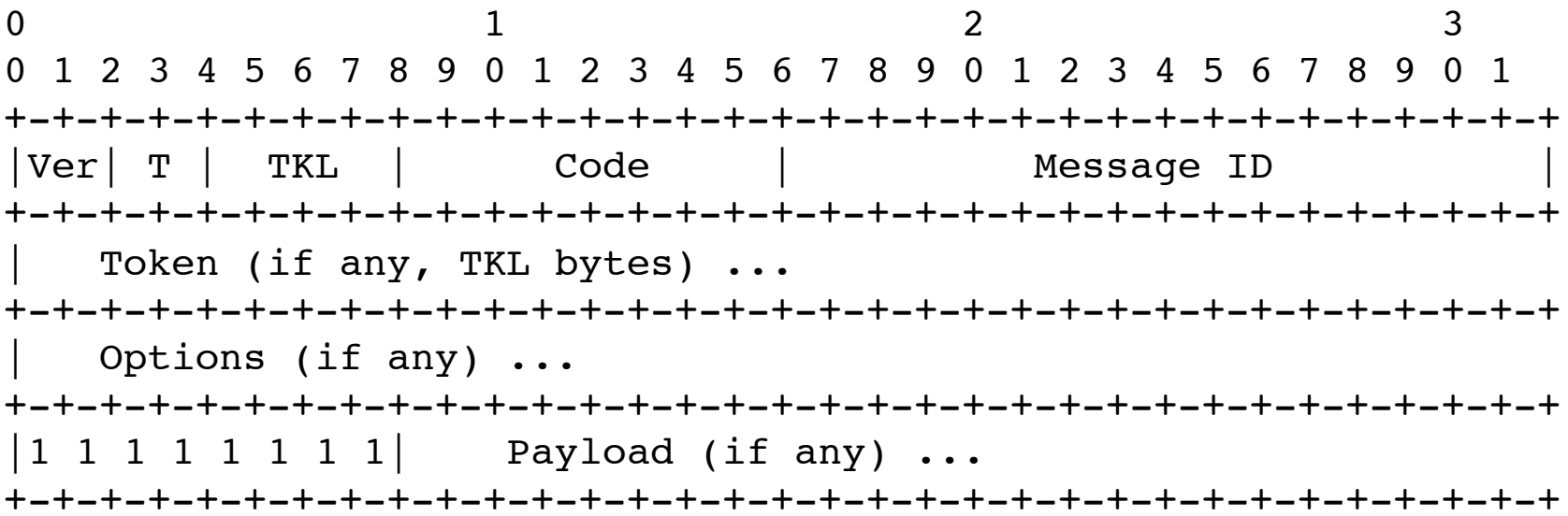
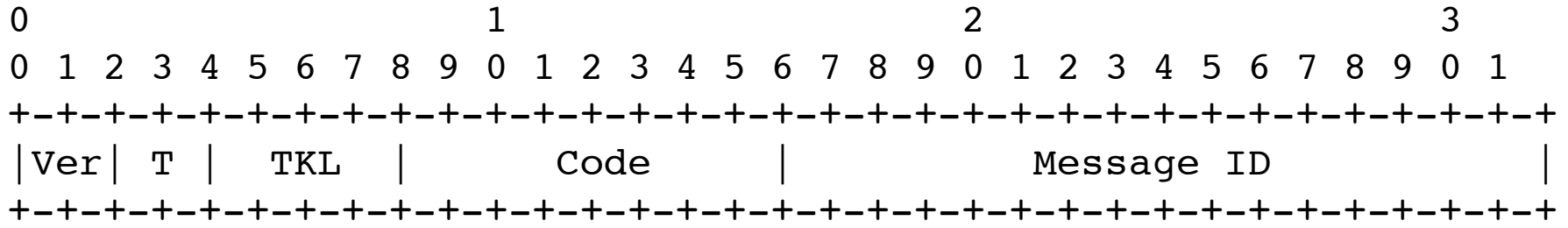


Figure 7: Message Format

Message format: The header



- Version (Ver): Indicates the CoAP version number, currently 1.
- Type (T): Indicates the type of message
  - 0: Confirmable
  - 1: Non-Confirmable
  - 2: Acknowledgement
  - 3: Reset
- Token Length (TKL): Length of the variable-length Token field (0-8 bytes).
- Code: Message code
  - 0: empty
  - 1-31: request
  - 64-191: response
  - else: reserved
- Message ID: 16-bit unsigned integer in network byte order.
  - Used for the detection of message duplication.
  - To match messages of type Ack/Reset to CON/NON messages

Message transmission

- A message carries
  - A request
  - A response
  - Is empty

- A CoAP endpoint is the source or destination of a CoAP message.
- Endpoint is identified by (IP address, UDP port)

Message transmission

- CoAP messages are exchanged asynchronously between CoAP endpoints.
- They are used to transport CoAP requests and responses
- As CoAP is bound to non-reliable transports such as UDP, CoAP messages may arrive
  - out of order
  - appear duplicated
  - go missing without notice

-> Mechanism for reliable transport required!

Message transmission

- CoAP implements a lightweight reliability mechanism, without trying to re-create the full feature set of a transport protocol like TCP.

- It has the following features:
  - Simple stop-and-wait retransmission reliability with exponential back-off for Confirmable messages.

  - Duplicate detection for both Confirmable and Non-Confirmable messages.

Message transmission: Reliability

- The reliable transmission of a message is initiated by marking the message as Confirmable in the CoAP header.
- A Confirmable message always carries either a request or response and MUST NOT be empty.
- A recipient MUST acknowledge such a message with an Acknowledgement message or, if it lacks context to process the message properly, MUST reject it.
- Rejecting a Confirmable message is effected by sending a matching Reset message and otherwise ignoring it.
- The Acknowledgement message MUST echo the Message ID of the Confirmable message, and MUST carry a response or be empty.
- The Reset message MUST echo the Message ID of the confirmable message, and MUST be empty.

Message transmission: Reliability

- The sender retransmits the Confirmable message at exponentially increasing intervals, until it receives an acknowledgement (or Reset message), or runs out of attempts.
- Retransmission is controlled by
  - a timeout
  - a retransmission counter

- If the retransmission counter reaches MAX_RETRANSMIT on a timeout, or if the endpoint receives a Reset message, then the attempt to transmit the message is canceled and the application process informed of failure.

Congestion control

- Basic congestion control for CoAP is provided by the exponential back-off mechanism
- Clients (including proxies) MUST strictly limit the number of simultaneous outstanding interactions that they maintain to a given server

Transmission parameters

- Message transmission is controlled by the following parameters:

Name	Default value
ACK_TIMEOUT	2 seconds
ACK_RANDOM_FACTOR	1.5
MAX_RETRANSMIT	4
NSTART	1
DEFAULT_LEISURE	5 seconds
PROBING_RATE	1 byte/second

Transmission parameters

- Time Values derived from Transmission Parameters
- MAX_TRANSMIT_SPAN is the maximum time from the first transmission of a confirmable message to its last retransmission.

```
ACK_TIMEOUT * (2**MAX_RETRANSMIT - 1) * ACK_RANDOM_FACTOR
```

- For the default transmission parameters, the value is 2*15*1.5 = 45 seconds

Transmission parameters

- MAX_TRANSMIT_WAIT is the maximum time from the first transmission of a confirmable message to the time when the sender gives up on receiving an acknowledgement or reset.

```
ACK_TIMEOUT * (2 ** (MAX_RETRANSMIT + 1) - 1) * ACK_RANDOM_FACTOR
```

- For the default transmission parameters, the value is 2*31*1.5 = 93 seconds.

Transmission parameters

- MAX_LATENCY is the maximum time a datagram is expected to take from the start of its transmission to the completion of its reception.

- PROCESSING_DELAY is the time a node takes to turn around a confirmable message into an acknowledgement.

- MAX_RTT is the maximum round-trip time, or:

$$2 * \text{MAX\_LATENCY} + \text{PROCESSING\_DELAY}$$

Transmission parameters

- EXCHANGE_LIFETIME is the time from starting to send a confirmable message to the time when an acknowledgement is no longer expected, i.e. message layer information about the message exchange can be purged.

(ACK_TIMEOUT * (2 ** MAX_RETRANSMIT - 1) * ACK_RANDOM_FACTOR) + (2 * MAX_LATENCY) + PROCESSING_DELAY

248 seconds with the default transmission parameters

Transmission parameters

- NON_LIFETIME is the time from sending a non-confirmable message to the time its message-ID can be safely reused.

$$MAX\_TRANSMIT\_SPAN + MAX\_LATENCY$$

- 145 seconds with the default transmission parameters

Request/Response Semantics

- CoAP operates under a similar request/response model as HTTP
  - A CoAP endpoint in the role of a "client" sends one or more CoAP requests to a "server", which services the requests by sending CoAP responses.
  - Unlike HTTP, requests and responses are not sent over a previously established connection, but exchanged asynchronously over CoAP messages.

Request/Response Semantics

- Requests
  - A CoAP request consists of the method to be applied to the resource, the identifier of the resource, a payload and Internet media type (if any), and optional meta-data about the request.
  - CoAP supports the basic methods of GET, POST, PUT, DELETE, which are easily mapped to HTTP.
  - They have the same properties as in HTTP:
    - safe (only retrieval)
    - idempotent (you can invoke it multiple times with the same effects)

- Responses
  - A response is identified by the Code field in the CoAP header being set to a Response Code.

Request/Response Semantics

- The response codes are designed to be extensible

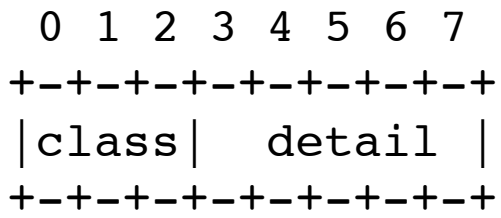


Figure 9: Structure of a Response Code

- There are 3 classes:
  - 2 - Success: The request was successfully received, understood, and accepted.
  - 4 - Client Error: The request contains bad syntax or cannot be fulfilled.
  - 5 - Server Error: The server failed to fulfill an apparently valid request.

Request/Response Semantics

- As a human readable notation for specifications and protocol diagnostics, the numeric value of a response code is indicated by giving the upper three bits in decimal, followed by a dot and then the lower five bits in a two-digit decimal.
- E.g., "Not Found" is written as 4.04 -- indicating a value of hexadecimal 0x84 or decimal 132.

Options

- CoAP defines a single set of options that are used in both requests and responses:
  - Content-Format
  - Etag
  - Location-Path
  - Location-Query
  - Max-Age
  - Proxy-Uri
  - Proxy-Scheme
  - Uri-Host
  - Uri-Path
  - Uri-Port
  - Uri-Query
  - Accept
  - If-Match
  - If-None-Match

Caching

- CoAP endpoints MAY cache responses in order to reduce the response time and network bandwidth consumption on future, equivalent requests.
- The goal of caching in CoAP is to reuse a prior response message to satisfy a current request.

- Freshness Model
  - When a response is "fresh" in the cache, it can be used to satisfy subsequent requests without contacting the origin server, thereby improving efficiency.
  - The mechanism for determining freshness is for an origin server to provide an explicit expiration time in the future, using the Max-Age Option

Methods

- **GET**
  - The GET method retrieves a representation for the information that currently corresponds to the resource identified by the request URI.
  - The GET method is safe and idempotent.
- **POST**
  - The POST method requests that the representation enclosed in the request be processed.
  - POST is neither safe nor idempotent.
- **PUT**
  - The PUT method requests that the resource identified by the request URI be updated or created with the enclosed representation.
  - PUT is not safe, but is idempotent.
- **DELETE**
  - The DELETE method requests that the resource identified by the request URI be deleted.
  - DELETE is not safe, but is idempotent.

CoAP URIs

- CoAP uses the "coap" and "coaps" URI schemes for identifying CoAP resources and providing a means of locating the resource.
- CoAP URI scheme
  - "coap:" "//" host [":" port] path-abempty ["?" query]
- CoAPs URI scheme
  - "coaps:" "//" host [":" port] path-abempty ["?" query]

- Examples (all identical)
  - coap://example.com:5683/~sensors/temp.xml
  - coap://EXAMPLE.com/%7Esensors/temp.xml
  - coap://EXAMPLE.com:/%7esensors/temp.xml

Discovery

- Service Discovery
  - A server is discovered by a client by the client knowing or learning a URI that references a resource in the namespace of the server.
  - Alternatively, clients can use Multicast CoAP and the "All CoAP Nodes" multicast address to find CoAP servers.

- Resource Discovery
  - A CoAP client can query a server for its list of hosted resources.
  - It is up to the server which resources are made discoverable (if any).

Query processing

Query processing

- The classical way of developing applications
  - Write programs in low-level languages like C
  - Difficult, error prone, and expensive
  - Most users are computer/network experts

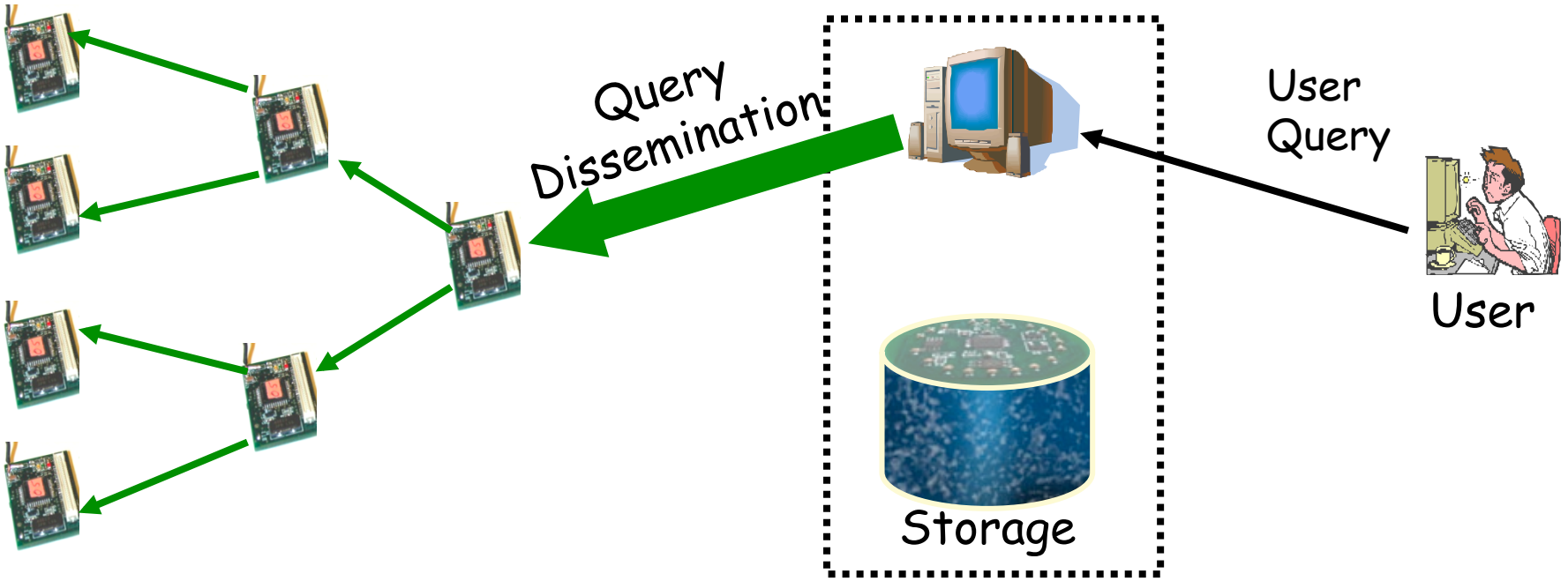
- Consider the network (e.g. WSN) as a distributed database system
  - Alternative way application "development"
  - Applications can "created" in the form of a "query"

- But: Traditional DB approaches are not applicable

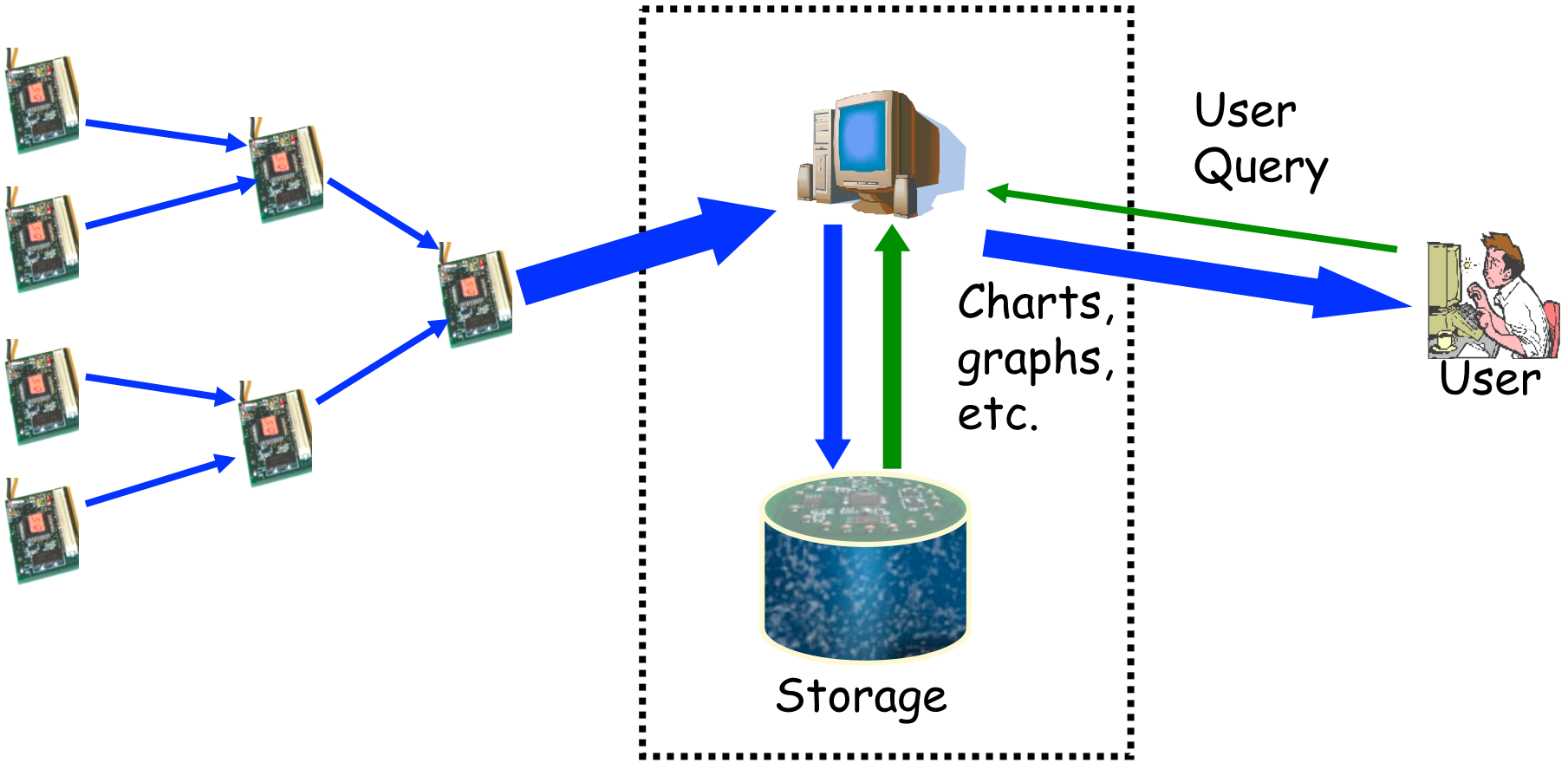
Query processing

- Properties of wireless multi-hop networks
  - **Streaming data**: Nodes produce data continuously
  - **Real-time processing**: Generated data represent real-time events
  - **Communication errors**: Wireless multi-hop communication affects reliability
  - **Uncertainty**: Generated data contains noise from environment
  - **Limited (disk, memory) space**: Generated data cannot be stored for long time
  - **Processing vs. communication**: Processing is cheap in comparison to communication -> Process more, communicate less!

Query processing

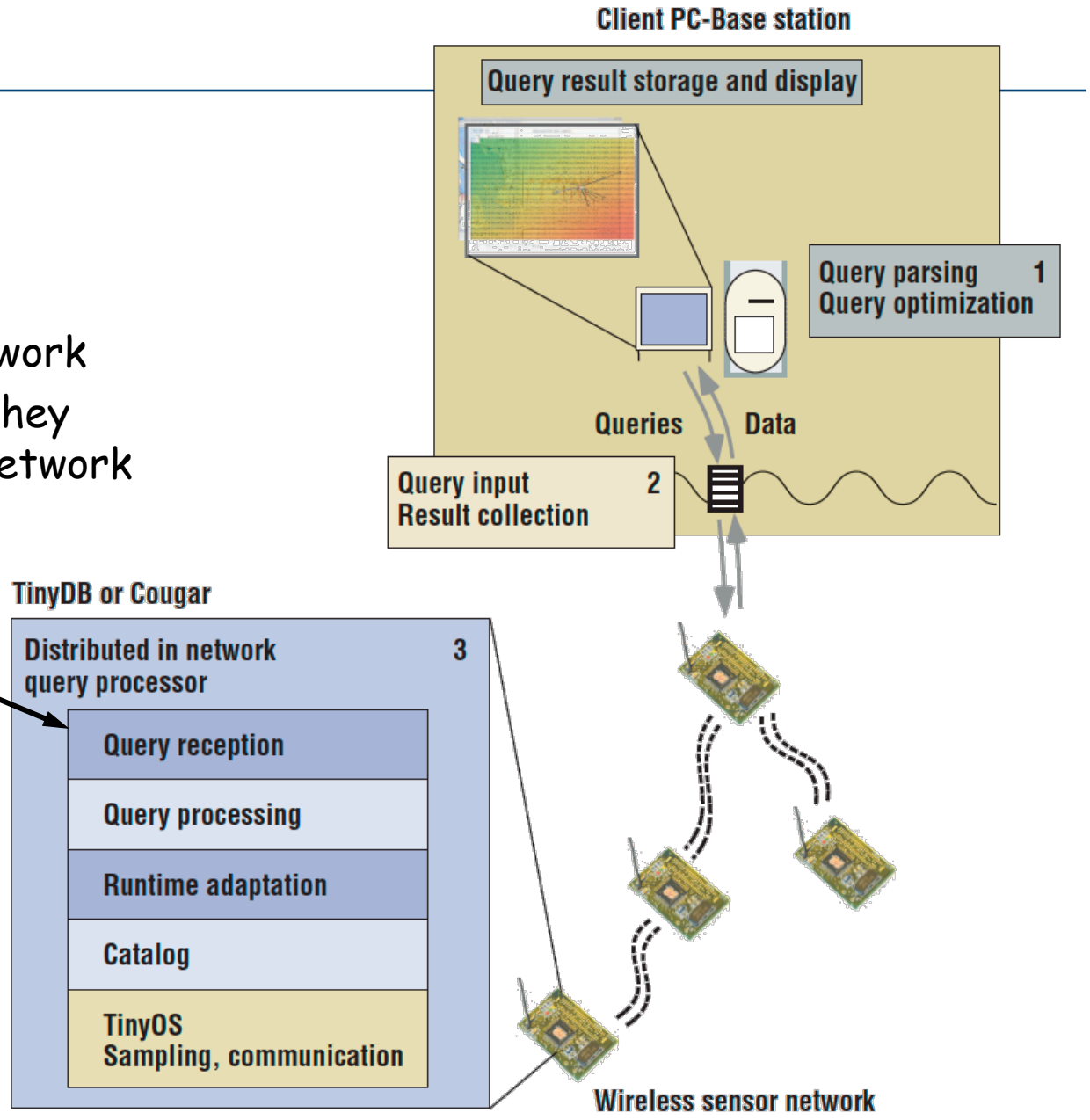


Query processing



Query processing

- Architecture
  - Server-side
    - parse query
    - deliver into the network
    - collects results as they stream out of the network
  - Sensor-side



[Madden04]

Query processing

Query representation

Query representation

- The representation of the query is a key factor
  - Representation for the user
  - Representation for processing algorithms
- Representation is send to the (sensor) nodes
  - Representation -> interest -> task

- Example:
  - User sends a query and gets response from sensor nodes
- Which area has humidity higher than 50?
  - Type = humidity
  - Timestamp = 28/01/2013/19:38:27
  - Location = [60N,120W]
  - Humidity > 50

Query representation

- Example: Animal Tracking Query
  - Type = four legged animal (detect animal location)
  - Interval = 30 s (send back events every 30 s)
  - Duration = 1h (... for the next hour)
  - Rec = [-100,100,200,400] (from sensors within the rectangle)

- Sensor detecting the animal generates the data
  - Type = four legged animal (type of animal seen)
  - Instance = elephant (instance of this type)
  - Location = (125,220) (node location)
  - Intensity = 0.6 (signal amplitude measure)
  - Confidence = 0.85 (confidence in the match)
  - Timestamp = 19:41:40 (event generation time)

Query representation

Alternative query representation based on SQL

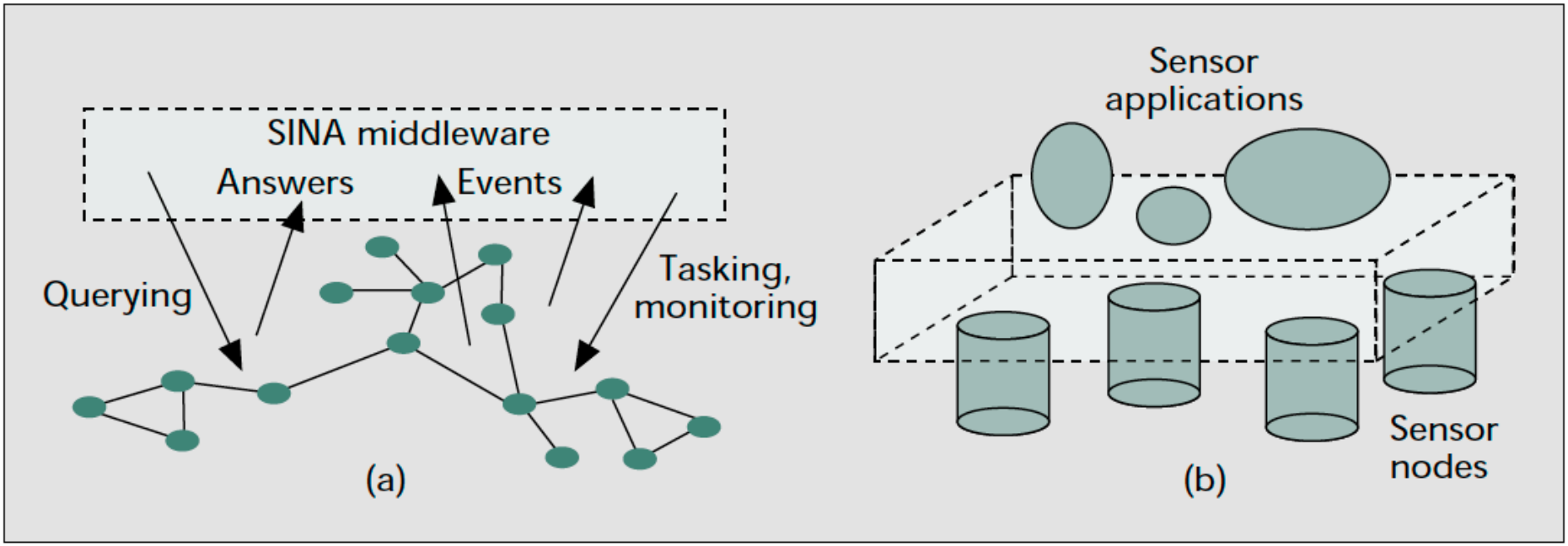
```
Select [ task, time, location, [distinct | all], amplitude,
       [ [avg | min |max | count | sum ] (amplitude)]]
From   [any , every ]
Where  [ power available [<|>] PA      |
         location [in | not in] RECT  |
         tmin < time < tmax            |
         task = t | amplitude [<|==|>] a ]
Group by task
Based on [time limit = lt | packet limit = lp |
          resolution = r  | region = xy ]
```

Query processing

Sensor Query and Tasking Language (SQTL)

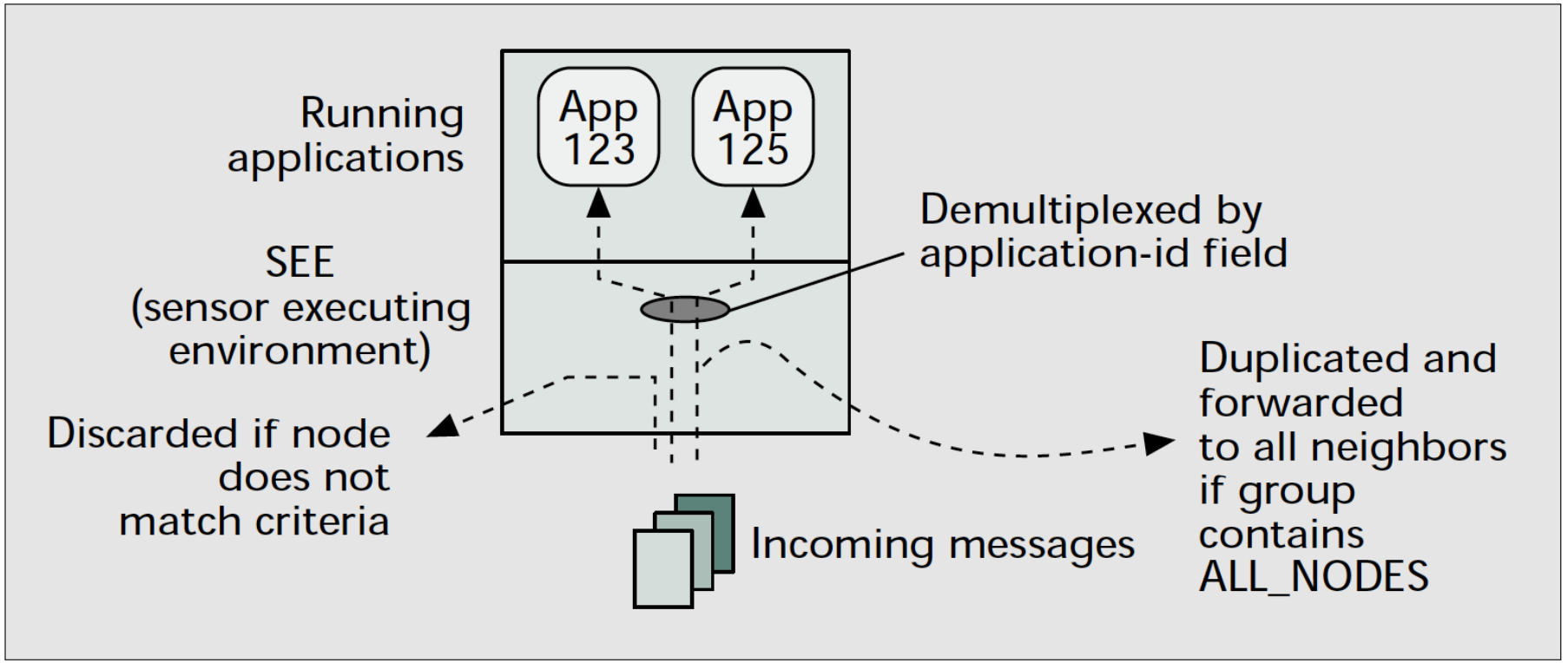
Sensor Query and Tasking Language (SQTL)

- Sensor Information Networking Architecture and Applications (SINA)
  - Middleware for sensor networks
  - SQTL -> Query language
  - Sensor Execution Environment (SSE) -> Runs on each node



SQTL

- Cooperation of SQTL and SSE
  - Dispatching of messages received by a sensor node



SQTL and SSE

- Once an SQTL script is injected into the network, the script is pushed to other nodes in order to complete the assigned task.
- A tell message is generated after a result is produced at each individual node and is delivered back
- SEE is responsible for
  - Demultiplexing of incoming SQTL messages
  - Transmission of outgoing SQTL messages from all running applications
    - Outgoing messages are distributed to target node(s) specified in the receiver argument through the underlying communication mechanism.
  - Translation of an attribute-based name into a unique, numeric link-layer address

SQTL

- SQTL is a procedural scripting language
- It provides …
  - **Sensor access**: getTemperature(), turnOn(), turnOff()
  - **Location awareness**: isNeighbor(), getPosition(), isNorthOf()
  - **Communication**: tell(), execute(), send()

SQTL

- Arguments used by actions in an SQTL wrapper

Argument	Meaning
`sender`	The sender of an SQTL message wrapper

`receiver`
`group`
 	
`criteria`
	Potential receivers specify by two subarguments:
	Subargument of receiver to specify group of receiver; its possible value can be one of `ALL_NODES`, or `NEIGHBORS`
	Subargument of receiver to specify selection criteria of receivers

`application-id`	A unique ID for each application in the same sensor network
`num-hop`	Number of hops away from a gateway node
`language`	Specify a language used in `content`
`content`	A payload containing a program, a message, or return values

`with (optional)`
 	
`parameter`
`type`
`name`
`value`
	Tuples of parameters used in the program passed from sender to receiver
	Repeatable subargument of `with`
	Data type of the parameter
	Name of the parameter
	Value of the parameter

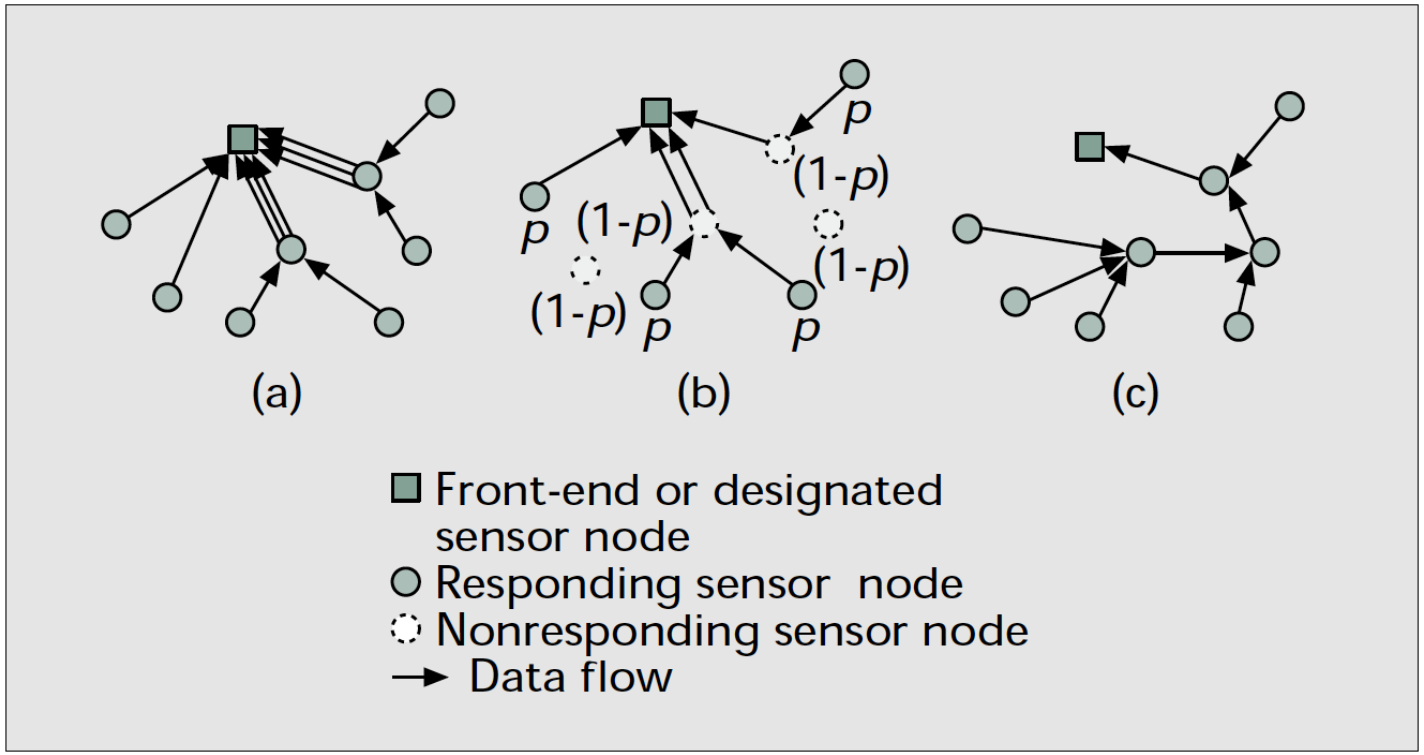
SQTL: Example

- Example for temperature reading application
  - Sink is interested in the maximum temperature

```
(execute
    :sender SINK
    :receiver (:group NODE(1) :criteria TRUE)
    :application-id 123
    :language SQL
    :content ( SELECT Max(getTemperature())
               FROM ALL_Nodes
             )
)
```

SQTL: Response implosion problem

a) The response implosion problem
b) Reduction of responses by probabilistic reply
c) Aggregation at intermediate nodes



SQTL

- By using the upon command, a programmer can create an "event handling block" for 3 types of events:

1. Events generated when a message is received by a sensor node (RECEIVE)

2. Events triggered periodically (EVERY)

3. Events caused by the expiration of a timer (EXPIRE)

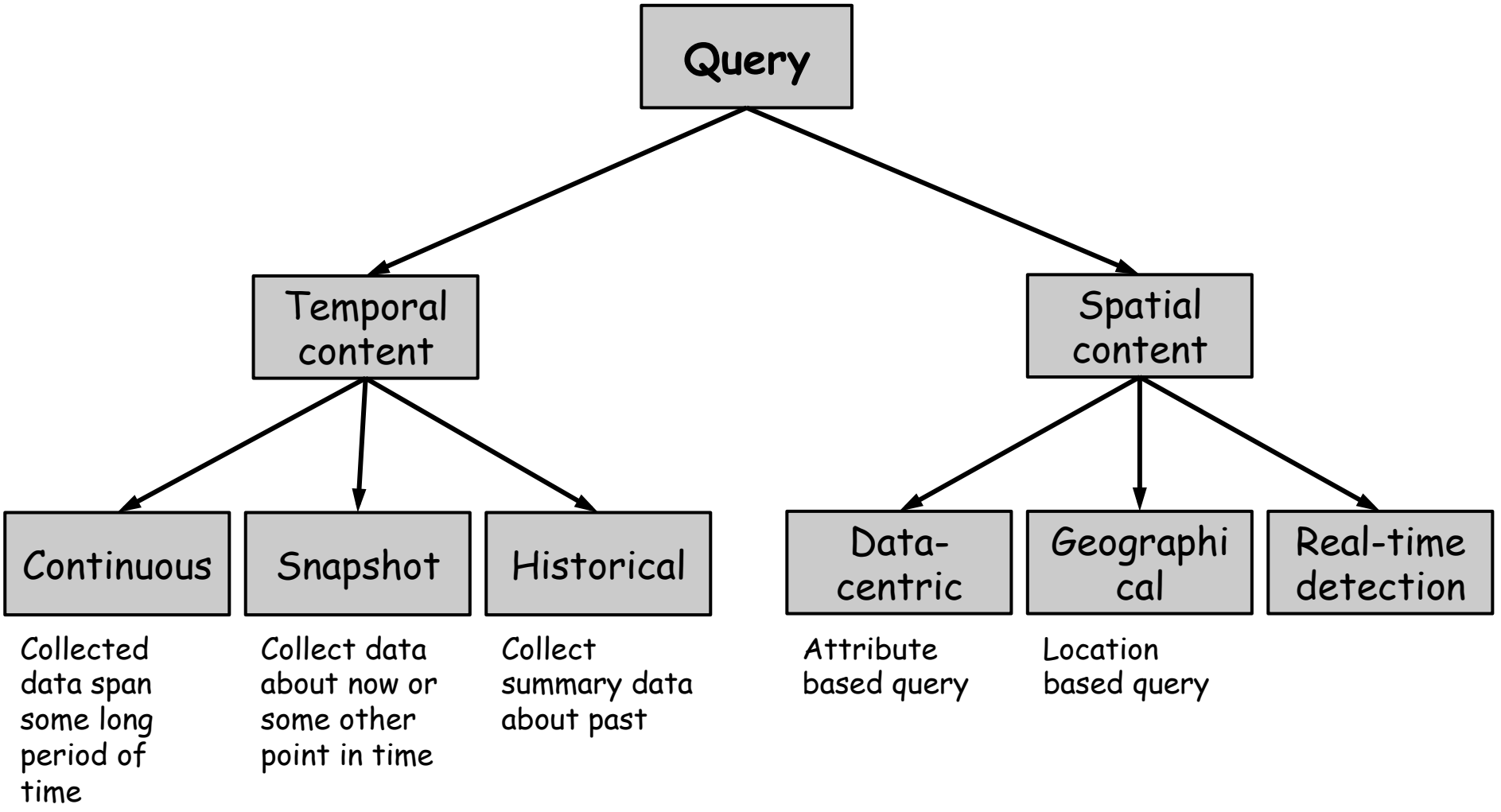
Main approaches in query processing

- Pull-based (interest dissemination)
  - Sink-initiated query dissemination
  - Users send their interest to a sensor node, a subset of the nodes or the entire network.
  - This interest may be about a certain attribute of the sensor field or a triggering event.

- Push-based (advertisement of available data)
  - Sensor-initiated information delivery
  - Sensor nodes advertise the available data to the users
  - The users query the data which they are interested in

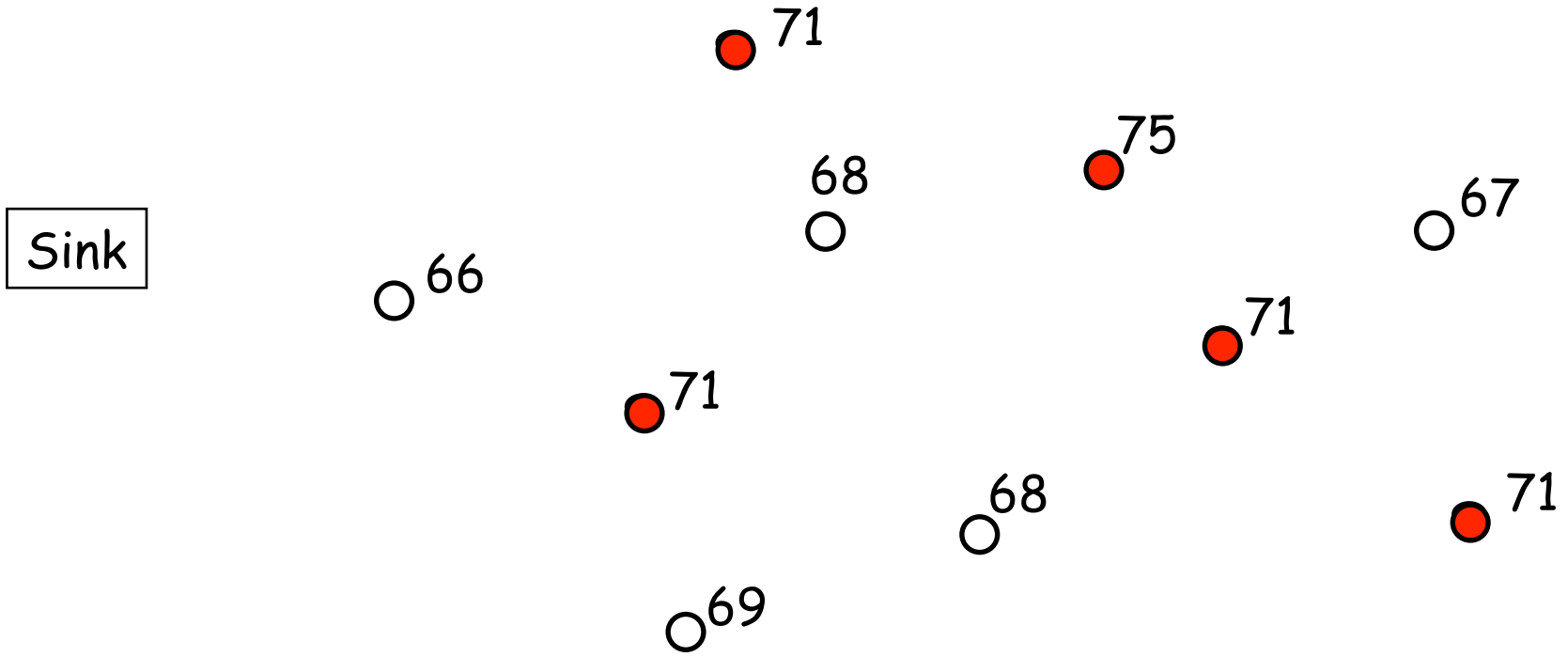
- Push-pull
  - Both sensors/sink are actively involved

Query classification



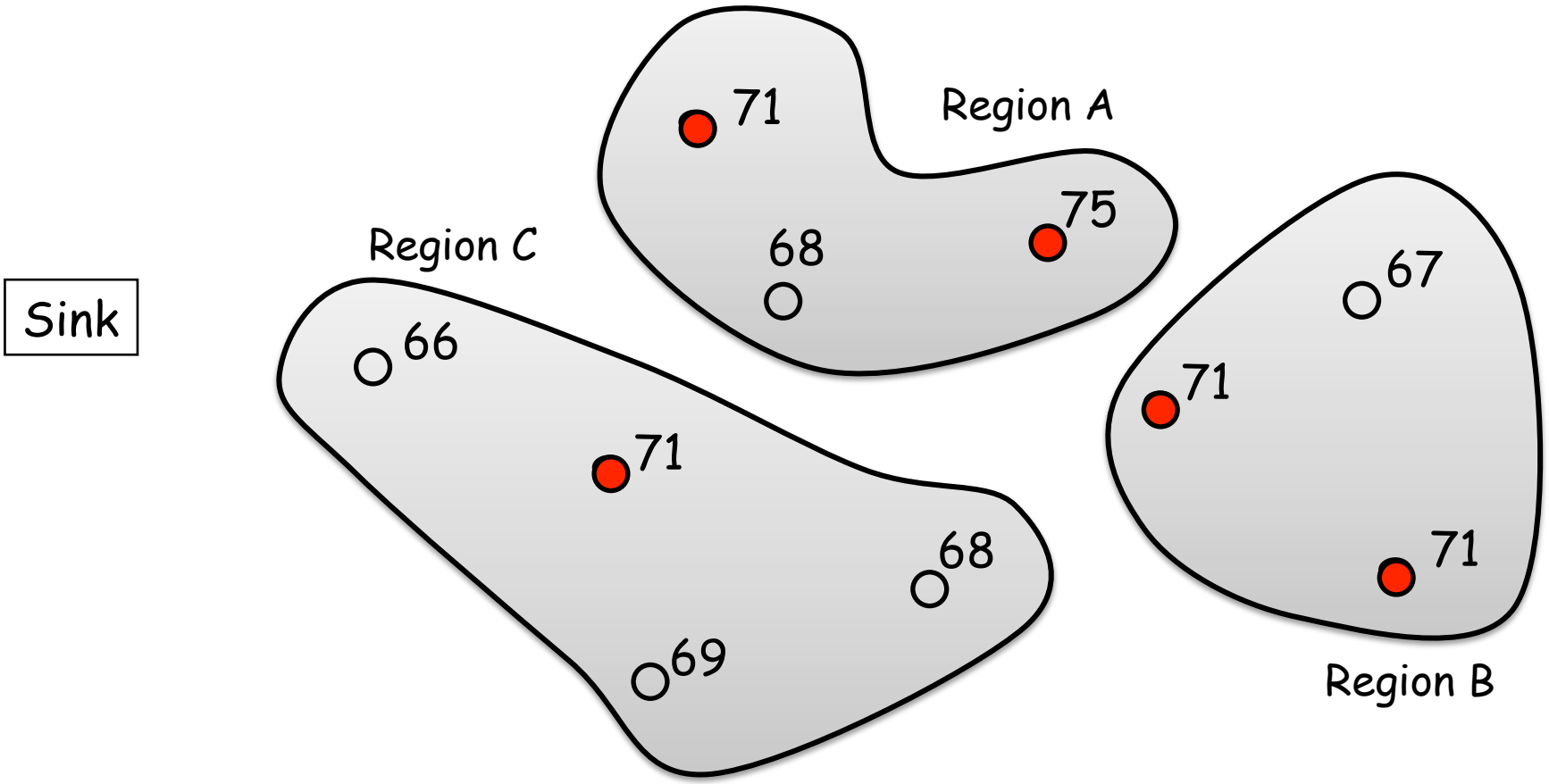
Example:
Data-centric query (attribute based query)

Query: Sensor nodes that read >70°F temperature



Example:
Geographical query (location based query)

Query: Region A nodes should send their temperatures



Query processing

Data aggregation

Data Aggregation/Fusion

- Impossible to continuously collect raw sensor data
  - Limited memory and bandwidth
  - Information overload
- Individual sensor readings are of limited use
  - Interest in collective information rather than individual sensor data
  - Exploit in-network processing
- Goal
  - Save energy and increase network lifetime by combining several sensor data

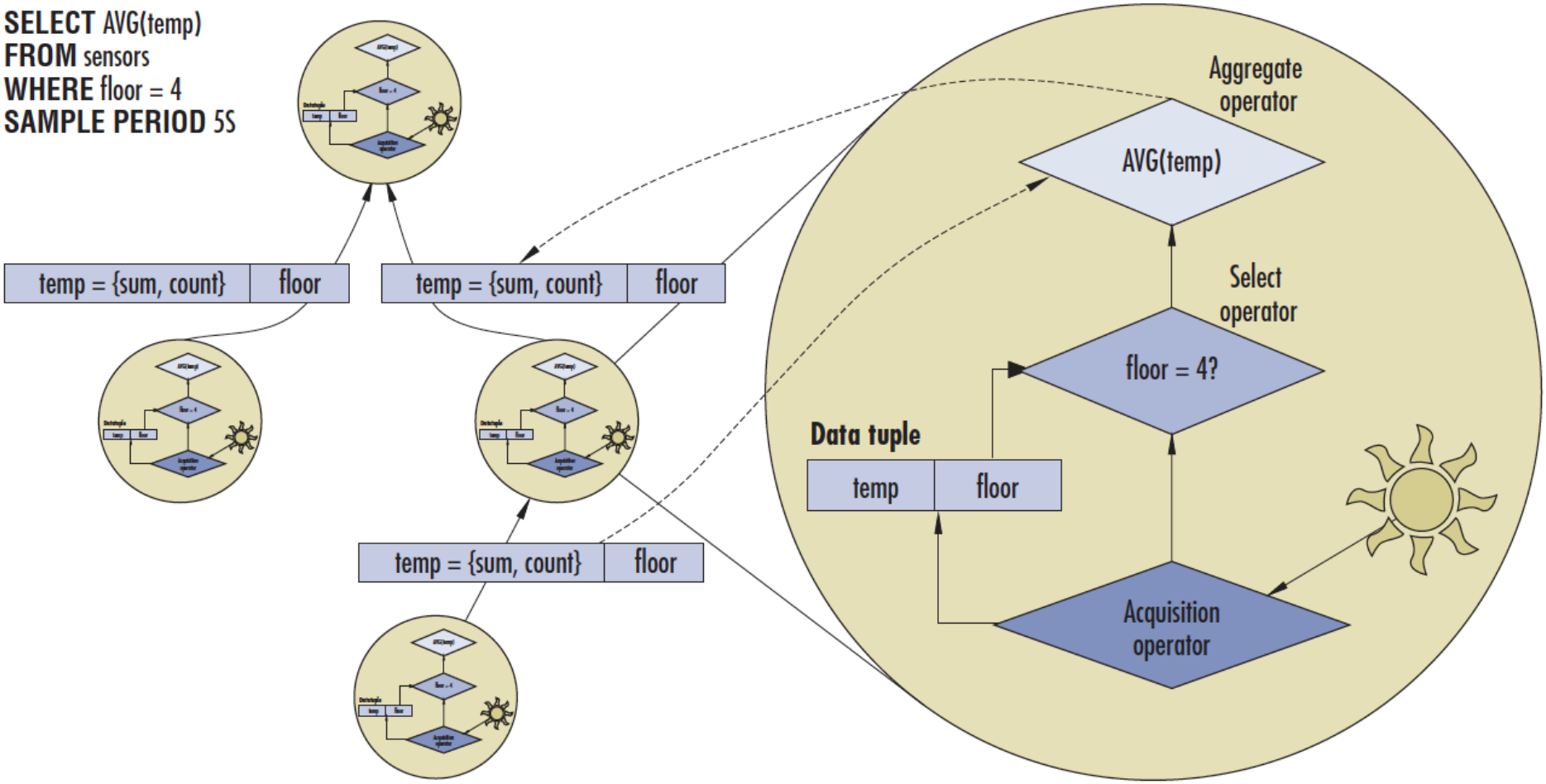
Data Aggregation/Fusion

- Data Aggregation
  - Process of combining data or information to estimate or predict events

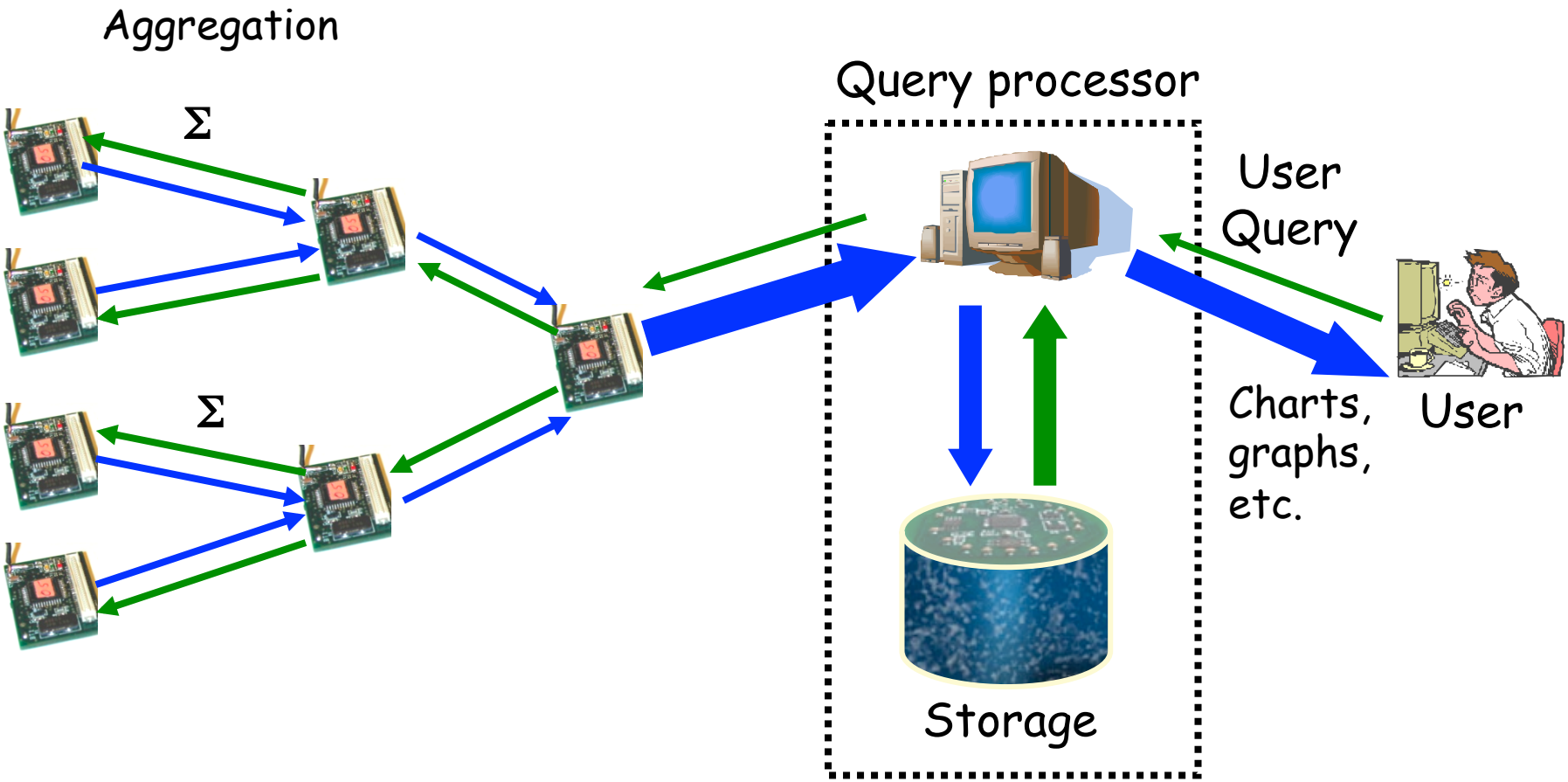
- Idea
  - Take advantage of the routing hierarchy and high network density

Data Aggregation/Fusion

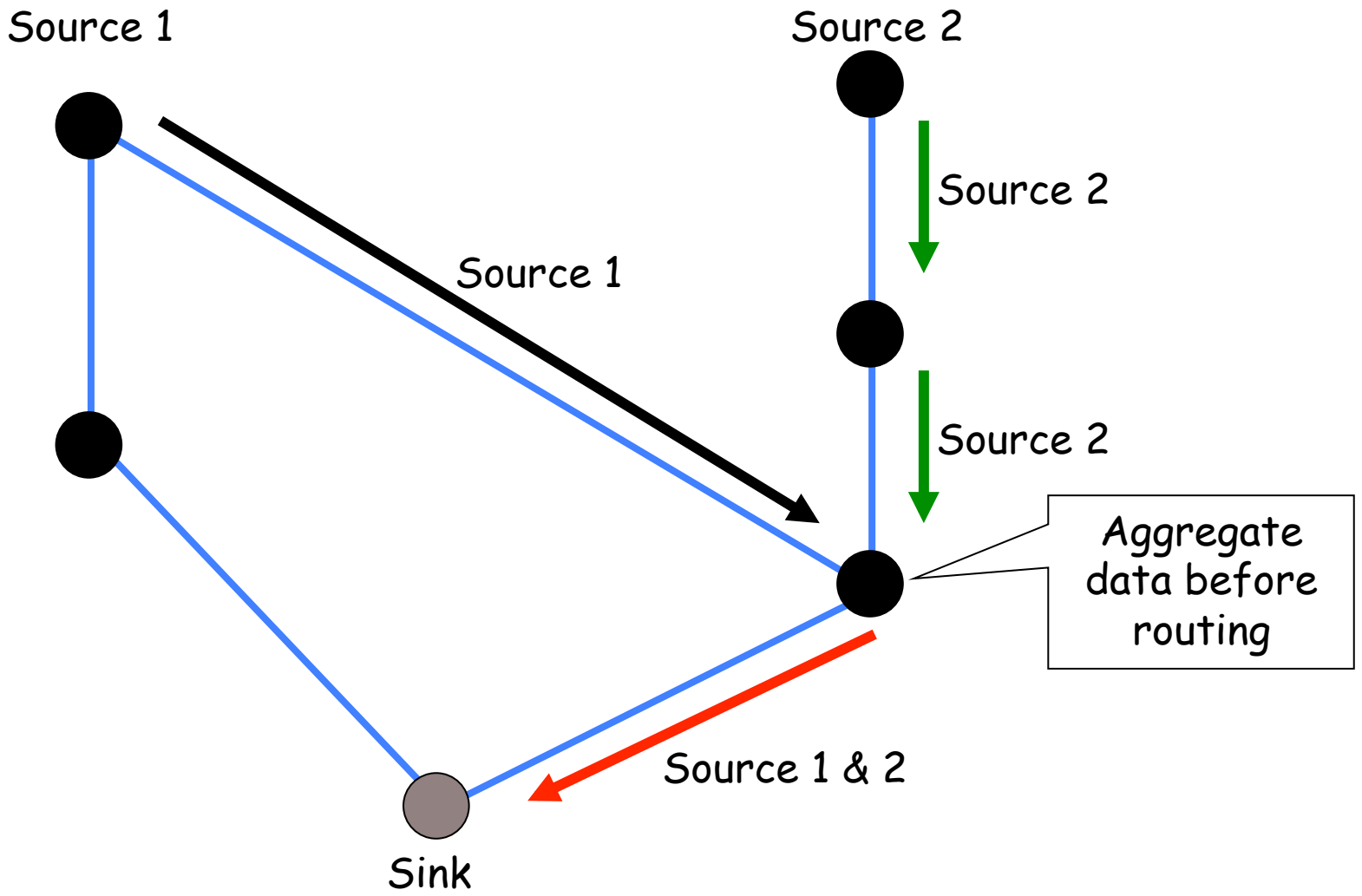
[Madden04]



Data Aggregation/Fusion

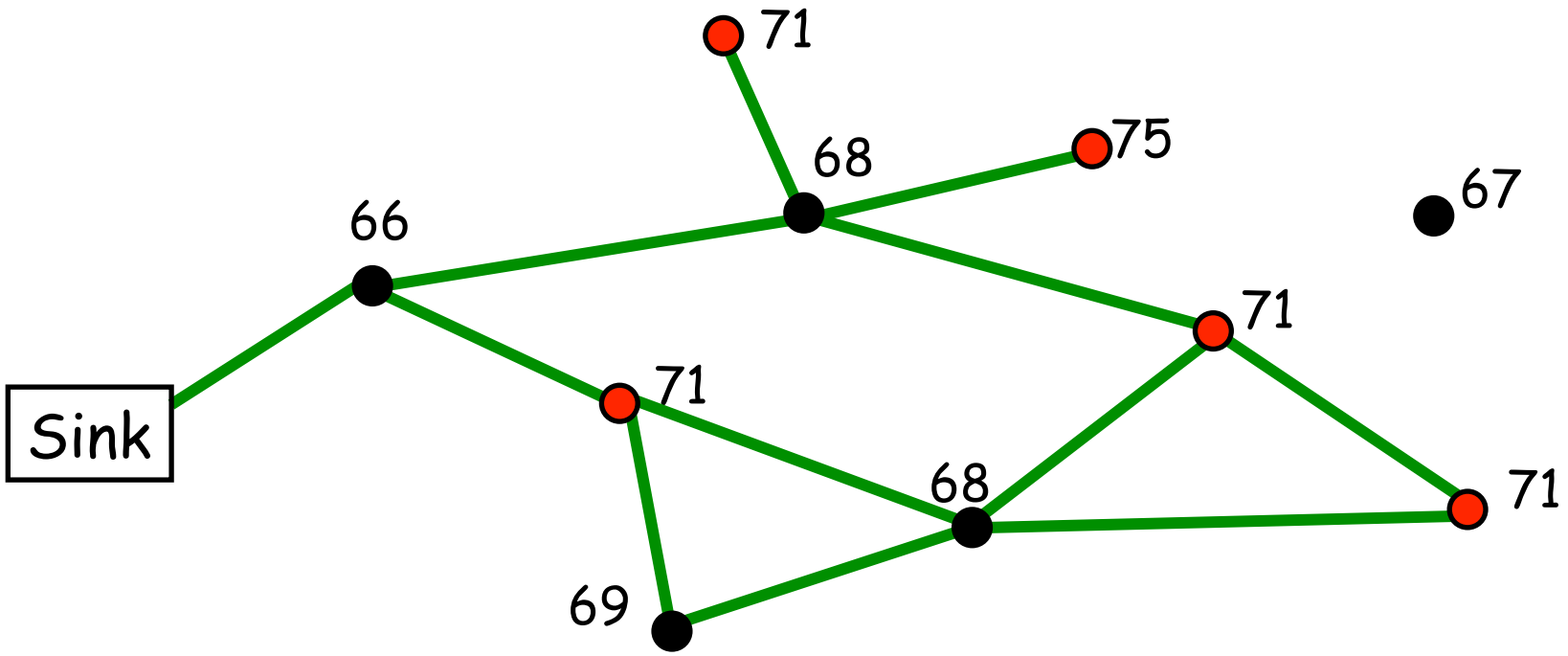


Data aggregation



Data aggregation/fusion

The sink asks the sensor nodes to report certain conditions.
Data coming from multiple sensor nodes are aggregated.

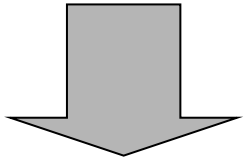


Query: Sensor nodes that read >70°F temperature

Data aggregation components

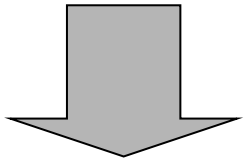
Data Storage

Store sensor data in a memory efficient way, while preserving the accuracy of the information.



Aggregation Functions

Place aggregation points on the paths from sensors to sink.



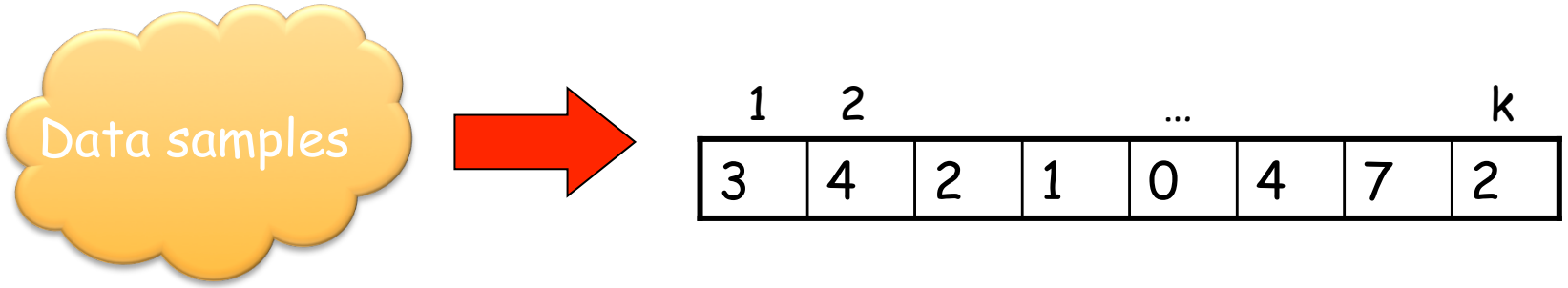
Aggregation Paths

Which are the optimal aggregation points?
Which is the most suitable path from source to sink to favor data aggregation?

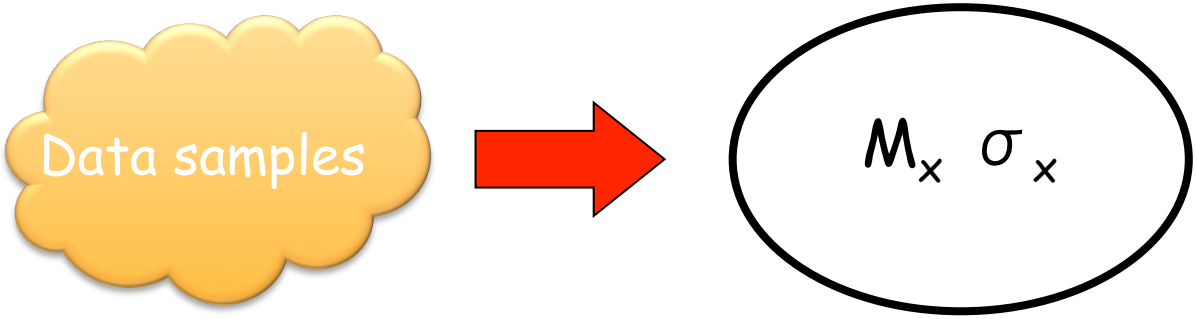
Data storage representations

Data can be represented with different degrees of accuracy.

Store individual data -> histograms and lists



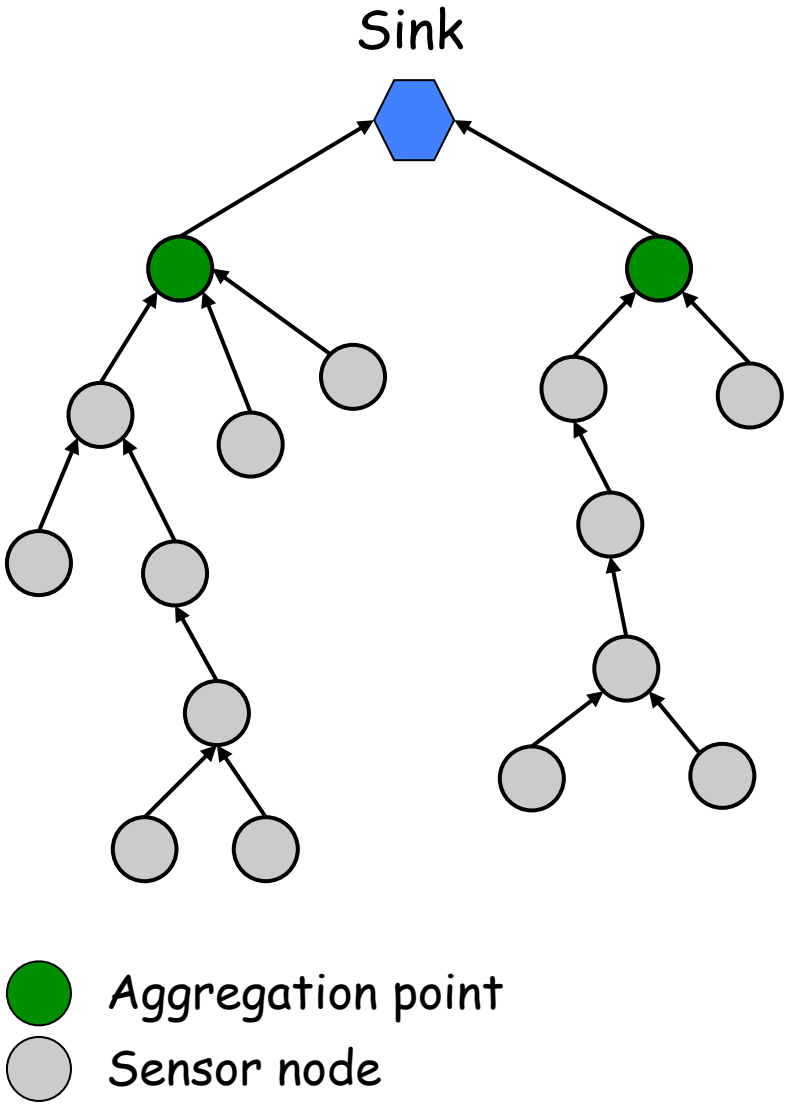
Store only mean and variance or other statistical representations



Aggregate functions

- Simple functions
  - Average, max, min, median
  - Suppression of duplicates
- More sophisticated functions
  - Exploit spatial and temporal correlation
  - Signal processing (convolution, filtering, etc.)
- What type of information should we expect from aggregation?
  - Streams
  - Robust estimates

Aggregate paths



- Challenges
  - Find the optimal number of aggregation points
  - Selection of aggregation points
  - Dynamic change of aggregation points
    - Energy efficiency

Aggregate challenges

- Properties of the environment
  - unreliable environment
    -> Never complete/up-to-date information about the neighborhood
  - certain information unavailable
    -> Information trickles in one message at a time
  - expensive to obtain

- Open questions
  - how many nodes are present?
  - how many nodes are supposed to respond?
  - what is the error distribution?
  - what about malicious nodes?

-> Trying to build an infrastructure to remove all uncertainty from the application may not be feasible

Optimal data aggregation

- Optimal data aggregation is NP-Hard

- Sub-optimal algorithms
  - Opportunistic
    - Just aggregate when possible
  - Center at nearest source (CNS)
    - The nearest source acts as the aggregation point
  - Shortest path tree (SPT)
    - Sources send using shortest path, if able aggregate
  - Greedy Incremental Tree (GIT)
    - Recursively select the closest source to the tree
  - Clustered Diffusion with Dynamic Data Aggregation
    - Hybrid between diffusion and clustering with the ability to aggregate data at the cluster heads

Advantages and disadvantages of aggregation

- Advantages
  - Reduce traffic load by aggregating data en route
    - Energy and memory efficiency
  - Scalable to large numbers of both sinks and sensors

- Disadvantages
  - Requires careful design to tradeoff accuracy and storage and message size
  - Incurs information loss, making robust estimation more difficult
    - e.g. a single outlier reading can damage MAX/MIN aggregates

Query challenges

- High density: Rich and massive data
- Correlation: Spatially distributed and correlated data
- Uncertainty: Noise, erroneous data, outliers
- Semantics: Individual data ≠ useful information

Querying Sensor Networks

- TinyDB
  - Information storage framework
  - Tree-based data collection
- COUGAR
  - Individual sensor data
  - Distributed Gathering
  - SQL-like
- TAG (Tiny Aggregation)
  - Focus on Aggregation using SQL-like query language
  - Integrated in TinyOS

Query processing

TinyDB

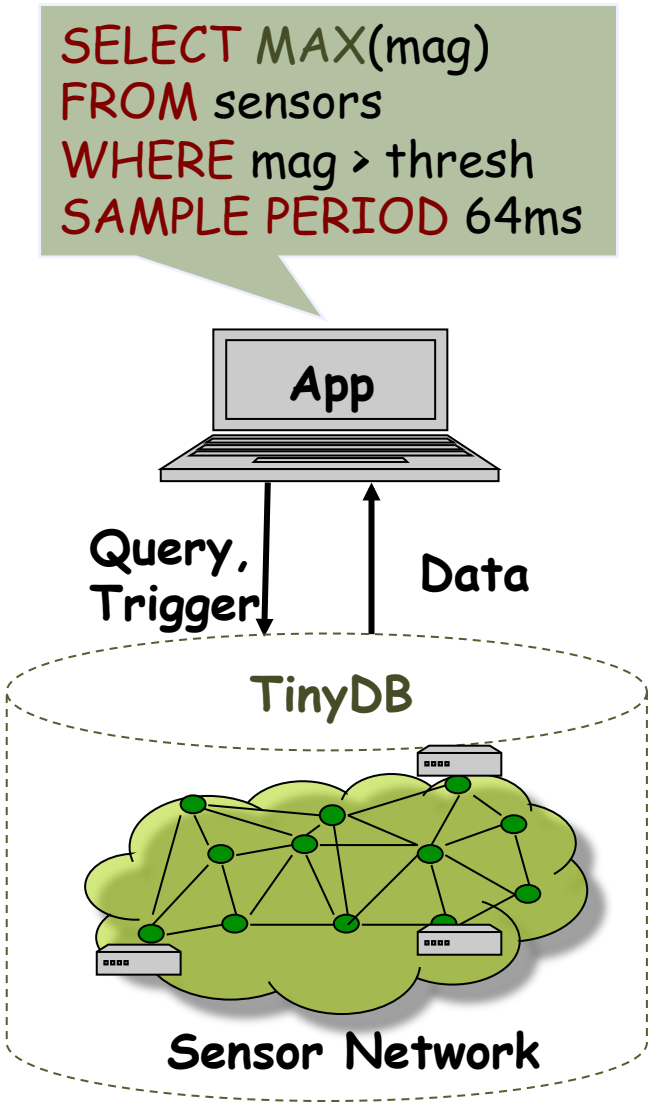
TinyDB

- A distributed query processor for networks of Micas
  -> Sensor network = Distributed database
- Goal: Eliminate the need to write C code for most users

- Features
  - Declarative queries
  - Temporal + spatial operations
    - Data stored locally
  - Multi-hop routing -> tree-based routing
  - In-network storage
    - Top-down SQL query
    - Results aggregated back to the sink

TinyDB

- High level abstraction
  - Data centric programming
  - Interact with sensor network as a whole
  - Extensible framework
- Under the hood
  - Intelligent query processing
    - query optimization
    - power efficient execution
  - Fault mitigation
    - automatically introduce redundancy
    - avoid problem areas



Feature Overview

- Declarative SQL-like query interface
- Metadata catalog management
- Multiple concurrent queries
- Network monitoring (via queries)
- In-network, distributed query processing
- Extensible framework for attributes, commands, and aggregates
- In-network, persistent storage

Architecture

TinyDB GUI

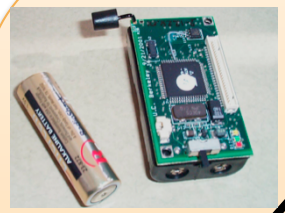
TinyDB Client API

JDBC

DBMS

PC side

Mote side



0

1

2

8

4

5

6

7

TinyDB query processor

Sensor network

Data Model

- Entire sensor network as one single, infinitely-long logical table: sensors
- Columns consist of all the attributes defined in the network
- Typical attributes
  - Sensor readings
  - Meta-data: node id, location, etc.
  - Internal states: routing tree parent, timestamp, queue length, etc.
- Nodes return NULL for unknown attributes
- On server, all attributes are defined in catalog.xml
- Discussion: other alternative data models?

Query Language (TinySQL)

SELECT <aggregates>, <attributes>

FROM {sensors | <buffer>}
WHERE <predicates>
GROUP BY <exprs>
SAMPLE PERIOD <const> | ONCE
INTO <buffer>
TRIGGER ACTION <command>

Comparison with SQL

- Single table in FROM clause
- Only conjunctive comparison predicates in WHERE and HAVING
- No subqueries
- No column alias in SELECT clause
- Arithmetic expressions limited to column op constant
- Only fundamental difference: SAMPLE PERIOD clause

TinySQL Examples

"Find the sensors in bright nests."

1

```
SELECT nodeid, nestNo, light
FROM sensors
WHERE light > 400
EPOCH DURATION 1s
```

Sensors

Epoch	Nodeid	nestNo	Light
0	1	17	455
0	2	25	389
1	1	17	422
1	2	25	405

TinySQL Examples

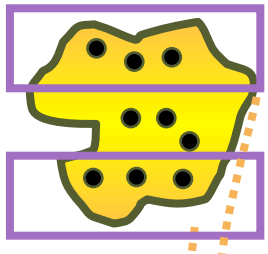
2

```
SELECT AVG(sound)
FROM sensors
EPOCH DURATION 10s
```

"Count the number occupied nests in each loud region of the island."

3

```
SELECT region, CNT(occupied), AVG(sound)
FROM sensors
GROUP BY region
HAVING AVG(sound) > 200
EPOCH DURATION 10s
```



Epoch	region	CNT(…)	AVG(…)
0	North	3	360
0	South	3	520
1	North	3	370
1	South	3	520

Regions w/ AVG(sound) > 200

Event-based Queries

- ON event SELECT …
- Run query only when interesting events happens
- Event examples
  - Button pushed
  - Message arrival
  - Bird enters nest
- Analogous to triggers but events are user-defined

Query over Stored Data

- Named buffers in Flash memory
- Store query results in buffers
- Query over named buffers
- Analogous to materialized views

- Example:
  - CREATE BUFFER name SIZE x (field1 type1, field2 type2, …)
  - SELECT a1, a2 FROM sensors SAMPLE PERIOD d INTO name
  - SELECT field1, field2, … FROM name SAMPLE PERIOD d

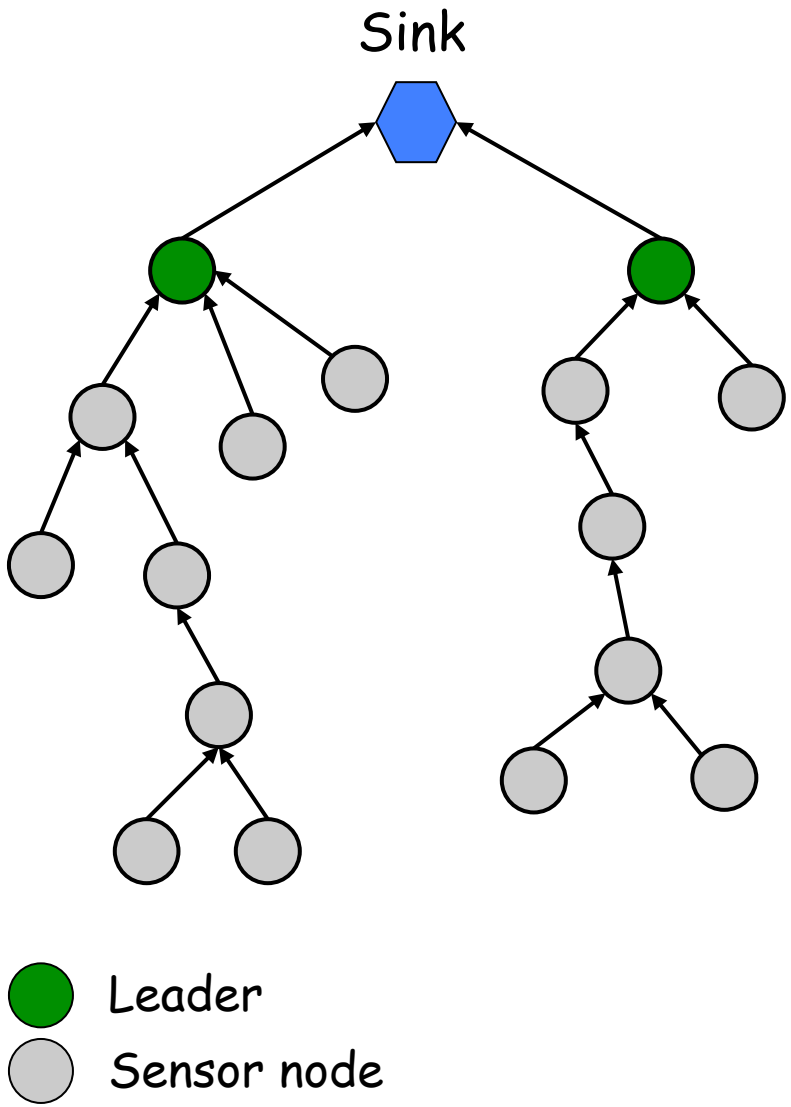
Query processing

COUGAR

COUGAR

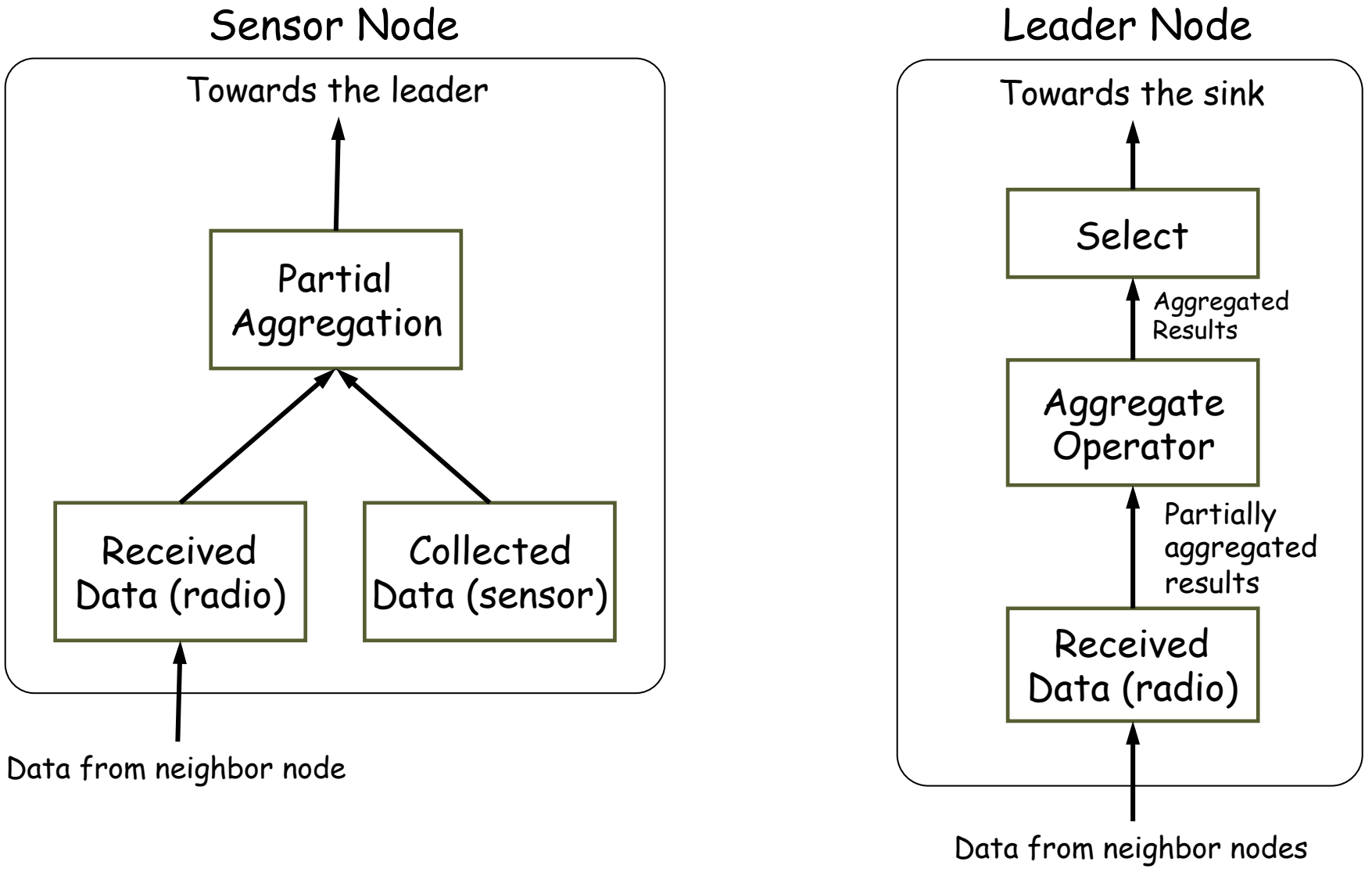
- Performs in-network computation
- Decreases energy consumption
- Attempts to merge similar queries
- Propagates results to sink

COUGAR Architecture



- Sensors send their information towards designated leaders
- Partial aggregation is performed at sensors
- Collected data aggregated optimally at the leaders

COUGAR Components



COUGAR Leader Selection

- Methods
  - Fixed
  - Randomly
- Leader selection policy
  - Dynamically maintained in case of failure
  - Minimize communication distance

[Madden02]

Query processing

Tiny Aggregation (TAG)

Tiny Aggregation (TAG)

- Aggregation service for TinyOS sensor motes
- SQL-like declarative language
- Uses simple and declarative interfaces
- Sensitive to resource constraints and link failures
- Reduces costs depending on type of aggregates

TAG Approach

- TAG approach consists of two phases

1. Query Distribution: Aggregate queries are pushed down the network to construct a spanning tree from the sink
   - Root broadcasts the query, each node hearing the query broadcasts.
   - Each node selects a parent. The routing structure is a spanning tree rooted at the query node.

2. Data Collection: Aggregate values are routed up the tree.
   - Internal node aggregates the partial data received from its sub-tree.

TAG: Query representation

- TAG queries have the form:

```
SELECT {agg(expr), attrs}
FROM sensors
WHERE {selPreds}
GROUP BY {attrs}
HAVING {havingPreds}
EPOCH DURATION i
```

TAG: Classification of aggregates

- Classification keys
  - **Duplicate** insensitive: aggregates are unaffected by duplicate readings from a single device while duplicate sensitive aggregates will change when a duplicate reading is reported.
  - **Exemplary:** aggregates return one or more representative values from the set of all values; **summary** aggregates compute some property over all values.
  - **Monotonic**: Given two partial states s1, s2, evaluator function e, and merging function f
    Resulting state record s' = f(s1, s2)

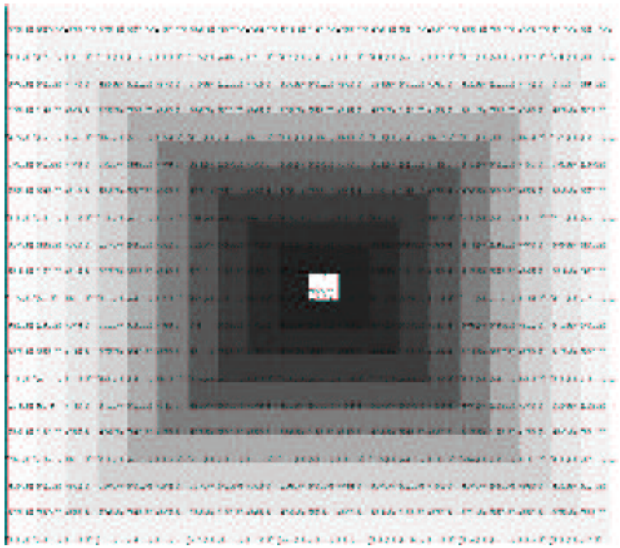
    → $e(s') \geq \max(e(s1), e(s2)) \wedge e(s') \leq \min(e(s1), e(s2))$
  - **Partial**: amount of state that is required

TAG: Classification of aggregates

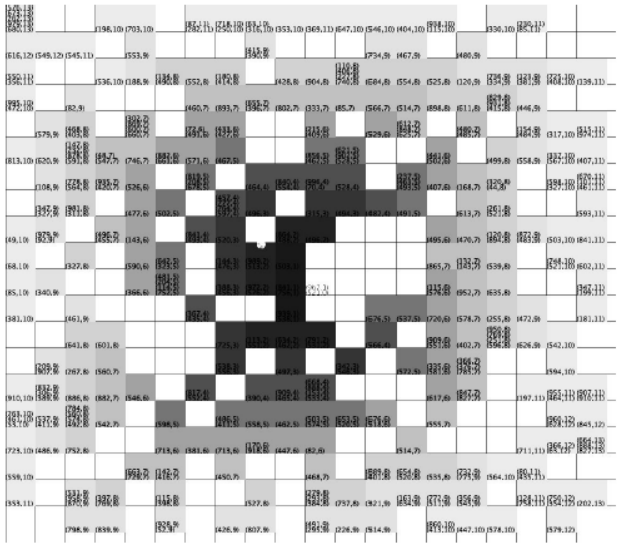
- Classes of aggregates

	MAX,MIN	COUNT, SUM	AVERAGE	MEDIAN	COUNT DISTINCT	HISTOGRAM
Duplicate sensitive	No	Yes	Yes	Yes	No	Yes
Exemplary (E), Summary (S)	E	S	S	E	S	S
Monotonic	Yes	Yes	No	No	Yes	No
Partial state	Distributive	Distributive	Algebraic	Holistic	Unique	Content-Sensitive

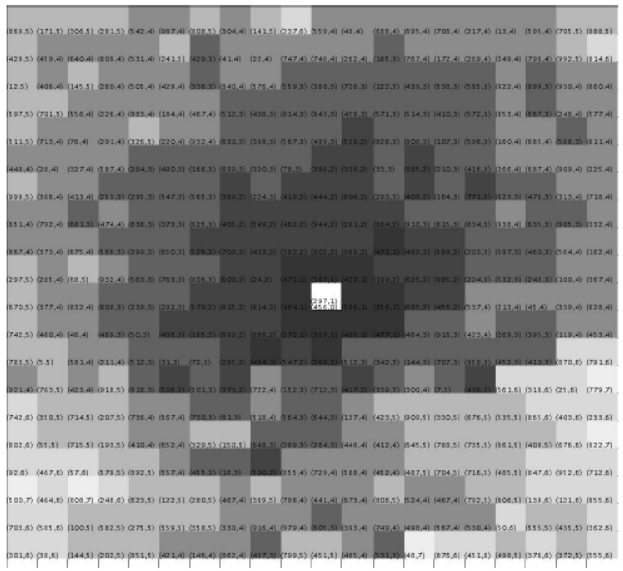
TAG: Results



Simple



Random



Realistic

TAG: Results

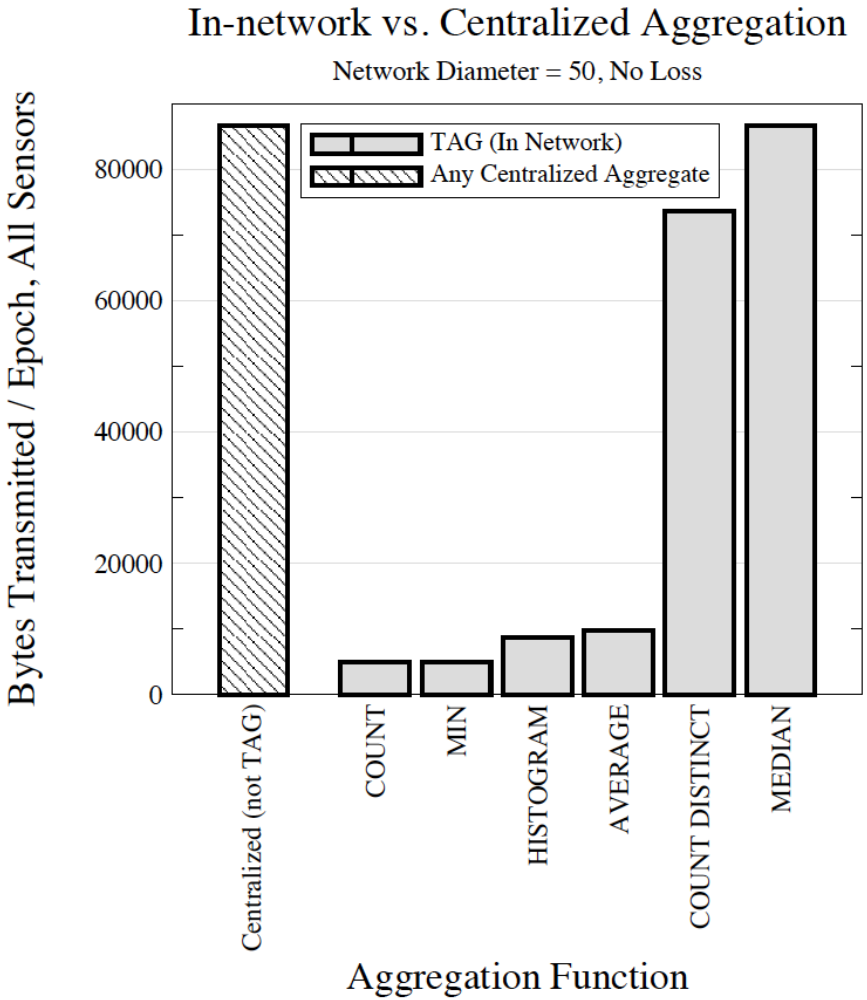
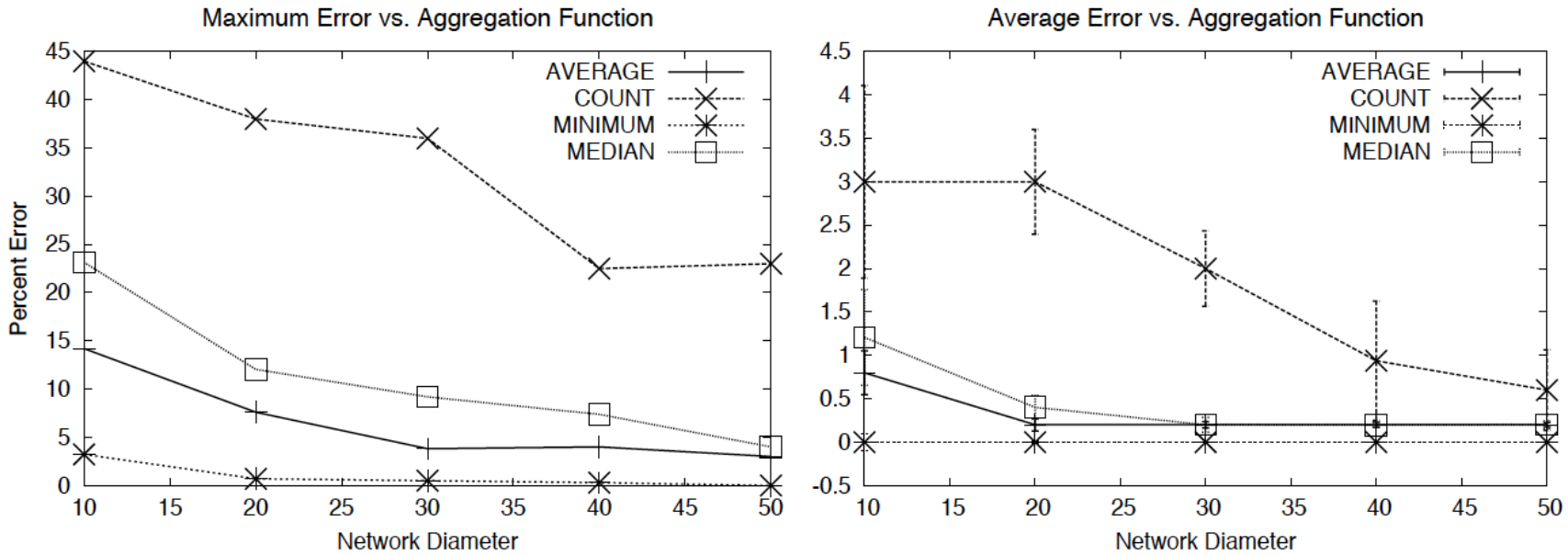


Figure 4: *In network Vs. Centralized Aggregates*

TAG: Results



(a) Maximum Error (b) Average Error

Figure 6: *Effect of a Single Loss on Various Aggregate Functions*. Computed over a total of 100 runs at each point. Error bars indicate standard error of the mean, 95% confidence intervals.

TAG: Results

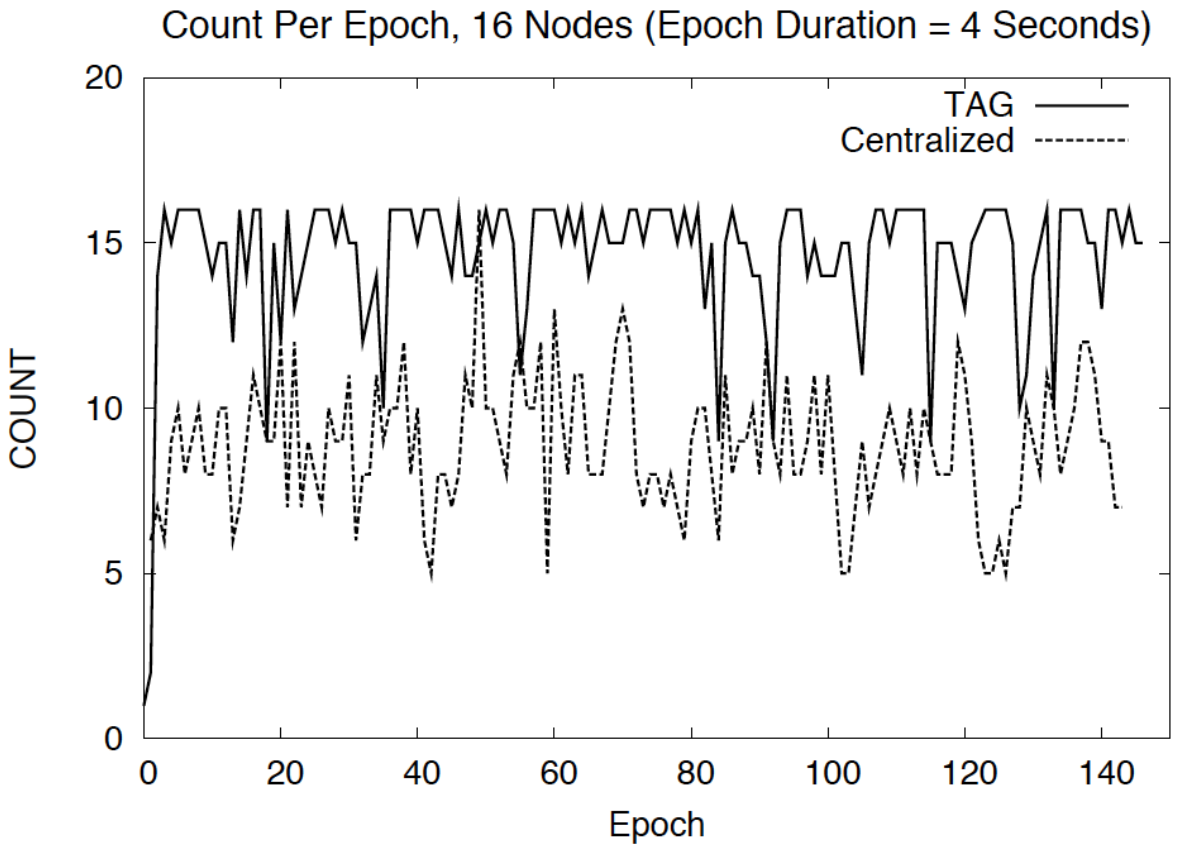


Figure 8: *Comparison of Centralized and TAG based Aggregation Approaches in Lossy, Prototype Environment Computing a COUNT over a 16 node network.*

Summary

Summary

- Application protocols very specific
- CoAP as flexible application protocol
- Query processing systems for simple "programming"

Literature

Literature

[Shelby] Shelby, et al., "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-13, Expires June 9, 2013

[RFC6690] RFC 6690, "Constrained RESTful Environments (CoRE) Link Format", (draft-ietf-core-link-format), 2012-08, RFC 6690 (Proposed Standard)

[Shen] C-C Shen, et.al., "Sensor Information Networking Architecture and Applications", IEEE Personal Communications Magazine, pp. 52-59, August 2001.

[Madden05] S. R. Madden, et. al., "TinyDB: An Acquisitional Query Processing System for Sensor Networks",' ACM Transactions on Database Systems, pp. 122–173, March 2005.

[Madden04] S. Madden and J. Gehrke, "Query Processing in Sensor Networks", IEEE Pervasive Computing, vol. 3, No. 1., March 2004.

[Madden02] S. Madden et al., "TAG: A Tiny Aggregation Service for Ad-Hoc Sensor Networks", in Proc. of OSDI, 2002

[Yao] Yong Yao, Johannes Gehrke, "The Cougar Approach to In-Network Query Processing in Sensor Networks", Sigmod Record, Volume 31, Number 3. September 2002

[Gubbi] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, Marimuthu Palaniswami, "Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions", Technical Report CLOUDS-TR-2012-2, Cloud Computing and Distributed Systems Laboratory, The University of Melbourne, June 29, 2012