Scalable Context Data Distribution in Mobile Environments

Ph.D. Student:
Mario Fanelli

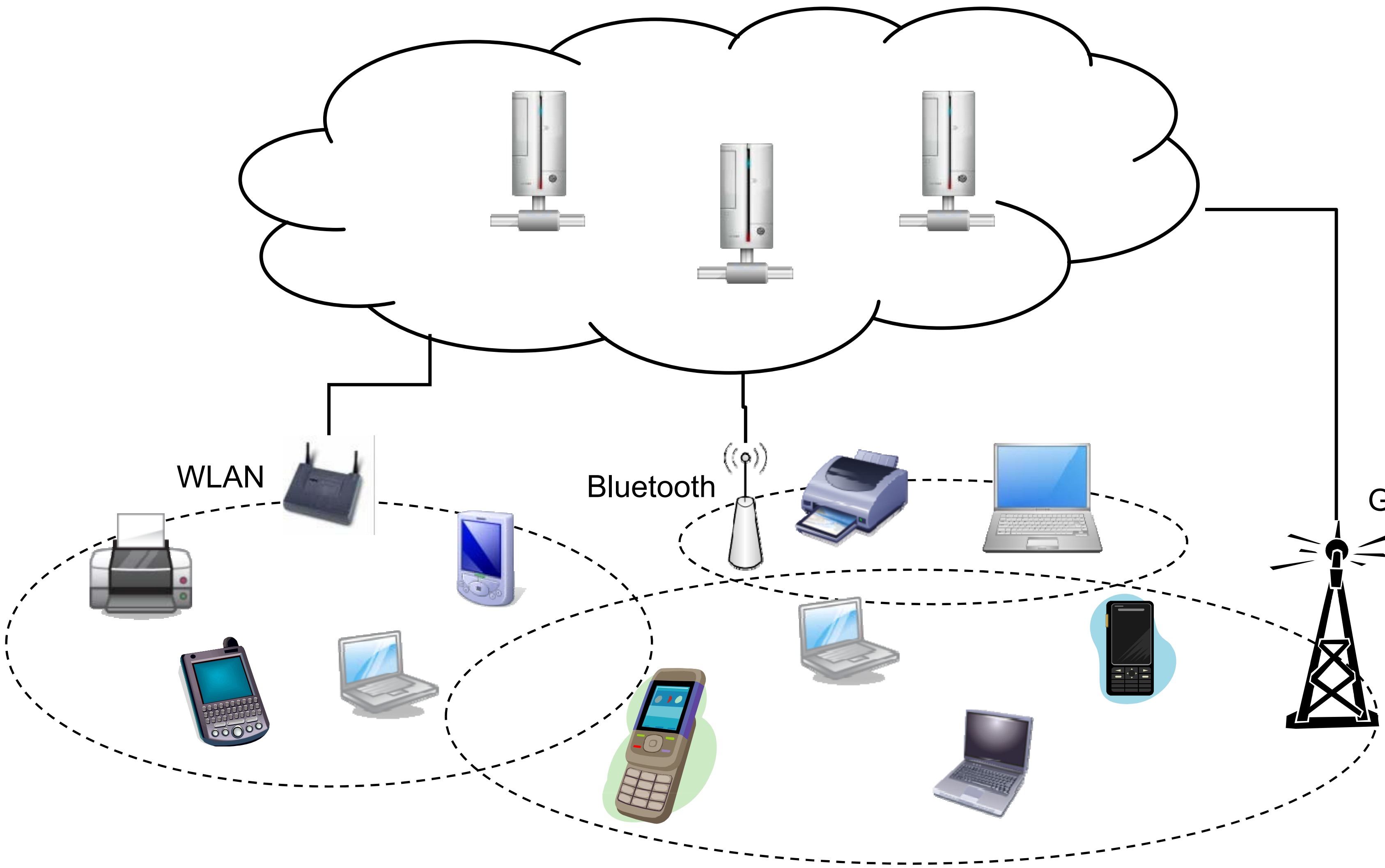
DEIS, University of Bologna, ITALY
mario.fanelli@unibo.it

July 20th

Agenda

1. Principal scenario: Context-Aware applications for mobile environments
2. System model
    i. Context definition and classification
    ii. Generic context handling infrastructure
    iii. Middleware-based system model
3. Context Data Distribution Service
    i. Context Data Distribution Service Logical Architecture
    ii. Context Data Filtering Layer
    iii. Context Data Dissemination Layer
    iv. Dissemination Infrastructure Layer
    v. Adaptation support
4. Enable system scalability
    i. Context Data Level
    ii. Communications Level
5. Current solution: Scalable context-Aware middleware for mobiLe EnviromentS
6. Conclusions and ongoing work

Application scenario: Context-Aware applications for mobile environments



Context definition

- Broadly speaking, a context-aware application is an application able to adapt itself according to the **current execution context**
- The **current execution context** is a very vague concept. It is strictly related with the aspects the designer considers useful to application behavior adaptation
- In literature, many different definitions exist:
  1. the **current execution context** contains where you are, who you are with and what resources are nearby
  2. the **current execution context** contains any information that can be used to characterize the situation of an entity
  3. …

Leaving out the definition, to support our principal scenario we need appropriate support infrastructures able to distribute context data to mobile nodes

Context classification

User devices form a **mobile system.** Consequently, each device has **two different types** of context:

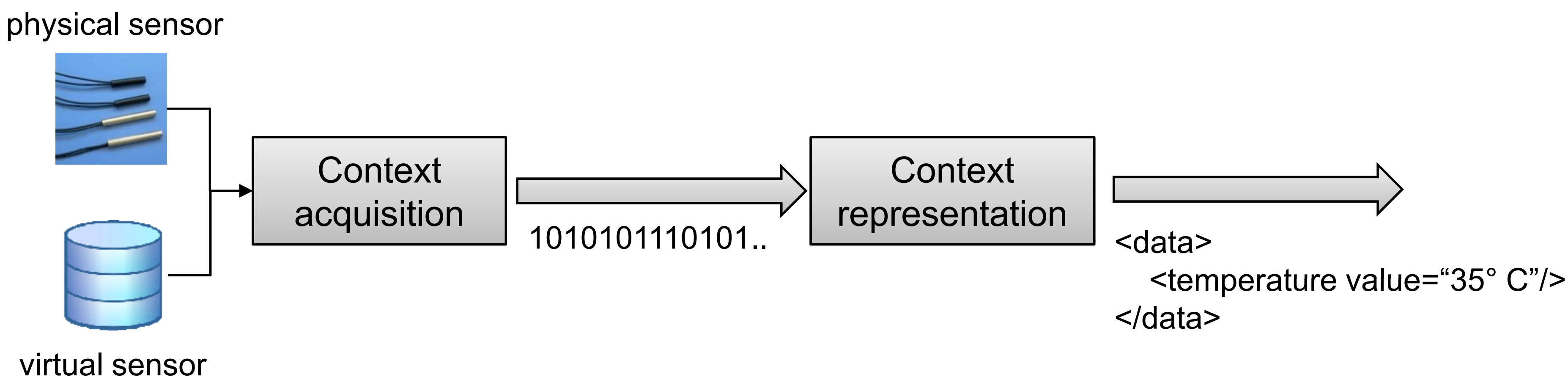
1. **Individual context –** Individual context contains context aspects descending from node egocentric view
2. **Social context –** Social context contains context aspects descending from the awareness of being an actor in the entire system

Besides, we can classify context aspects in:

1. **System level aspects –** Low-level/technical attributes used to trigger/adapt management functions
2. **Application level aspects –** High-level attributes usually directly associated with application-level goals

	Individual context	Social context
System level aspects	available bandwidth, CPU load	CPU loads shared between near nodes to address task migrations
Application level aspects	local user profile, place profile	user profiles of nearby people

Generic context handling infrastructure

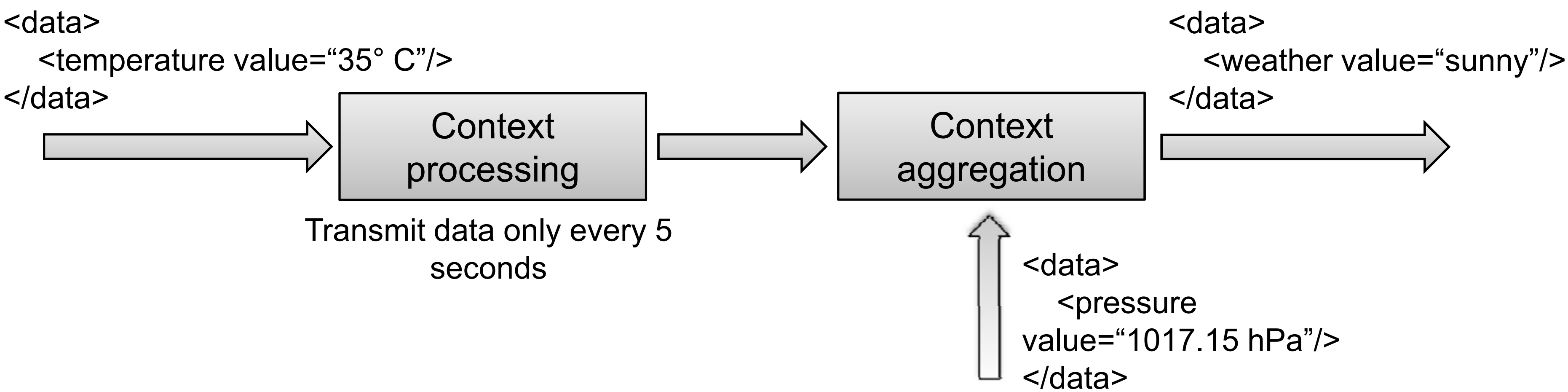


- A context-aware system has to support context data during their entire lifetime
- Data lifetime is divided in four phases:

1. **Context acquisition –** Context data are retrieved by:
   - **Physical sensors:** temperature sensor, pressure sensor, etc.
   - **Virtual sensors:** place/user profile stored in a database, etc.

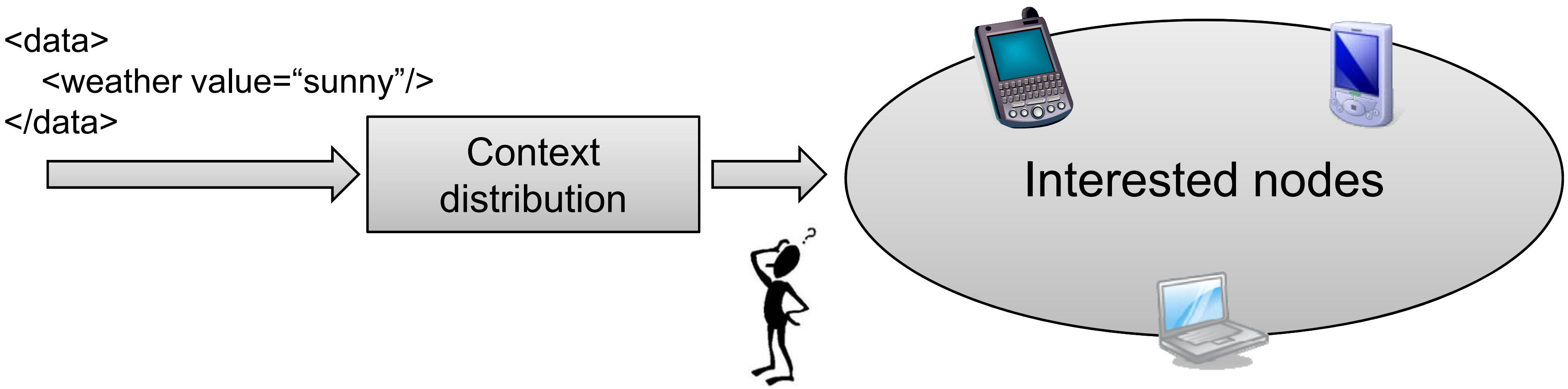
2. **Context representation –** Context data are represented using an high-level data representation technique (key-value pairs, first-order logic, ontology, etc.)

Generic context handling infrastructure



3. **Context data elaboration –** Context data can be subject of:
   - **Processing techniques:** These techniques aim to reduce system overhead by performing local elaborations. For instance, traditional processing techniques tailor data production rate by imposing timer-/threshold-based filters
   - **Aggregation techniques:** These techniques aim to derive high-level knowledge by melting together different data. For instance, the current weather status can be obtained combining data associated with temperature, pressure, etc.

Generic context handling infrastructure

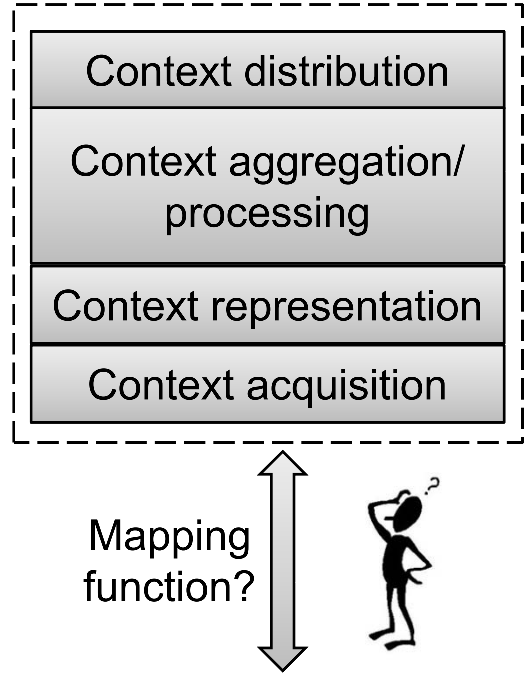


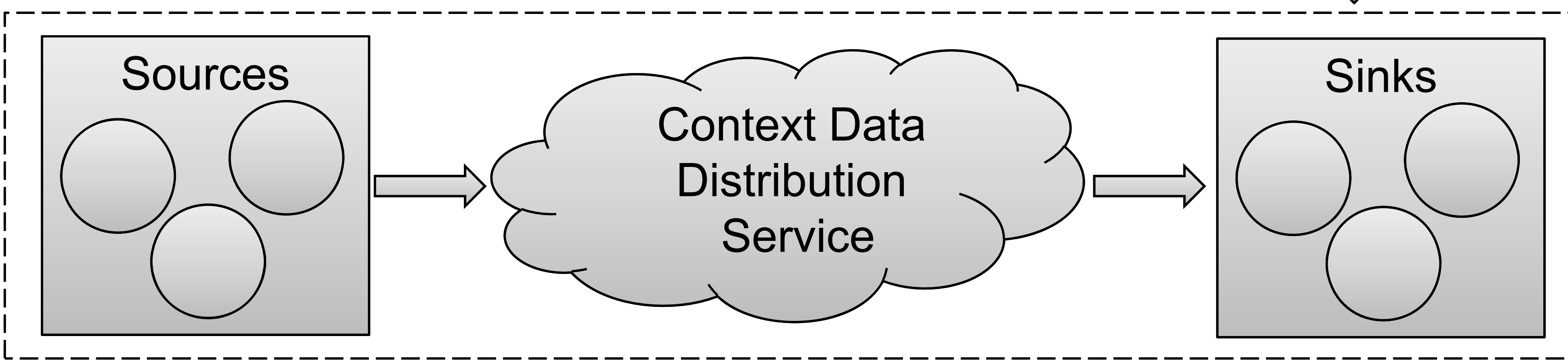
4. **Context distribution and retrieval –** Mobile nodes access needed context data. There are two different principal solutions:
   - **Sensors direct-access:** Each node accesses directly sensors deployed in the physical proximity
   - **Middleware-based access:** The middleware hides sensors by supplying appropriate access APIs. Nowadays, this is the standard-de-facto approach for the realization of context-aware infrastructures

Middleware-based system model

Focusing on the middleware-based model, context data distribution process involves two principal entity types and one principal service:

- **Source –** Each source is a producer of new context data
- **Sink –** Each sink is a consumer of context data. It expresses proper context data needs to guide data routing inside the system
- **Context Data Distribution Service –** The context data distribution service connects sinks and sources to enable the real data flow inside the system



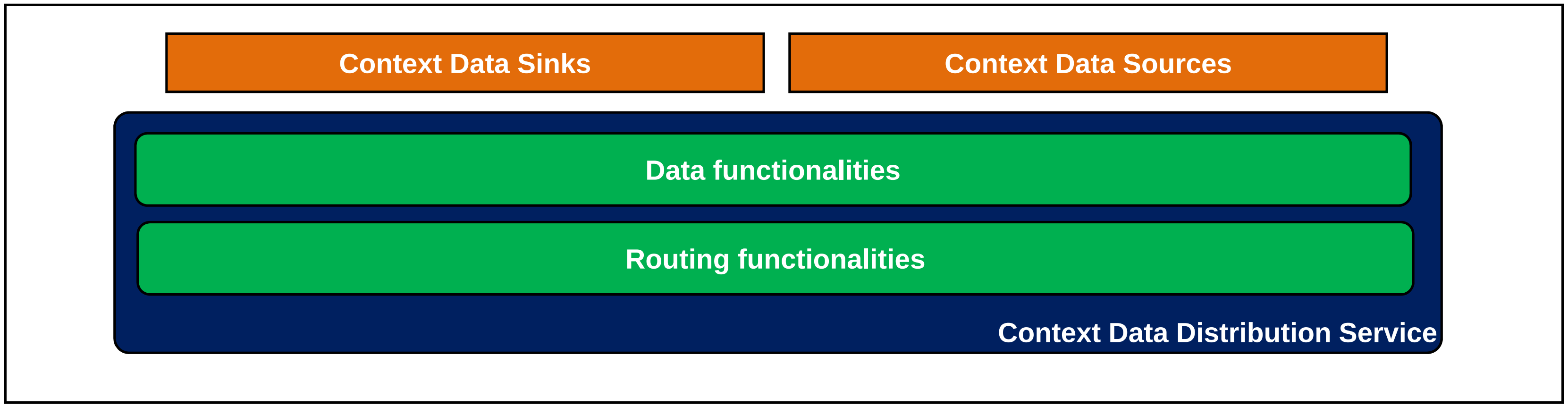


Middleware-based system model

	Source	Sink	Context Data Distribution Service
Context acquisition	√	x	x
Context representation	√	x	√
Context processing	√	x	√
Context aggregation	√	x	√
Context distribution	x	x	√

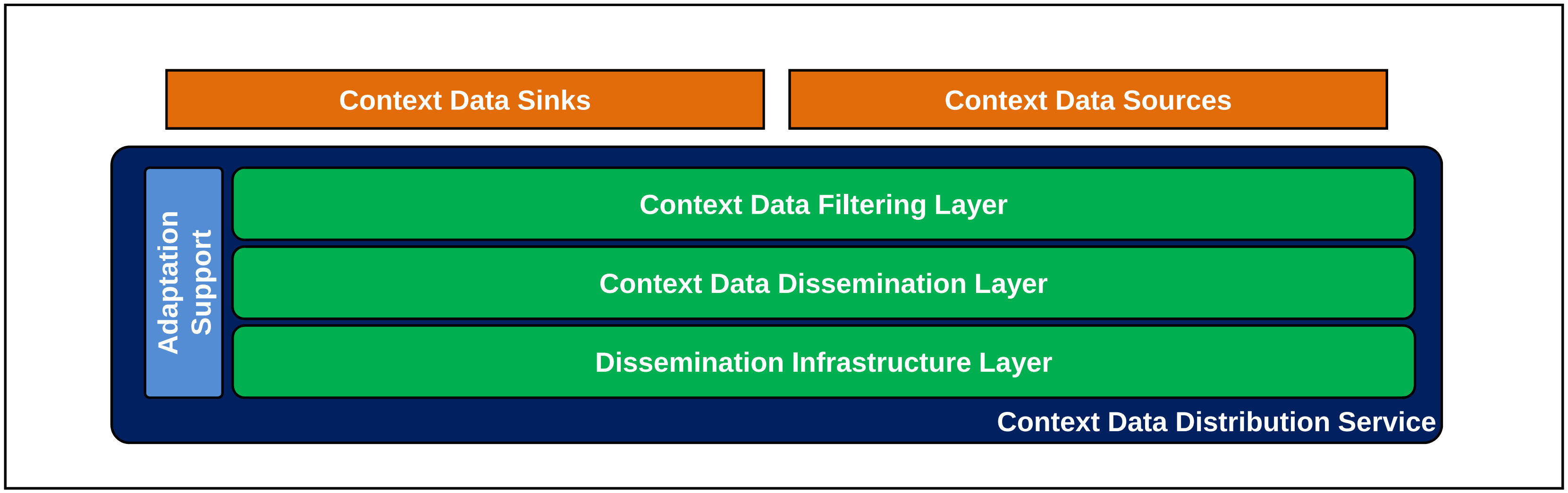
- **Sinks** can be also **sources** for new/aggregated data
- Both sources and context data distribution service can perform their own **context representation, processing** and **aggregation**
- Only the **context data distribution service** performs data distribution. **Sources** are completely agnostic of **sinks**

Context Data Distribution Service



- The **Context Data Distribution Service** has **routing functionalities.** In fact, it implements the **data distribution process**, i.e., the mechanism that enables the distribution of data by bridging together subscriptions and produced data
- The **Context Data Distribution Service** has also **data functionalities**:
  1. It can supply either **imperfect** or **old data** when nothing better is available
  2. It should offer a **data aggregation/processing framework**
  3. It should offer a **data history** functionality when necessary
  4. It should deal with **data security** problem by offering mechanisms to protect data integrity, availability and privacy

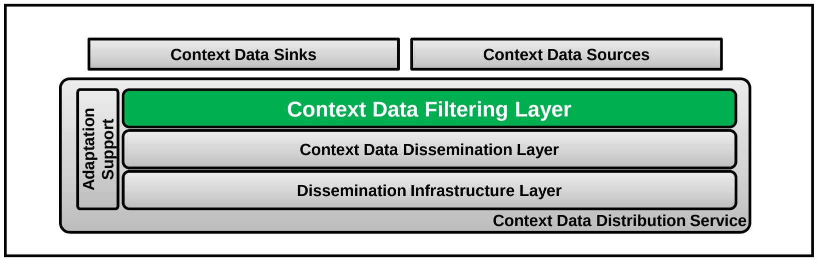
Context Data Distribution Service Logical Architecture

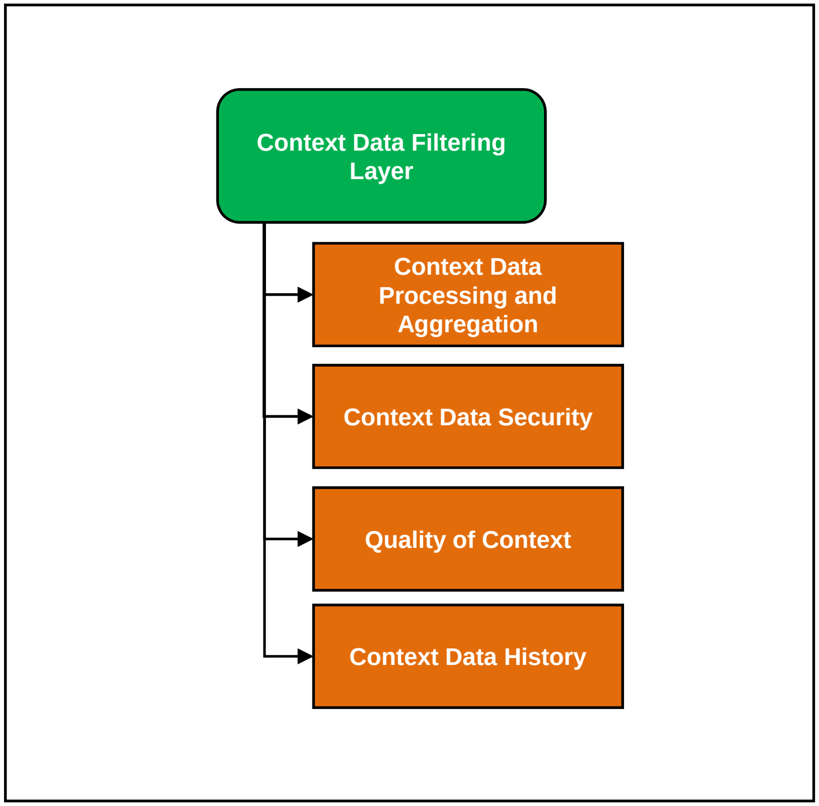


The **Context Data Distribution Service** can be sub-divided in four principal layers:

1. The **Context Data Filtering Layer** tailors context data distribution and retrieval according to specific application needs
2. The **Context Data Dissemination Layer** realizes the algorithm/s used by the data distribution service to disseminate the data inside the system
3. The **Dissemination Infrastructure Layer** maps dissemination algorithms on the real existing data dissemination infrastructure
4. The **Adaptation Support** realizes the software infrastructure needed by the obvious cross-cutting concern, i.e., adaptation

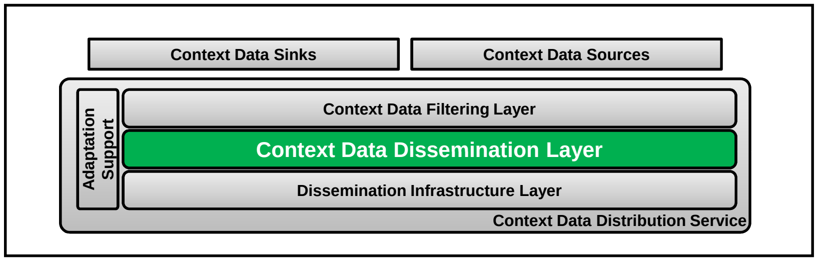
Context Data Filtering Layer

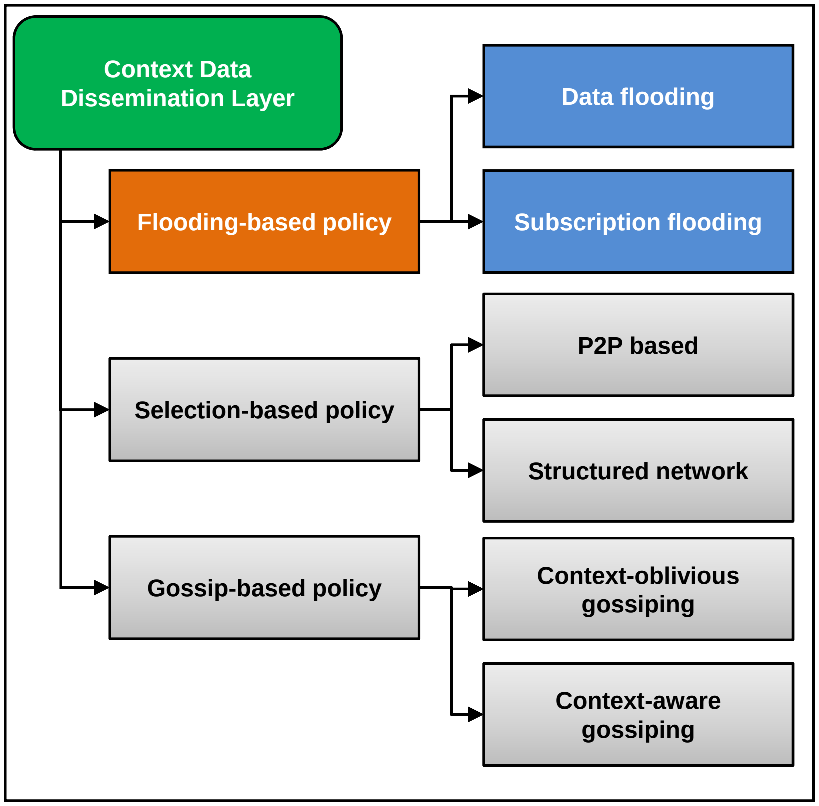




- The ***Context Data Processing and Aggregation*** function offers processing and aggregation facilities
- The ***Context Data Security*** function introduces security primitives to elaborate data produced by sources and received by sinks
- The ***Quality of Context (QoC)*** function introduces data quality indicators. Above all, the system should consider creation time to deal with data aging
- The ***Context Data History*** function enables the retrieval of data according to specific time coordinates

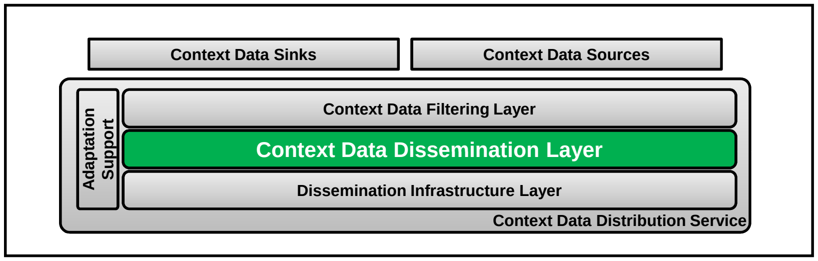
Context Data Dissemination Layer: Flooding-based protocols

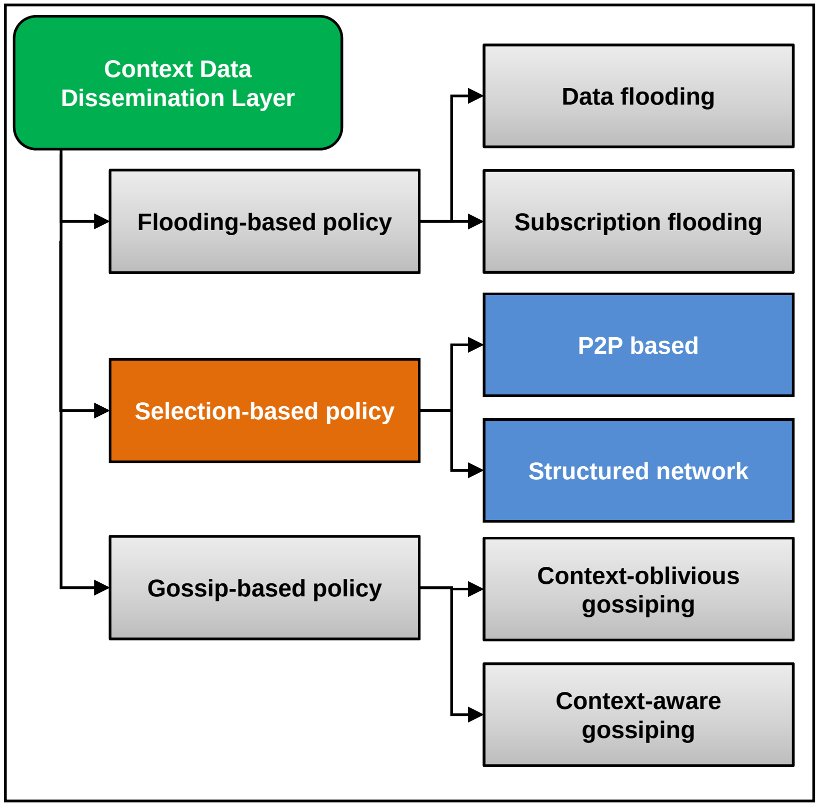




- **Flooding-based algorithms** are deterministic approaches based on broadcast operations. These algorithms can be divided in:
  1. **Data flooding –** Each node broadcasts known data to spread them inside the entire system. Receiver nodes locally filter received data using sink registrations
  2. **Subscriptions flooding –** Each node broadcasts its context data subscriptions to all nodes to build routing structures. Each node memorizes the subscriptions from all other nodes to perform local matching on produced data (this schema assumes that all node pairs have a one-hop distance).

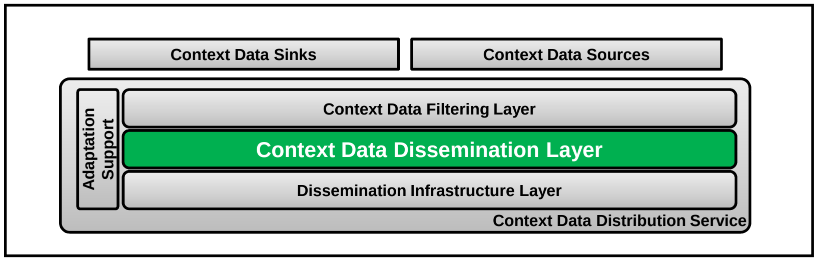
Context Data Dissemination Layer: Selection-based protocols

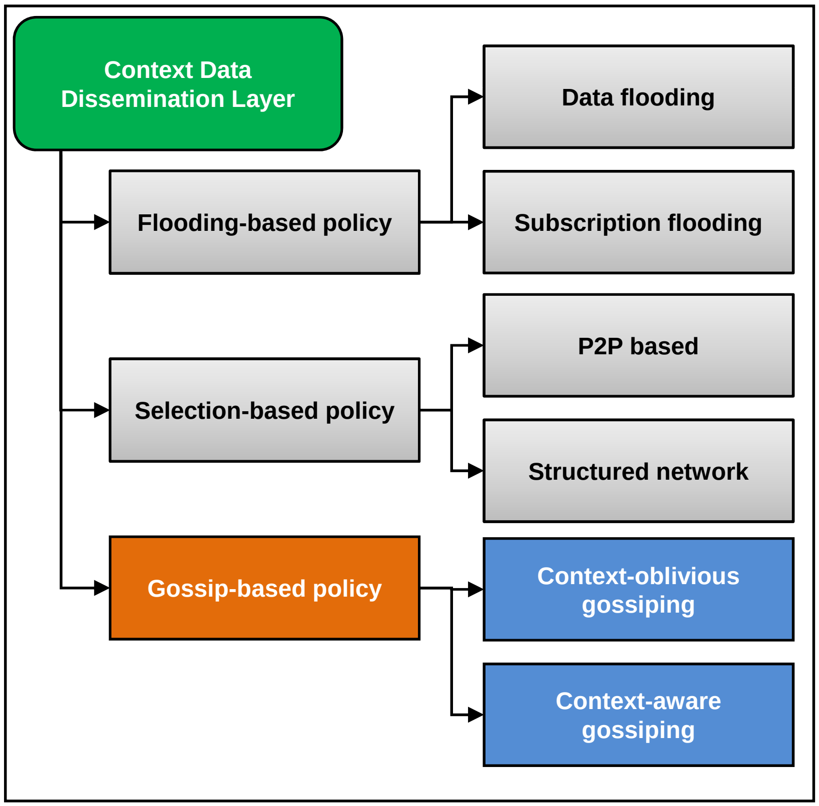




- **Selection-based algorithms** are deterministic approaches that exploit context data subscriptions to build dissemination backbones. Dissemination takes place only over the backbones, and reaches only interested nodes. These algorithms can be divided in:
  1. **P2P based –** All nodes belonging to the system are feasible overlays inside dissemination backbones
  2. **Structured network –** Only some nodes are selected to compose the dissemination backbones. Selection policies usually prefer powerful nodes

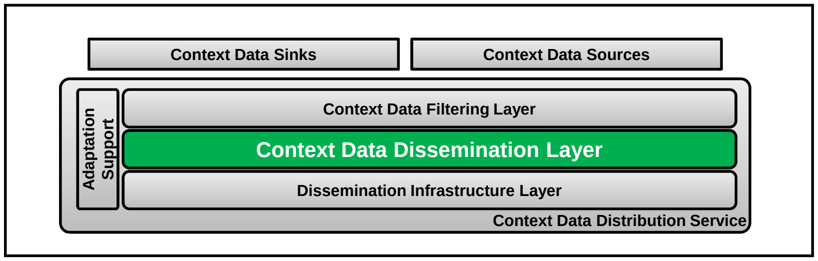
Context Data Dissemination Layer: Gossip-based protocols





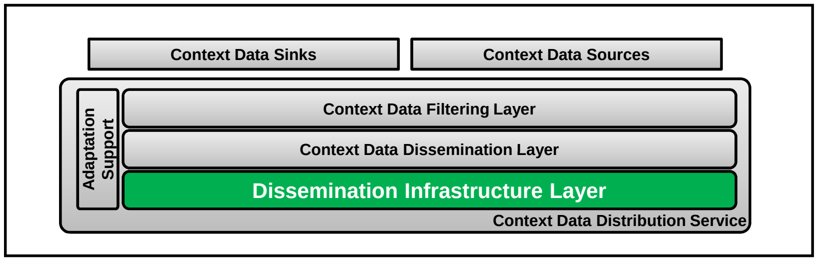
- **Gossip-based algorithms** are probabilistic approaches. Dissemination exploits all nodes inside the system and traditionally reaches not interested nodes. Receiver nodes locally filter received data using sink registrations. These algorithms can be divided in:
  1. **Context-oblivious gossiping** – Each node sends data to randomly selected neighbors without considering any external context information
  2. **Context-aware gossiping** – Each node sends data to selected neighbors. To increase the probability to hit interested nodes, context-aware protocols select neighbors to gossip data to by using external context information

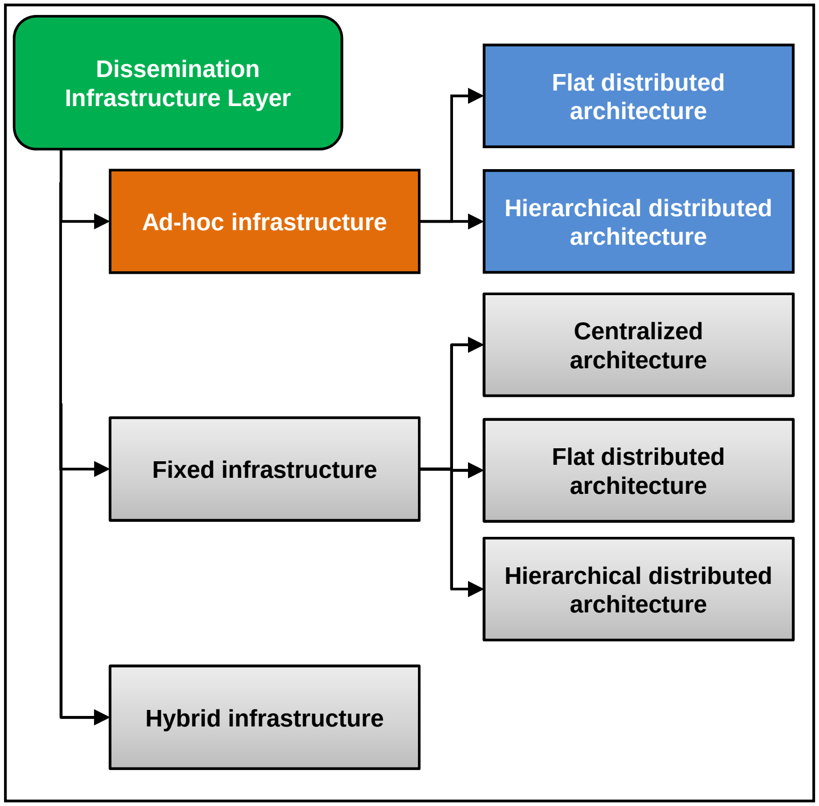
Context Data Dissemination Layer: Summary



Category	Sub-category	Pros	Cons
Flooding-based policy	Data flooding	• Deterministic dissemination • Small state on involved nodes • High data replication	• Dissemination traffic
	Subscription flooding	• Deterministic dissemination • Less traffic when subscriptions are smaller than data	• Dissemination traffic • Heavy routing tables
Selection-based policy	P2P based	• Deterministic dissemination • Dissemination reaches only interested nodes	• Routing structures building and maintenance
	Structured network	• Deterministic dissemination • Dissemination reaches only interested nodes • The selection process can increase the robustness/performance of the dissemination	• Routing structures building and maintenance
Gossip-based policy	Context-oblivious gossiping	• Small state on involved nodes • More scalable than data flooding approach	• Probabilistic dissemination • Low probability to hit interested nodes
	Context-aware gossiping	• Small state on involved nodes • High probability to hit interested nodes	• Probabilistic dissemination • High state on involved nodes

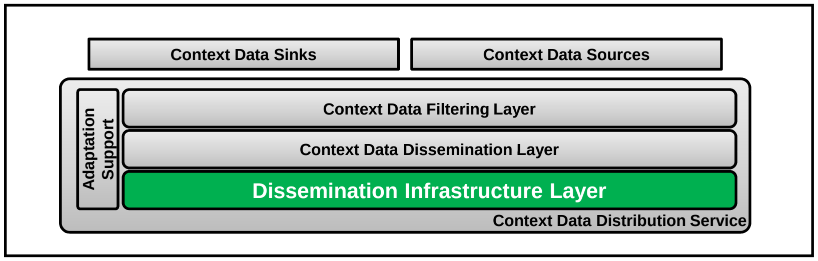
Dissemination Infrastructure Layer: Ad-hoc infrastructure

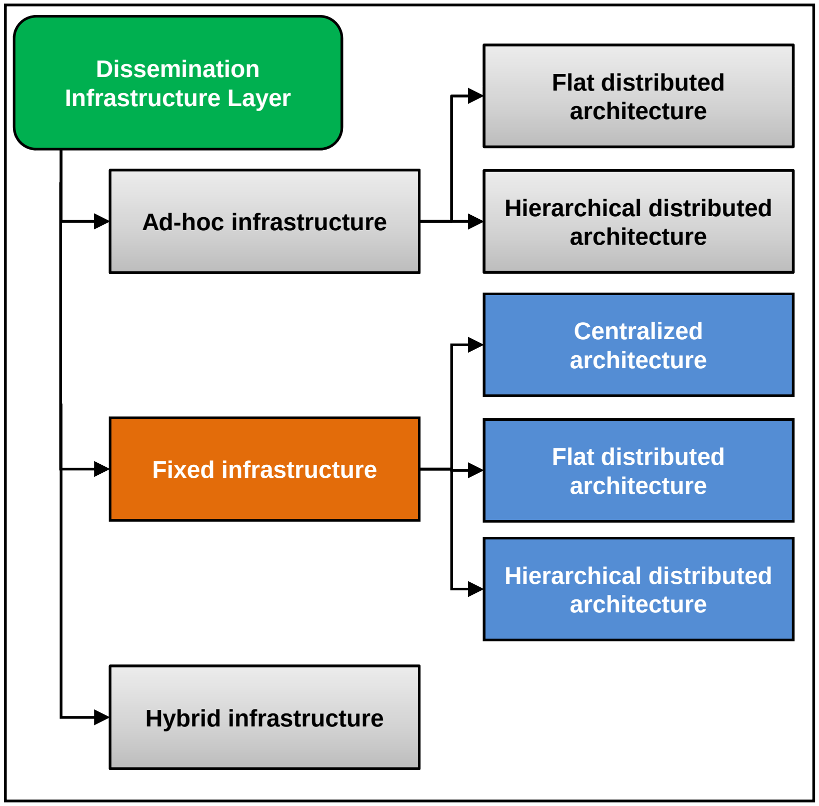




- The dissemination infrastructure is based on a completely decentralized approach. Communications between nodes exploit only ad-hoc links. No fixed infrastructure is needed. According to low-level routing protocols, we distinguish:
  1. **Flat distributed architecture –** Nodes form a P2P environment in which everyone has the same responsibilities
  2. **Hierarchical distributed architecture –** Nodes are categorized in cluster-heads, gateways, and simple nodes. Traditional clustering protocols aim to contain protocols overhead by building routing backbones based only on gateways and cluster-heads

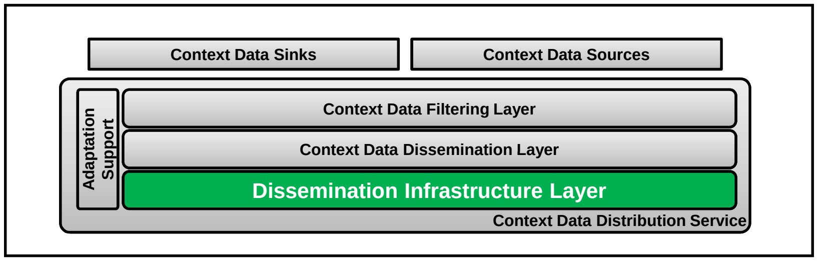
Dissemination Infrastructure Layer: Fixed infrastructure

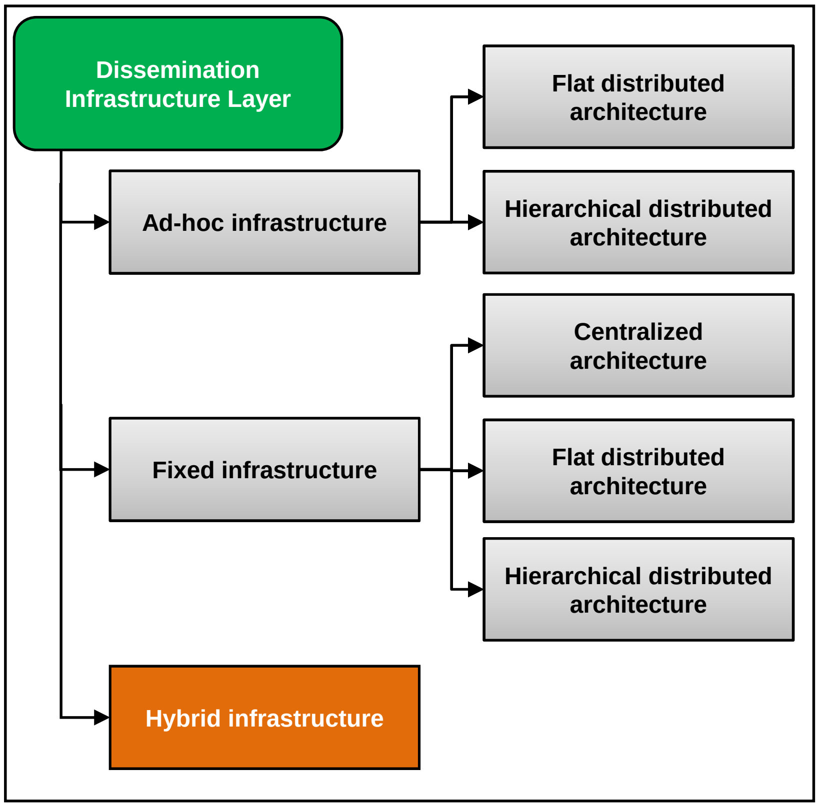




- The dissemination infrastructure is based on a service reachable through the fixed infrastructure. All communications inside the system introduce traffic on the fixed infrastructure. Even in this case, we distinguish:
  1. **Centralized architecture –** The service is based upon a unique central node
  2. **Flat distributed architecture –** The service is based upon different nodes organized in a P2P fashion
  3. **Hierarchical distributed architecture –** Similar to the previous, but nodes are organized hierarchically according to whatever locality principle

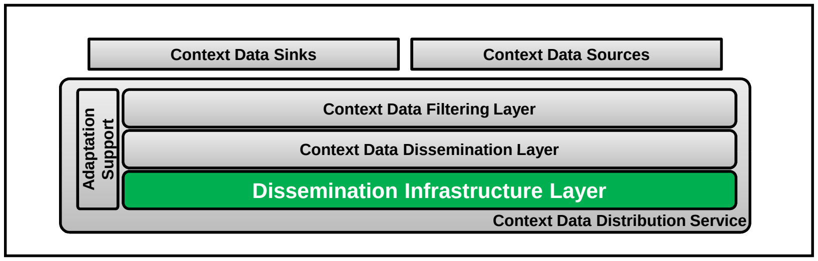
Dissemination Infrastructure Layer: Hybrid infrastructure



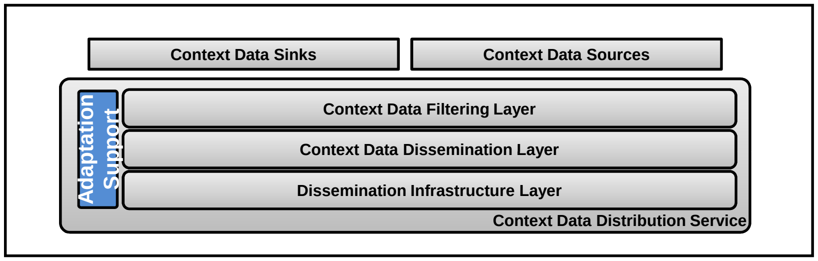


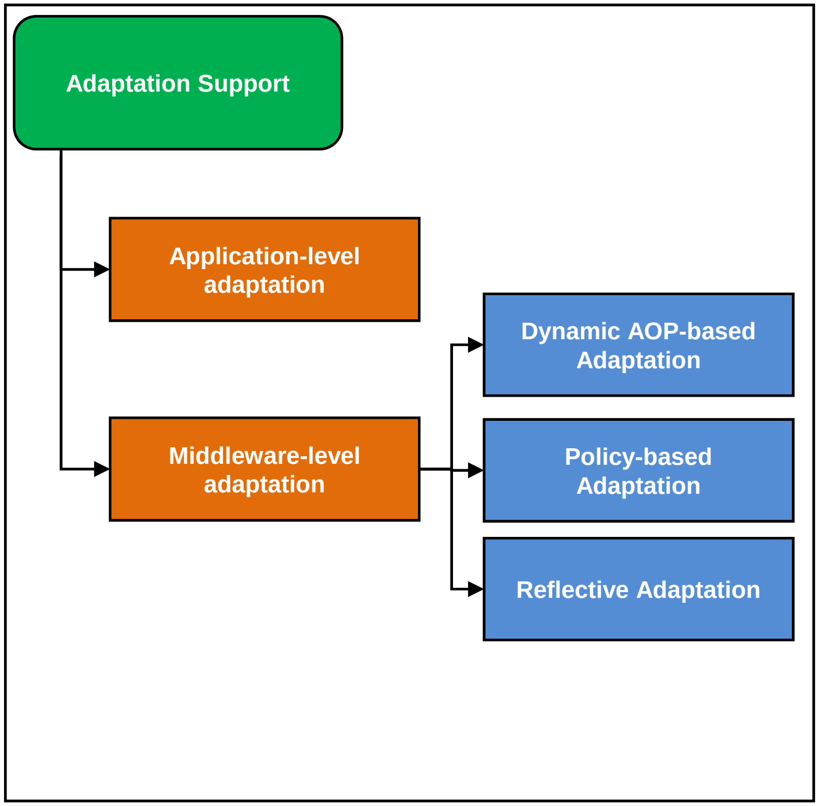
- The dissemination infrastructure exploits both the previous approaches. Obviously, both the fixed and the ad-hoc infrastructure can be organized according to the possibilities showed in the previous slides
  1. **Ad-hoc links –** Ad-hoc links enable cheap dissemination inside the system. This opportunity is very well suitable when data obey to the **physical locality principle**
  2. **Fixed infrastructure –** Fixed infrastructure ensures data access. It is useful when data do not obey to the **physical locality principle**

Dissemination Infrastructure Layer: Summary



Category	Sub-category	Pros	Cons
Ad-hoc infrastructure	Flat distributed architecture	• Dissemination guaranteed in infrastructure-free scenarios	• Context data access not guaranteed • Up-to-date data problem
	Hierarchical distributed architecture	• Dissemination guaranteed in infrastructure-free scenario • Scalability	• Context data access not guaranteed • Up-to-date data problem
Fixed infrastructure	Centralized architecture	• Guarantee of Context data access • No load on mobile nodes	• Scalability • Dissemination not guaranteed in infrastructure-free scenario
	Flat distributed architecture	• Guarantee of Context data access • No load on mobile nodes • Scalability	• Coordination • Dissemination not guaranteed in infrastructure-free scenario
	Hierarchical distributed architecture	• Guarantee of Context data access • No load on mobile nodes • Scalability based on physical locality principles	• Coordination • Dissemination not guaranteed in infrastructure-free scenario
Hybrid infrastructure	All possible combinations	• Data dissemination guaranteed both in infrastructure-free and infrastructure-enabled scenarios • Real system scalability	• Coordination between the different dissemination infrastructure is difficult • Up-to-date data problem





- The **Adaptation Support** enables both application- and middleware level-adaptation. The second one can be categorized in:
  1. **Dynamic AOP-based Adaptation –** Aspect Oriented Programming with run-time weaving
  2. **Policy-based Adaptation –** Applications supply profiles that detail what kind of service they require, while the middleware provides a service as close as possible to these requests
  3. **Reflective Adaptation –** The middleware provides a self-representation changeable by applications. These modifications affect the middleware itself since there is a causal relation representation-middleware

Context Data Distribution Service Issues

Reconsidering our principal scenario, the **Context Data Distribution Service** must distribute data taking care of:

1. **Mobility –** Mobility results in a management overhead related with device population and mobility patterns. Besides, handoff and reconfiguration mine data distribution dependability
2. **Heterogeneity –** Our principal scenario groups many devices with different computational capabilities and different wireless standards (WiFi, BT, etc.)
3. **Scalability –** Many different context data attributes deeply influence the data distribution overhead. Consequently, the distribution process can overwhelm the entire system

To enhance system scalability, we should exploit solutions belonging to two different principal levels:

- **Context data level –** To reduce the introduced overhead, the data distribution service should **handle data lifecycle,** and should **adapt distribution process** according to data attributes
- **Communications level –** To increase system scalability at communications level, the distribution service should use **different wireless standards and modes**

Enable System Scalability: Context data level

1. **Constraint data dissemination –** Data must be disseminated respecting their **visibility scope**. Imposing visibility scope reduces introduced overhead
   I. **Physical-locality principle:** location-dependent data must be kept near associated sources
   II. **Logical-locality principle:** data must be disseminated only to interested node

2. **Consistency –** Data consistency inside the visibility scope is an appealing property. Unfortunately, the performance/cost ratio is usually unsuitable in real distributed system
   i. Ensuring system-wide consistency is traditional very costly
   ii. Ensuring consistency for small environments can be feasible
   iii. As golden rule, **take into account the data semantic** and **decide consistency policies accordingly**

Enable System Scalability: Context data level

3. **Active data processing to reduce overhead –** The introduced overhead depends also on:
   1. **Payload length –** The bandwidth overhead increases with payload size
   2. **Production rate –** The bandwidth overhead is directly proportional to the data production rate. Data with large payload can represent a small overhead if production rate is very low, and vice-versa

   To reduce overhead, different processing techniques are available:
   1. Disseminate different versions of the same data to reduce **payload length**
   2. Tailor **production rate** inside the visibility scope according to the **consistency policies**
   3. If different visibility scopes exist, tailor **production rate** according to **consistency policies** and to **each scope**

4. **Life cycle –** Distribution process must handle data life cycle to disseminate only **valid data**. Life cycle management can exploit time deadline, version control, and so forth

Enable System Scalability: Context data level

5. **Adaptive dissemination process –** Use different dissemination algorithms depending on **available bandwidth**, **data scope** and deployed **Dissemination Infrastructure Layer**
   i. Tailor data dissemination according to the available bandwidth
   ii. Use different dissemination algorithms according to the **data scope** and the deployed **dissemination infrastructure**

Context Data Dissemination Layer and **Dissemination Infrastructure Layer** are tightly coupled → Use **cross-layer techniques**

For instance:

i. If the dissemination scope is very small, flooding-based algorithms are feasible
ii. Flooding-based algorithms with an ad-hoc wireless infrastructure take advantage from the broadcast nature of the wireless medium
iii. Selective-based approaches are feasible if they are implemented on a mobile network composed by almost static nodes
iv. Selective-based approaches on fixed infrastructure are very feasible
v. Gossip-based approaches are very feasible if nodes are highly mobile
vi. …

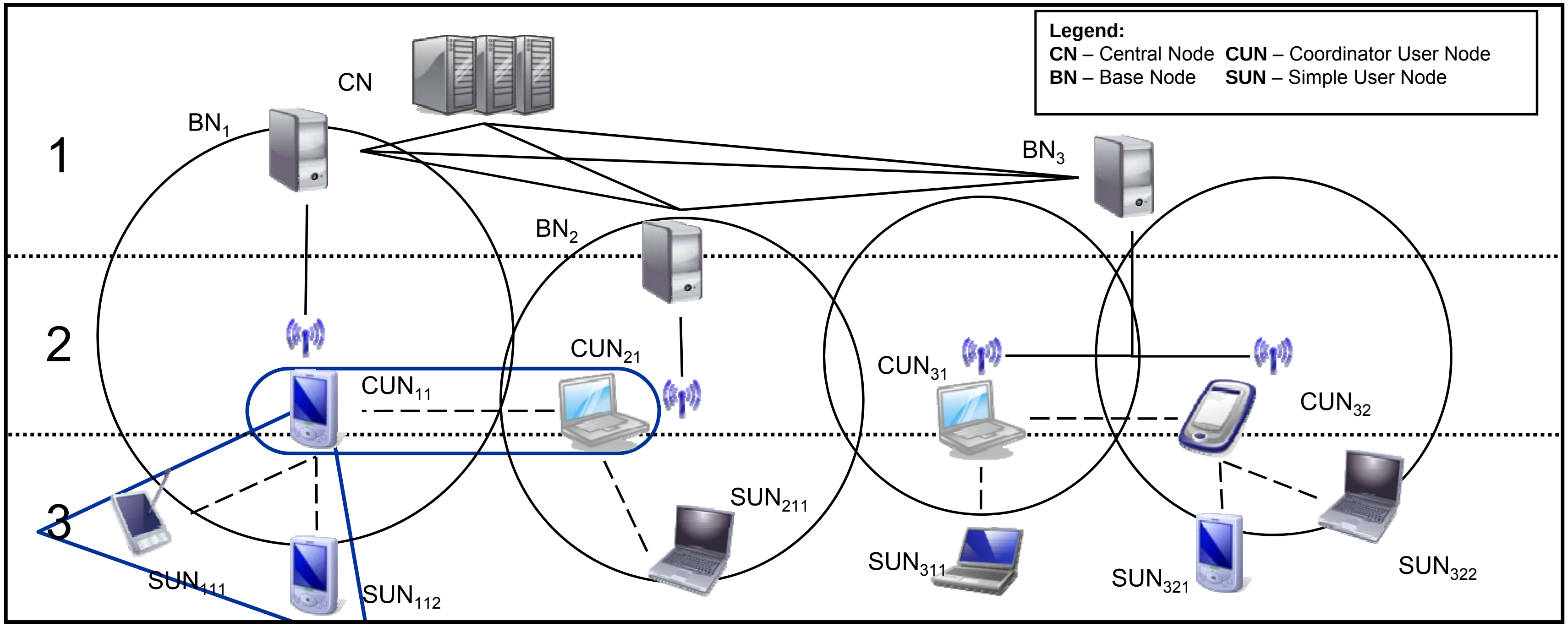
Enable System Scalability: Communications level

1. **Use heterogeneous wireless communications**
   – Supporting different wireless standards increases system coverage and total available bandwidth

2. **Exploit different wireless modes**
   – Fixed infrastructures guarantee data availability
   – Ad-hoc links enable cheap data dissemination

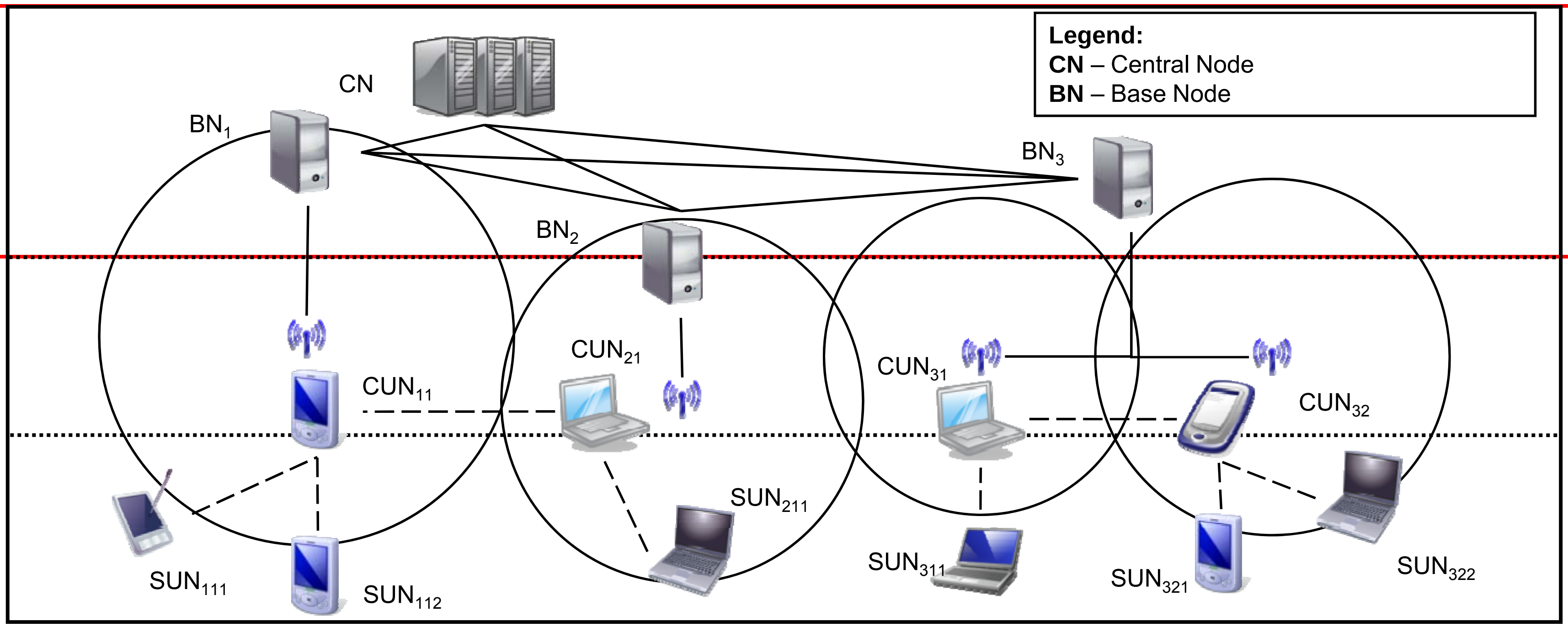
3. **Adapt mobility management protocols**
   – Adapt mobility management protocols according to nodes joint mobility and wireless transmission range

Current solution: Scalable context-Aware middleware for mobiLe EnviromentS



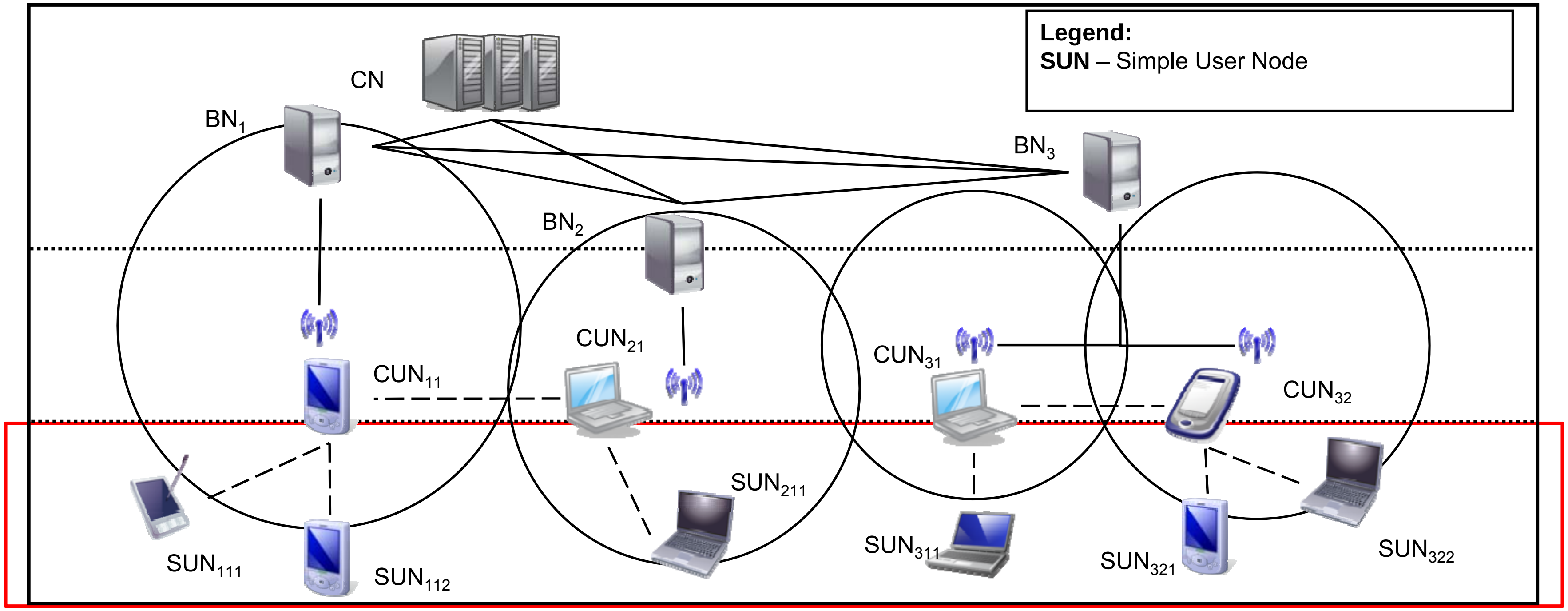
- **Tree-like three-level** architecture
- The distributed architecture reflects **physical locality principle** → Each father node groups near child nodes
- Nodes belonging to the same level form a **collaborative network** in which data can be disseminated in a peer-to-peer (P2P) manner

SALES Fixed Infrastructure



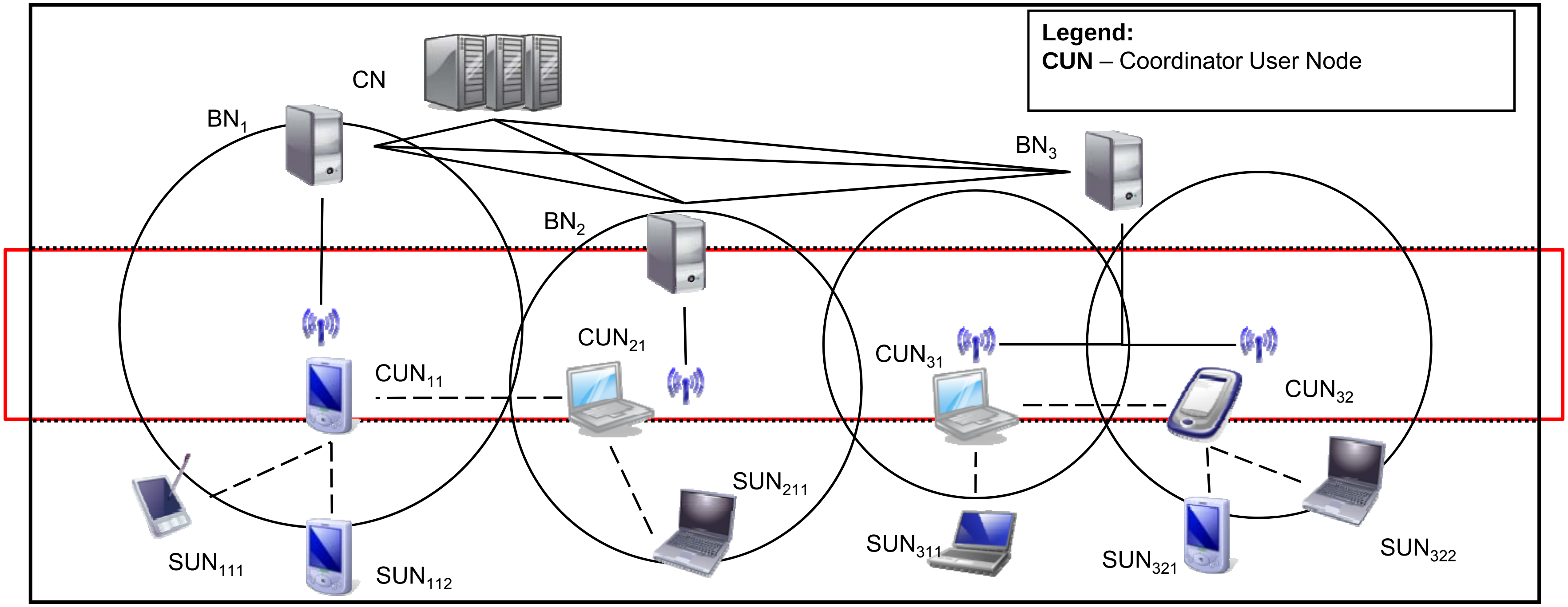
- **Fixed infrastructure**
  - Central Node ensures data history and access
  - Base Nodes are the SALES fixed infrastructure entry points
  - Base Nodes memorize context data to reduce the requests routed up to the Central Node

SALES Mobile Infrastructure: SUN



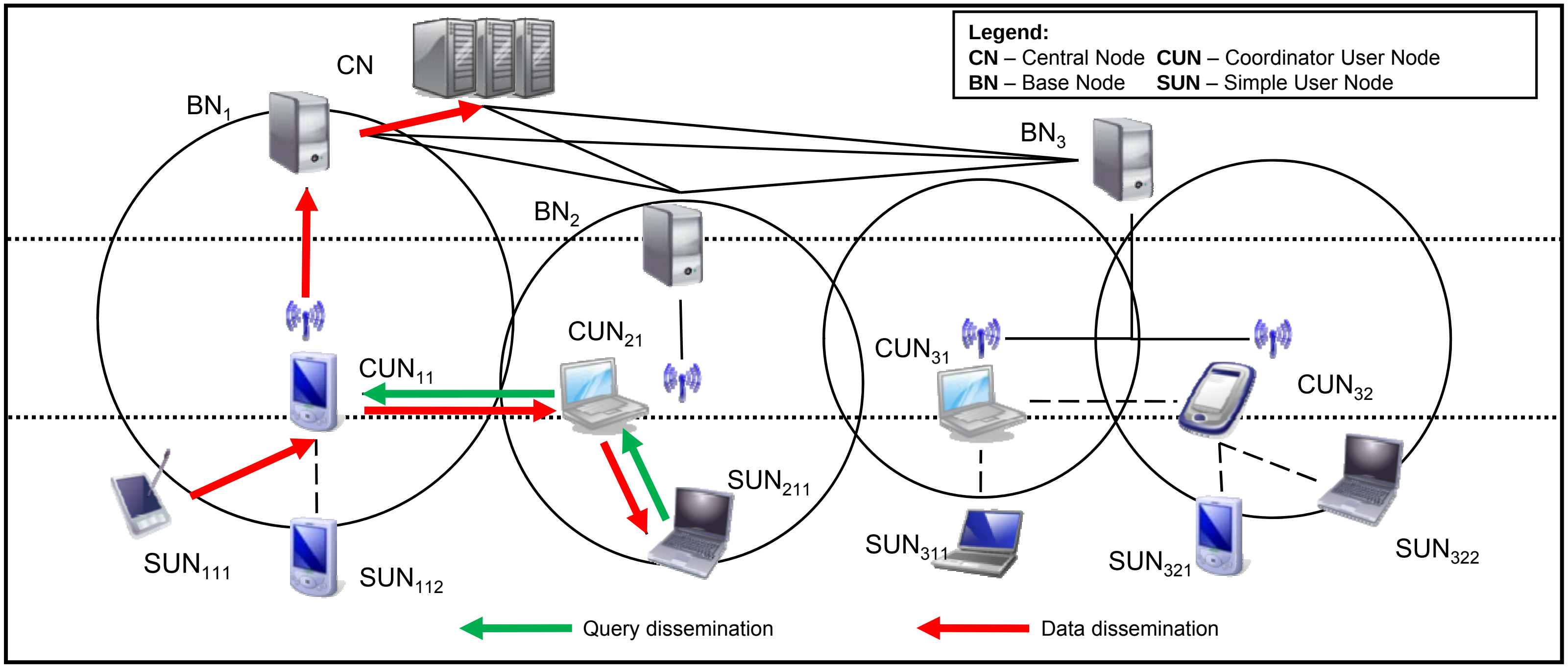
- **Mobile infrastructure**
  - Communications between user nodes exploit only ad-hoc links
  - Simple User Nodes share local context data repositories with peers
  - Coordinator User Nodes share local context data repositories with peers and served nodes

SALES Mobile Infrastructure: CUN



- **CUNs bridge together the fixed and the mobile infrastructure**
  - CUNs should be multi-homed nodes
  - CUNs enact as routers even between different technology-specific networks
  - Mobility management protocols between a CUN and its served SUNs are completely based on ad-hoc links

Context data dissemination



- To build dissemination paths, SALES adopts **context queries.** A context query captures context needs by imposing constraints on data values
- The data dissemination takes place as follows:
  1. At default, data flow only on the bottom-up path between the data creator node and the Central Node
  2. Different dissemination paths are considered only if matching queries exist

Context data dissemination details

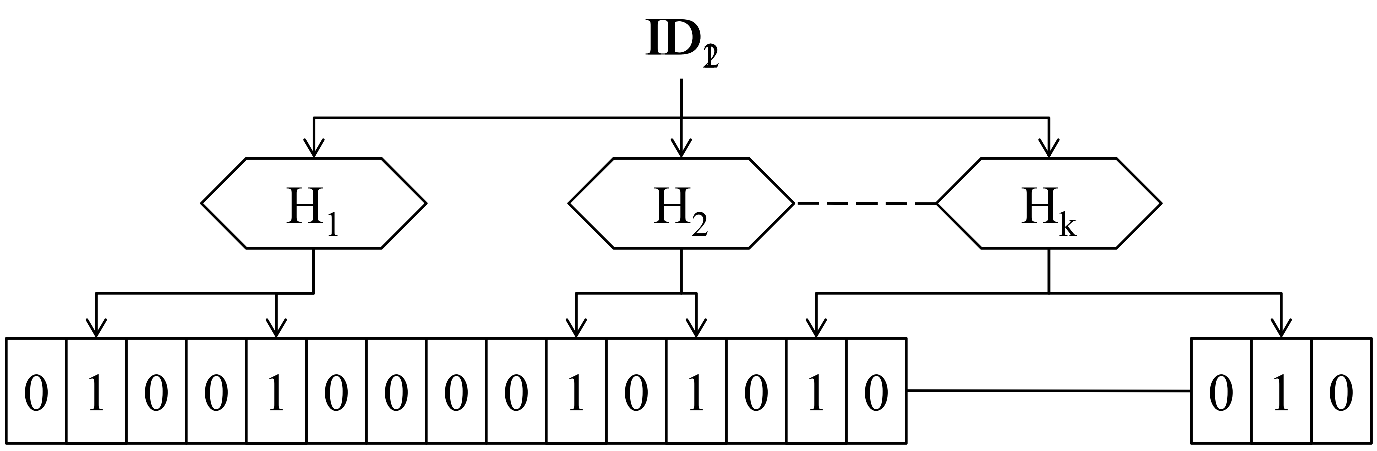
- To build different dissemination paths, each query is disseminated inside the SALES distributed architecture:
  1. First, the query is disseminated on the same level **(horizontal propagation)**
  2. Then, the query is disseminated on the upper level **(vertical propagation)**
  3. Repeat from 1. until the query is valid and current node is not the CN

- Both context query and data have different parameters that affect dissemination process. Each data has two different parameters:
  1. **Hierarchical Level Tag:** The maximum visibility inside SALES system
  2. **Data lifetime:** Deadline used to limit data lifetime

  Each query has four different parameters:
  1. **Horizontal time to live**: The maximum number of nodes traversed at the same hierarchy level. It is used to constraint horizontal query scope
  2. **Routing delay:** Delay used to temporize the query dissemination process
  3. **Query lifetime:** Deadline used to limit query lifetime
  4. **Maximum context query responses:** The maximum number of wanted context responses. Used to further limit query lifetime

Bloom filter

- To reduce management overhead associated with queries, SALES optimizes their representation using **Bloom filters**
  1. A **Bloom filter** is a **space-efficient probabilistic data structure** useful to support **membership queries**
  2. Given a set A={a_1, a_2, …., a_n} of *n* keys, the associated filter is a bit vector of *m* bit obtained by setting to 1 all bits at positions $h_1(a)$, $h_2(a)$, ..., $h_k(a)$ for each element $a \in A$

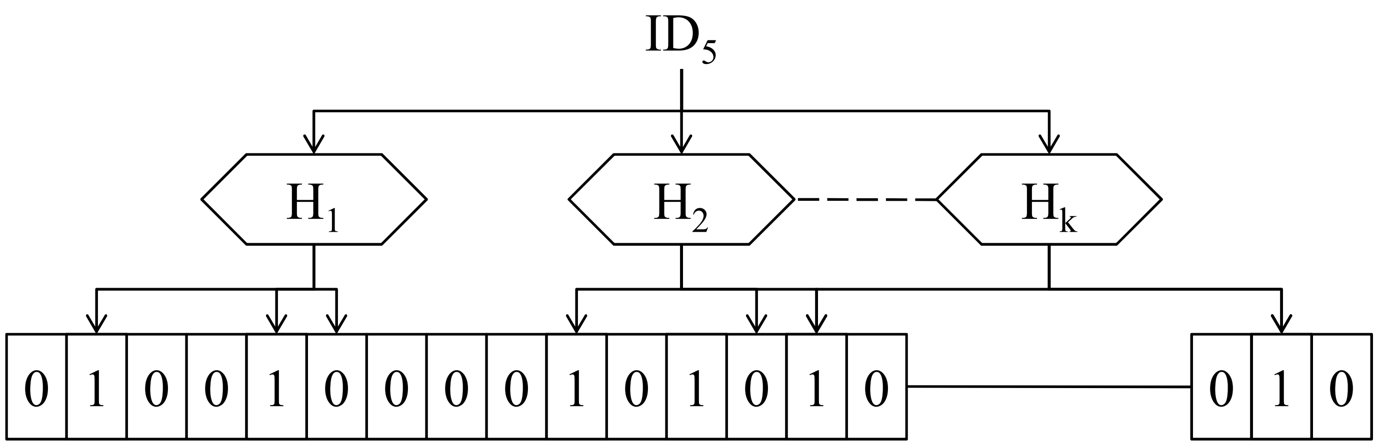


Bloom filter of {ID_1, ID_2} set

Bloom filter

Given a query for a generic key b, check all the bits at positions $h_1(b),\dots, h_k(b)$ and

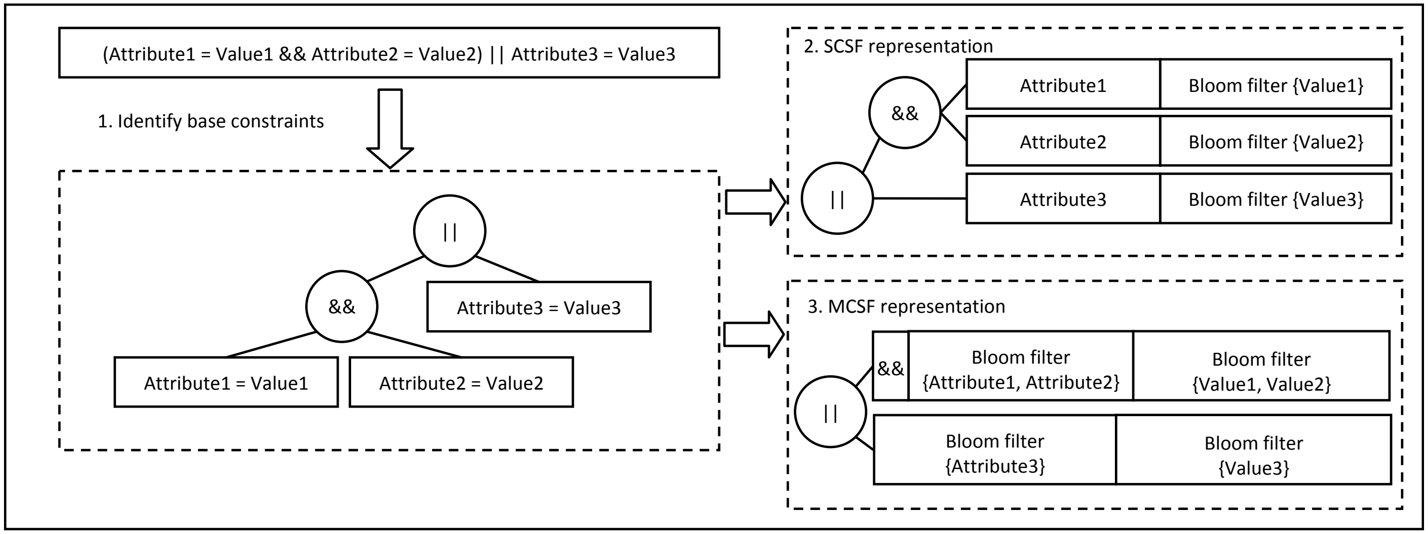
- if any of them is 0, then certainly b is not in the original set
- otherwise, we assume that b is in the set, although it may not be the case because we may have false positive events



False positive

The **false positive ratio probability** is equal to **0.6185^(m/n)**: thus, given an upper bound to |A|, we can reduce this probability by increasing m

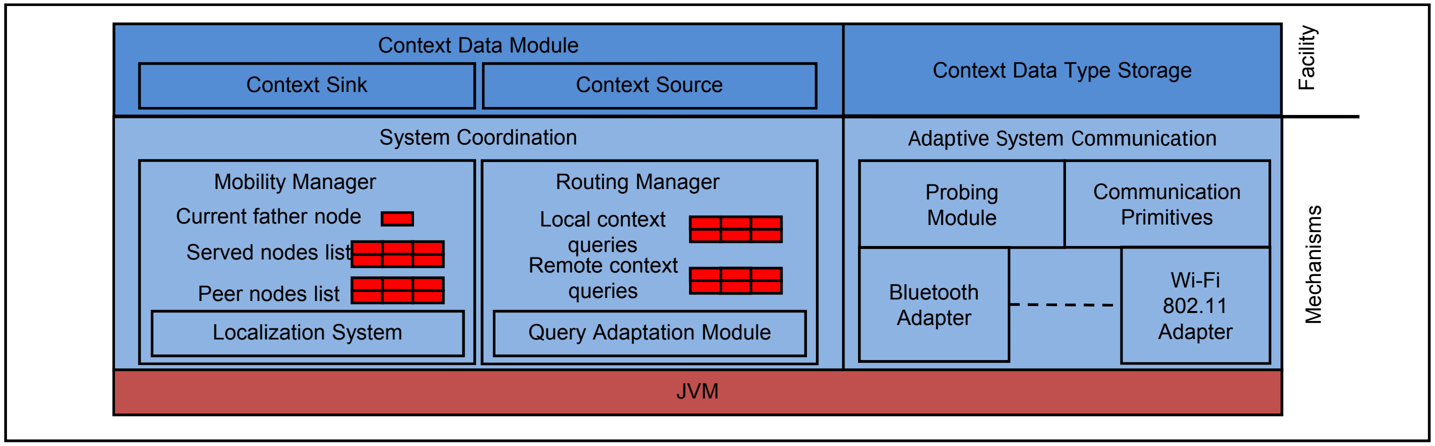
Query representation



- SALES represents queries in two different ways :
  1. **Single Constraint - Single Filter (SCSF):** This modality maps each base constraint to a proper name-filter pair. False positive ratio is very easy to control
  2. **Multiple Constraints - Single Filter (MCSF):** This modality aims to obtain very small query. It merges together different base constraints (tied by the same logical operator) to reduce the final size. False positive ratio is difficult to control since we mix different hash functions

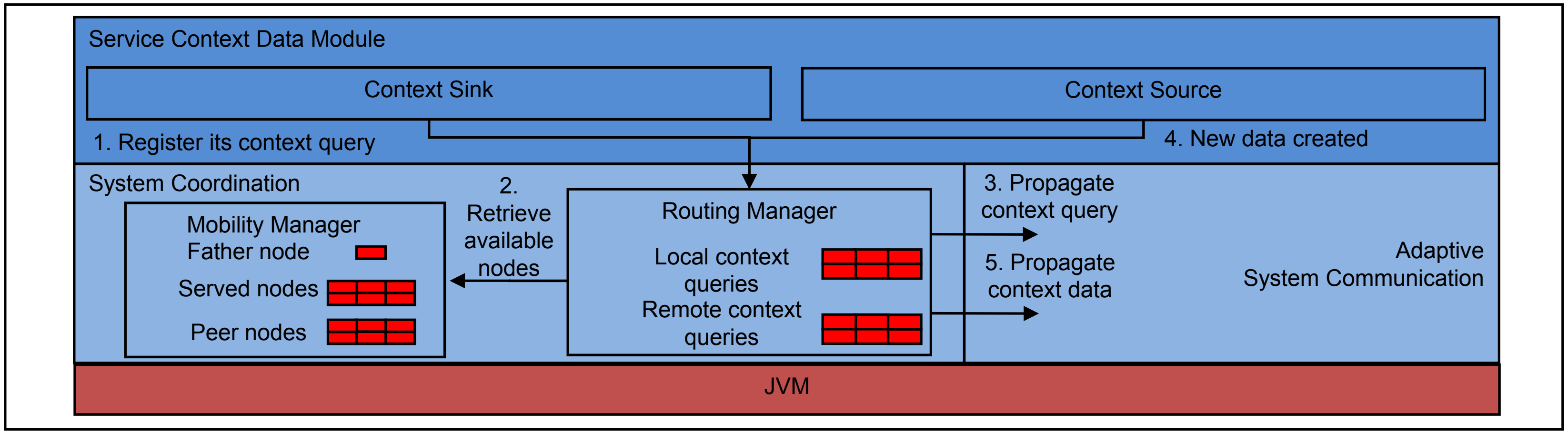
Representation is selected according to channel status

SALES middleware architecture



- **Facility Layer**
  - Each context data type is associated with a *Context Data Module*. The source enables the data injection, while the sink addresses the data retrieval
  - *Context Data Type Storage* maintains context data type definition
- **Mechanisms Layer**
  - *Adaptive System Communication* offers communication primitives
  - *System Coordination* addresses mobility and data dissemination

Implementation insights



- *Routing Manager* has two different query tables:
  - **Local context queries** stores queries emitted by local sinks
  - **Remote context queries** stores queries received by other SALES nodes
- Local context sinks push local queries to the *Routing Manager* (step 1)
- When a query dissemination is needed, *Routing Manager* retrieves destination nodes, either peer nodes or father node, (step 2) and propagates the query (step 3)
- When new data are produced (step 4), they are matched against local and remote queries and propagated consequently (step 5)

Experimental evaluations

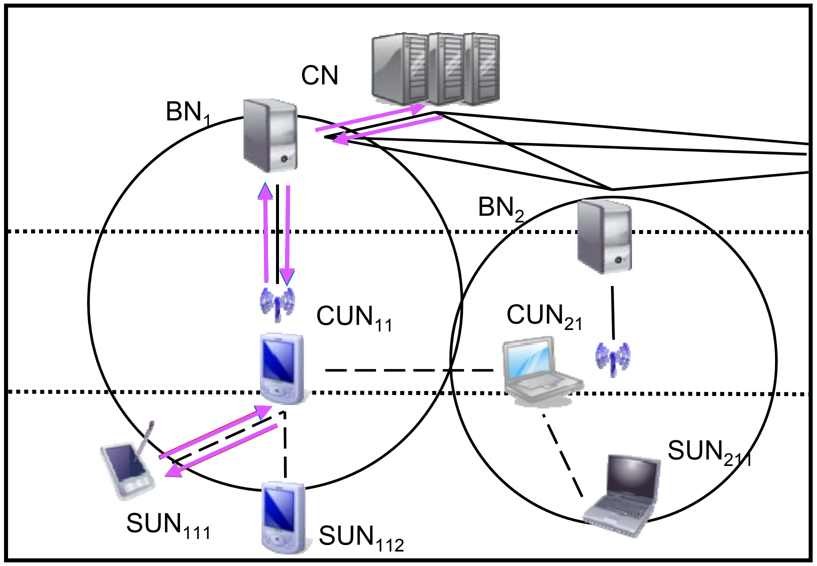
Experimental testbed

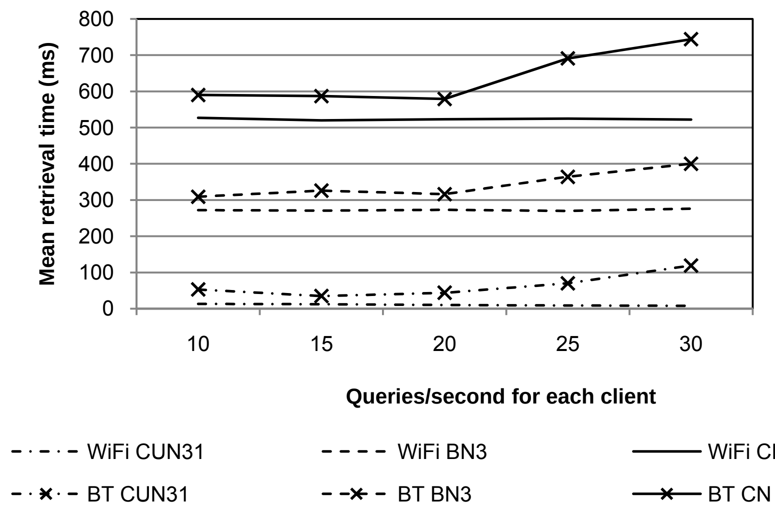
- SALES Middleware has been completely realized on J2SE 1.6
- BNs and CN execute on 2 CPUs 1,80GHz, 2048MB RAM, Linux Ubuntu
- Wireless infrastructure composed by Wi-Fi Cisco Aironet 1100 AP
- Test stations with 2 IEEE 802.11g Realtek RTL8185, Bluetooth dongle and Linux Ubuntu
- Each context query in the subsequent tests has a routing delay of 250 milliseconds, an horizontal time to live equal to 0, a lifetime of 3 seconds and a maximum context query responses equal to 1
- Test code with 10 clients that send contemporary a variable number of context data requests
- Bandwidth control enabled

We executed tests modifying:

1. the SALES level able to supply the context data response
2. the query representation technique
3. the adopted wireless communication standard

Data retrieval time: Different SALES level

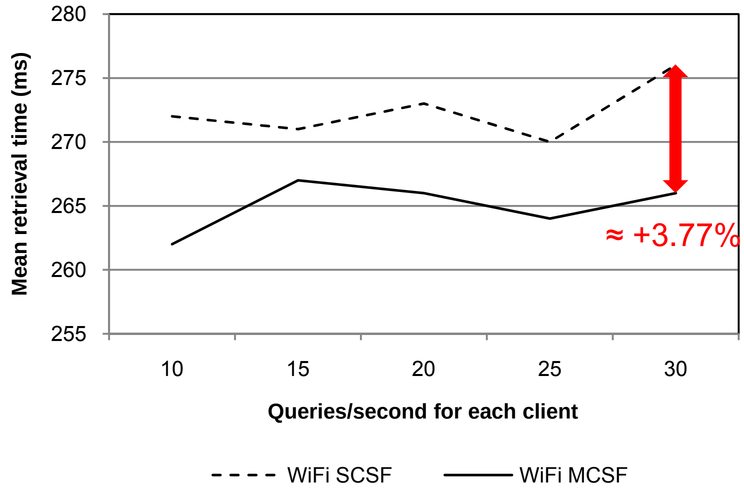




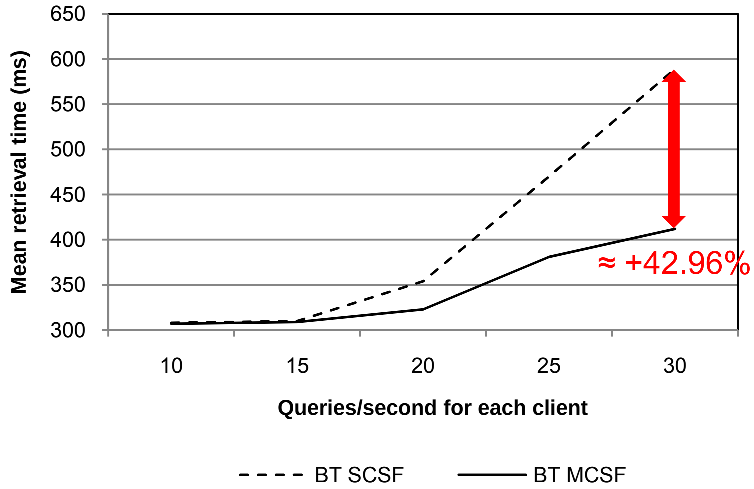
- Each involved node in the dissemination path introduces a routing delay equal to 250 milliseconds
- Since the horizontal time to live is equal to zero, horizontal query propagation is not performed
- **Physical locality principle** reduces **data retrieval time**: the nearer the node able to supply the data, the smaller retrieval times
- Obviously, the usage of BT between SUN_{111} and CUN_{11} results in higher retrieval times than the ones obtained with WiFi (due to the bandwidth limitations)

Data retrieval time: Different query representation

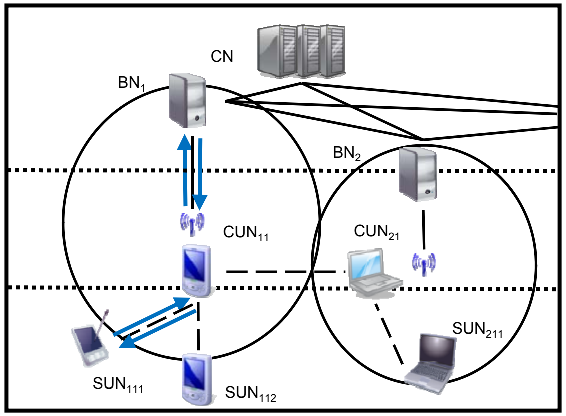
BT bw ≈ 0.7 Mbit/s



Wifi bw ≈ 6.1 Mbit/s

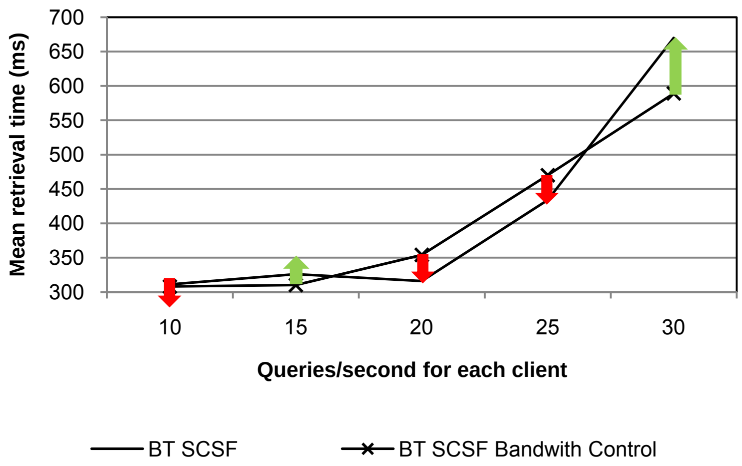


- MCSF schema obtains shorter retrieval times than the ones obtained by SCSF
- This effect is a consequence of the bandwidth control: the ratio between the query sizes obtained with the SCSF and the MCSF representation is approximately equal to 3
- In the BT case, the SCSF consequences are terrible ☺

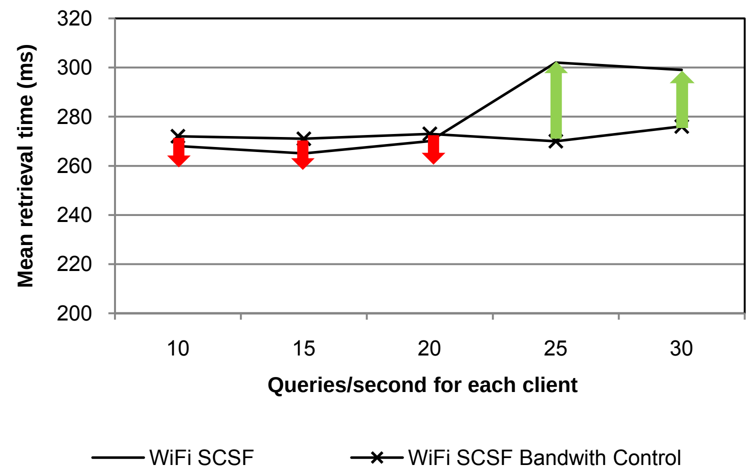


Data retrieval time: Effect of the bandwidth control

BT bw ≈ 0.7 Mbit/s



Wifi bw ≈ 6.1 Mbit/s



- When bandwidth control is disabled, SALES sends messages as soon as possible
    1. With small sending rate, that results in shorter retrieval times than the ones obtained with bandwidth control enabled
    2. With high sending rate, the results suddenly worsen. Consequently, retrieval times become longer than the ones obtained with bandwidth control
- Limiting sending rate according to available bandwidth, SALES avoids wireless channel saturation. Besides, it prevents continuous MAC-level back-off invocations that, in their turn, can result in higher retrieval times

Conclusions and ongoing work

Conclusions:

Data dissemination must be carefully addressed to obtain **scalability**

- **Locality** enhances scalability by constraining data dissemination scope
- **Mixing ad-hoc with infrastructure-based communications** reduces overhead
- **Different wireless communication standards** increase total available bandwidth

Ongoing work:

- **Multiple dissemination paths** to increase dependability
- **Different dissemination algorithms**, e.g., flooding- or gossip-based, according to data scope and environmental conditions

SALES project web site and contacts

- Prototype code and information:
  http://lia.deis.unibo.it/Research/SALES

- Contacts: Mario Fanelli (mario.fanelli@unibo.it)

Thanks for your attention!

Questions