DAD
Desenvolvimento de Aplicações Distribuídas

Publish-Subscribe Systems

Agenda

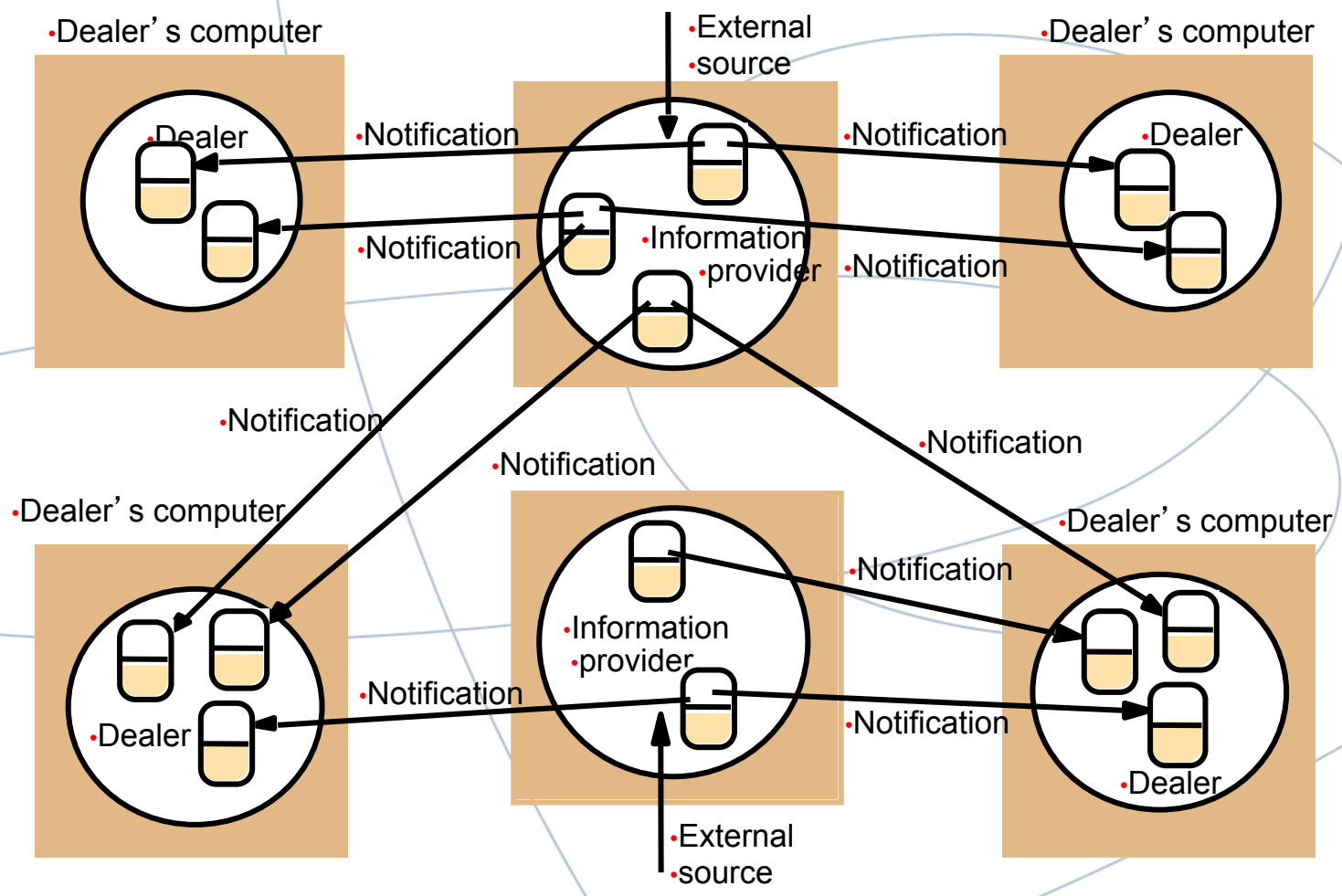
- **Introduction**
- **Elements in Pub-Sub**
- **Subcription Models**
- **Architecture Model**
  - transport, infrastructure, subscription matching, routing
- **Distributed Event Routing**
  - flooding, selective (filtering/rendez-vous), gossip,...
- **Example Systems**
  - Scribe, Siena, ...

Introduction

- **Publish-Subscribe Paradigm**
  - intuitive, application-driven
    - stock tickers, news feed, air traffic control, location events, application and Internet monitoring.
  - full decoupling
    - senders (publishers) and receivers (subscribers)
    - no direct knowledge, possibly anonymous
    - no synchronous interaction
  - indirect communication
    - via events, no explicit addressing, routing according to subscriptions
  - scalability
    - large-scale networks, large numbers of publishers, subscribers, subscriptions and events

Introduction

- **Publish-subscribe Example**

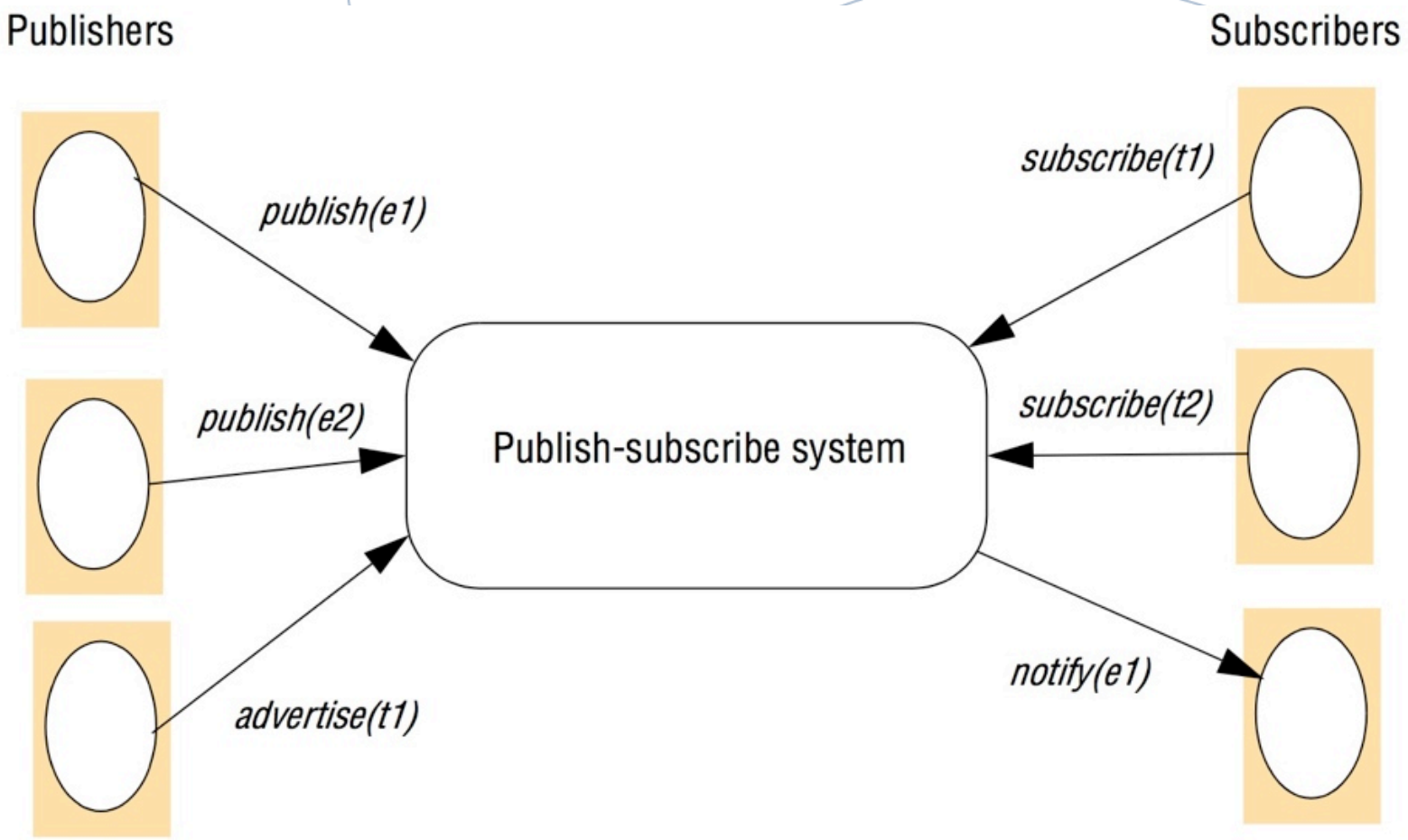


Introduction

- **Elements of a Pub-Sub System**
  - **clients**: publishers, subscribers
  - **events**: set of (named) attribute-value pairs
  - **subscription (f)**: filter, set of constraints
  - basic **operations** on events
    - *publish(e),*
    - *subscribe(f), unsubscribe(f)* (also σ for *subscription*)
    - *advertise(f), unadvertise(f)*
  - **(matching)**: test event against subscription filter
  - **notification**: event delivery
    - *notify(e)*

Introduction

- **Elements of a Pub-Sub System**



Subscription Models

- **Channel-based**
- **Topic-based**
- **Content-based**
- **Type-based**
- **Context/Location-aware**
- **Concept-based**
- ...and complex subscriptions on several events/conditions

Subscription Models

- **Channel-based**
  - primitive scheme
  - subscription to given communication channel
    - e.g., CORBA event service
  - all events sent (published) to channel are received (notified)

  - advantages: easy implementation
  - drawbacks: inflexible, no filtering, scalability issues

Subscription Models

- **Topic-based**
  - events group around topics
  - topic as a logical channel
  - coarse-grained
    - subscription implies all events with specific topic
  - e.g., Scribe, Bayeux, iBus...
  - advantages: leverage multicast, diffusion trees
  - drawbacks: limited expressiveness of the subscription language
  - solution: hierarchical topic space
    - topic subscription includes all of its sub-topics

Subscription Models

- **Content-based**
  - conditions (predicates) over event content
  - query filter
    - conjunction, disjunction, equality, comparison operators
    - regular expressions and more complex matching
  - fine-grained
    - correspondence between publisher and subscribers event by event basis
  - e.g., Gryphon, SIENA, JEDI,...
  - advantages: higher expressive power
  - drawbacks: higher resource consumption
    - calculating set of interested subscribers

Subscription Models

- **Type-based**
  - typed events, type hierarchy/graph
    - events belong to a specific type
    - encapsulated attributes and methods
  - middle-ground approach
    - coarse-grained as in topic-based
      - a type subscription is a channel
    - fine-grained as in content-based
      - constraints over attributes or methods
  - advantages:
    - type-safety at the pub-sub system not the application,
    - robustness

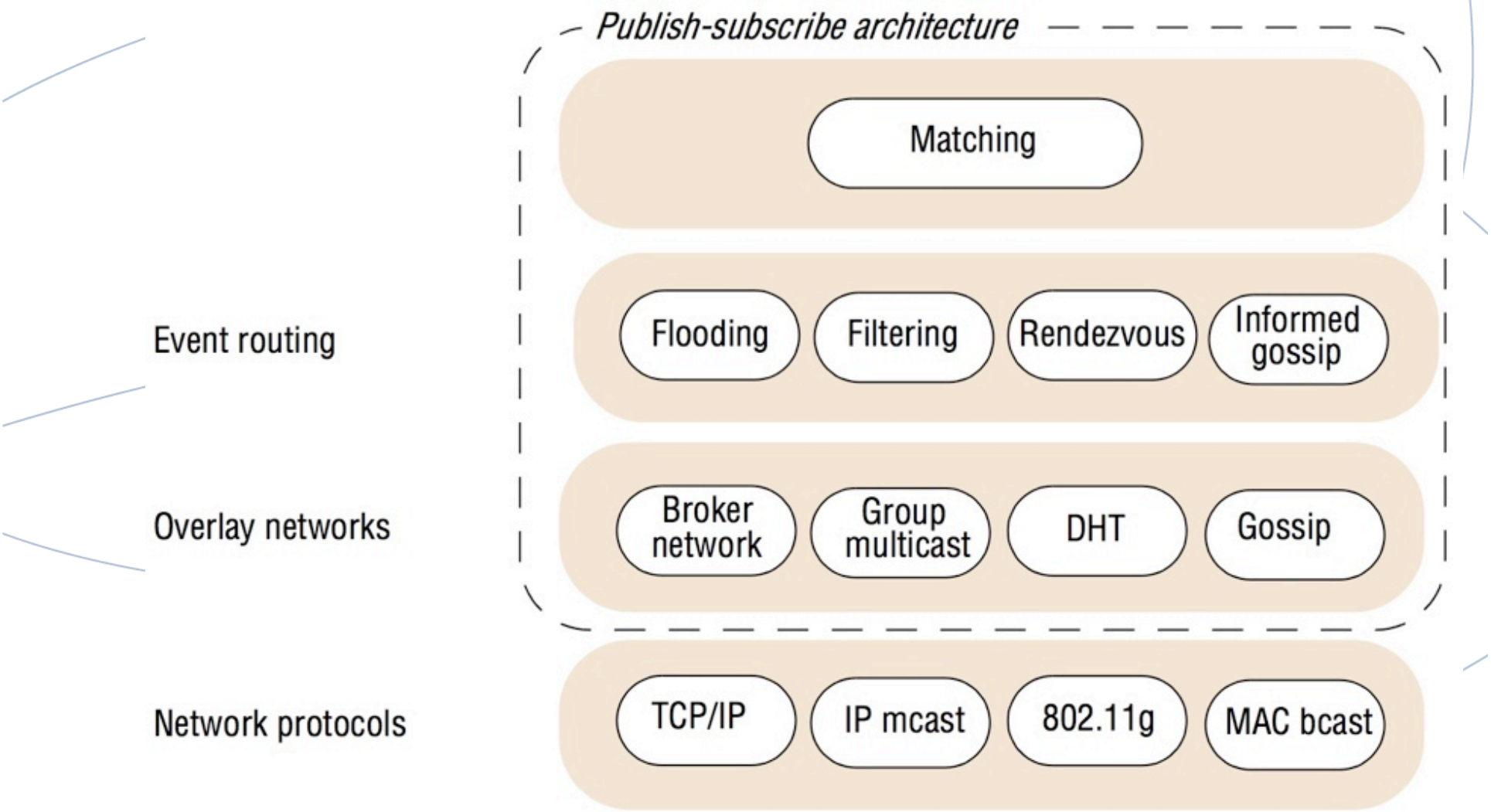
Subscription Models

- **Context/Location-Awareness**
  - suited to mobile environments
  - subscriptions based on/filtered by
    - user locations
    - vicinity of service
    - usually, close-by event publisher
  - drawbacks:
    - requires monitoring on user mobility

Subscription Models

- **Concept-based**
  - release from syntatical restrictions
    - number, name, type of attributes
  - release from semantical assumptions
    - (implied) meaning of each attribute
  - event schema using ontologies
    - Metadata, mapping functions, XML
  - advantages:
    - interoperability, extensibility, implementation-independence
  - drawbacks:
    - heavier XML-based subscription processing

Architectural Model



Architectural Model

- **Publish-Subscribe Functional Layers**
  - system
    - set of distributed nodes coordinated to dispatch events
    - to all (preferrably only) interested subscribers
  - overlay infrastruture
    - node organization, (de-)centralization aspects, dedicated servers,
    - groups in GCS can represent channels or topics
  - event routing
    - dispatching of events, exploit infrastructure, enhance with routing information to achieve scalbility
  - matching
    - testing events against subscriptions

Architectural Model

- **Overlay Infrastructure: Broker Overlay**
  - set of independent servers (brokers)
  - logically connected, not permanent links
  - each broker knows (some of) the others, (mostly) inherently static
    - topology changes rare
    - addition of new broker, failure handling
  - clients access system via any broker
- Hierarchical:
  - brokers as trees, subscribers at leaf nodes, publishers at root-nodes (or vice-versa)
- Flat: broker connected to any other brokers, less load on higher-nodes
- E.g., TIB/RV, Gryphon, SIENA, JEDI, REDS

Architectural Model

- **Overlay Infrastructure:Peer-to-Peer Structured Overlay (DHT)**
  - structured graph over virtual key space (key→node)
  - efficient discovery of data, granted correspondence
    - between any address (key) and an active node
  - advantages:
    - allows efficient unicast and multicast
    - better handling of dynamic aspects (joins, faults)
  - E.g., usage of P2P structured overlays
    - Pastry, Chord, Tapestry, CAN,
  - E.g., systems
    - Scribe, Bayeux (topic-based),
    - Hermes,Rebecca,Meghdoot (content-based)

Architectural Model

- **Overlay Infrastructure: Peer-to-Peer UnStructured Overlay (Gossip)**
  - absence of structure or direct mapping of keys
  - use flooding, gossiping, random walks for event difusion and information (subscriptions) retrieving
  - probabilistic in nature
  - fewer examples (mainly gossip-based)

Architectural Model

- **Matching**
  - checking events against subscriptions
  - performed on massive data sizes
  - routing algorithms need matching phase
    - distributed over nodes/brokers
    - avoid propagating all events or all subscriptions

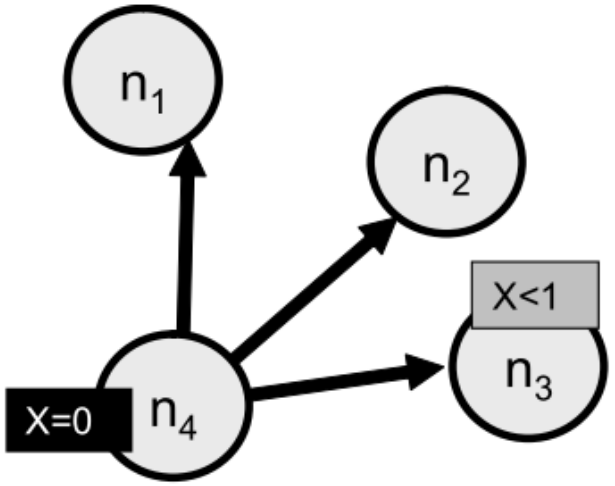
Architectural Model

- **Event Routing**
  - flooding algorithms: event, subscription
  - selective algorithms: rendez-vous, filter-based
    - reduce message and/or memory overhead
  - event gossiping: basic, informed gossip
  - issues:
    - message overhead, memory overhead, subscription language limitations,subscription changes, joins, leaves, failures (churn)

		Routing	*Filtering*	*Nodes storing Subs*	*Nodes handling Events*
Flooding	Event flooding	Det.	Subscribers	None	All
	Subs Flooding	Det.	Publishers	All	None
Selective	Filter-Based	Det.	Intermediaries	Subset	Subset
	Rendezvous-based	Det.	Intermediaries	Subset	Subset
Gossiping	Basic gossiping	Prob.	Subscribers	None	All
	Informed gossiping	Prob.	Intermediaries	Subset	Subset

Event Routing

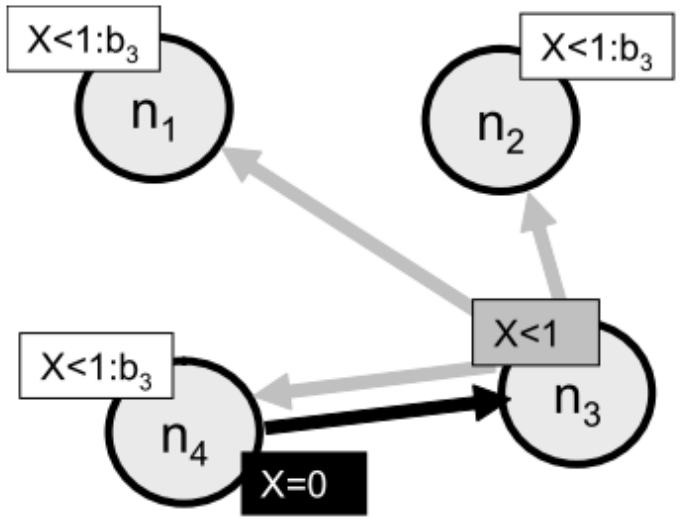
- **Event Flooding**
  - each event fully propagated
    - from publisher to all nodes in the system
  - subcriptions stored locally at subscribers
  - broadcast may be used
  - no message scalability
  - minimal memory overhead
  - no language limitations



(a) Event Flooding

Event Routing

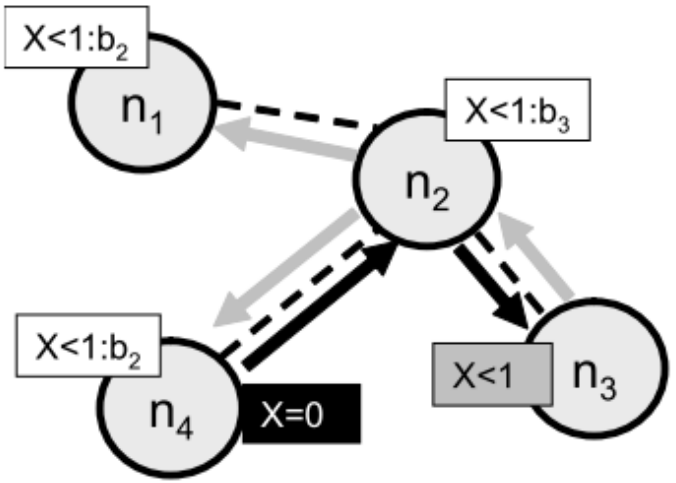
- **Subscription Flooding**
  - each subscription fully propagated
    - sent to all (publisher) nodes in the system
  - each node has complete knowledge of entire system
  - single-hop, optimal filtering
  - message overhead
    - subscription flood
  - memory overhead
    - all subscriptions
  - high coupling regarding
    - overlay membership



(b) Subscription Flooding

Event Routing

- **Filtering-based**
  - events forwarded only along
    - paths leading to interested subscribers
  - eliminate as soon as possible
    - events without subscribers
    - stop forwarding
  - diffusion paths
    - routing information at nodes
    - neighbour brokers and
      - reachable subscriptions
    - reverse paths followed by events published



(c) Filtering-based

Event Routing

- **Filtering-based**
  - reduced mesage overhead
  - optimizations: subscription containment
    - brokers modifiy routing
      - only when new subscriptions change the...
      - ...overall condition to be tested
  - natural architecture: brokers, tree or acyclic graph
    - more neighbors,
      - smaller network diameter
      - memory overhead increases
  - no subscription language limitations

Event Routing

- **Filtering-based**
  - data structures:
    - neighbor list, routing table, subscription list
  - routing table maps neighbors with subscription sets

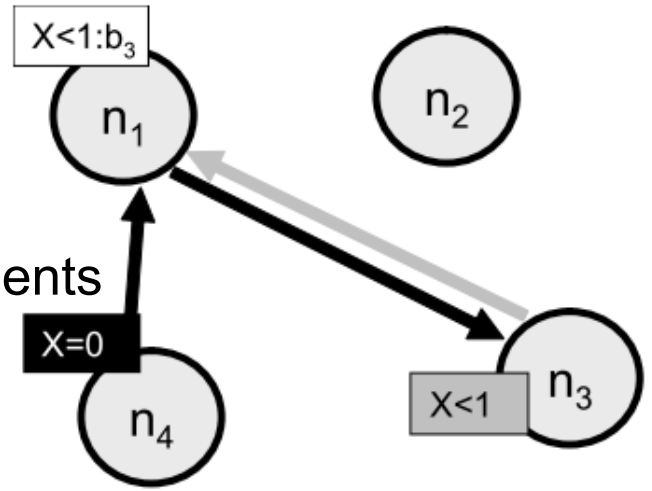
---

upon receive publish(event e) **from** node x
 $matchlist \leftarrow$ match(e,$subscriptions$);
 send notify(e) to $matchlist$;
 $fwdlist \leftarrow$ match(e,$routing$);
 send publish(e) to $fwdlist - x$;

upon receive subscribe(subscription s) **from** node x
 if x is client **then**
  add s to $subscriptions$;
 else add (x, s) to $routing$
 send s to $neighbors - x$;

Event Routing

- **Rendez-vous based**
  - mapping functions: SN, EN
  - SN(σ)→ set of rendez-voud nodes for subscription σ
    - store σ and forward matching events matching to all subscribers
  - EN(e)→ set of rendez-voud nodes for event
    - match event against subscriptions
  - two phases:
    - publisher sends events to EN(e)
    - EN(e) match e against subscriptions
    - forward to subscribers
  - EN(e) and SN(σ) must have common elements
    - *mapping intersection rule*
    - not trivial with content-based
    - clustering of subscription space



(d) Rendezvous-based

Event Routing

- **Rendez-vous based**
  - not suited to broker overlay
    - dynamism not efficiently handled
    - new node joins in, node leaving, crashing: similar problems
      - implies whole repartitioning among nodes
      - (many) subscriptions moved among nodes
  - unstructured P2P networks also not easily suited
  - highly-efficient on P2P structured overlays
    - mapping functions, fixed-size address space used as target
    - no need to know globally individual node in charge of each key
  - limitations: subscription language
    - mostly topic-based,
    - content-based: attribute mapping, numerical-only, limited
  - memory overhead, balanced subscriptions over keys

Event Routing

- **Rendez-vous based**
  - routing over neightbour lists
  - directed towards rendez-vous nodes

---

upon receive publish(event e) **from** node x **at** node i
 $rvlist \leftarrow EN(e)$;
 if $i \in rvlist$ **then**
  $matchlist \leftarrow$ match(e,$subscriptions$);
  send notify(e) to $matchlist$;
 else
 send(e) to $rvlist$;
upon receive subscribe(subscription s) **from** node x **at** node i
 $rvlist \leftarrow SN(s)$;
 if $i \in rvlist$ **then**
  add s to $subscriptions$;
 else
  send(s) to $rvlist$;

Event Routing

- **Gossip-based**
  - random choice of nodes
  - driven by local information
    - acquired during execution
    - fine-tuning routing tables
  - aggregation of subscriptions
  - organized into groups (super-peers)
    - delegate actions on other group leaders
    - outside vicinity
  - fully distributed, resilient to churn
  - fully probabilistic

Event Routing

- **Guaranteed Delivery**
  - non-zero delay between
    - subscription changes
    - event routing data capturing changes
  - problems
    - event loss
    - unnecessary event forwarding
  - solutions
    - resend lost events
    - keep separate streams for failed subscribers
    - store events with persistence

Surveying Publish/Subscribe Systems

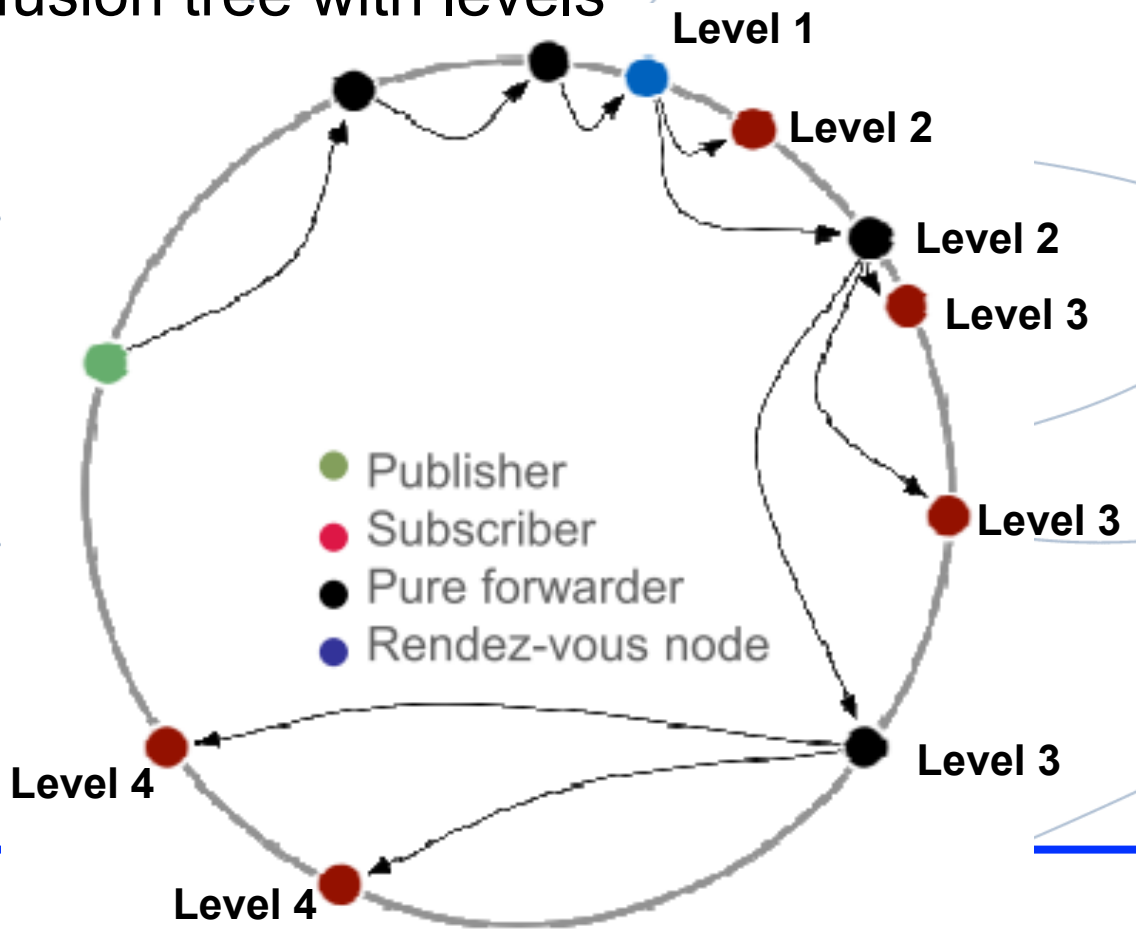
System (and further reading)	*Subscription model*	*Distribution model*	*Event routing*
CORBA Event Service (Chapter 8)	Channel-based	Centralized	-
TIB Rendezvouz [Oki *et al.* 1993]	Topic-based	Distributed	Ffiltering
Scribe [Castro *et al.* 2002b]	Topic-based	Peer-to-peer (DHT)	Rendezvous
TERA [Baldoni *et al.* 2007]	Topic-based	Peer-to-peer	Informed gossip
Siena [Carzaniga *et al.* 2001]	Content-based	Distributed	Filtering
Gryphon [www.research.ibm.com]	Content-based	Distributed	Filtering
Hermes [Pietzuch and Bacon 2002]	Topic- and content-based	Distributed	Rendezvous and filtering
MEDYM [Cao and Singh 2005]	Content-based	Distributed	Flooding
Meghdoot [Gupta *et al.* 2004]	Content-based	Peer-to-peer	Rendezvous
Structure-less CBR [Baldoni *et al.* 2005]	Content-based	Peer-to-peer	Informed gossip

Pub-Sub Systems

- **Scribe**
  - topic-based routing over structured overlay (Pastry)
    - each broker has unique ID with efficient routing to it
  - event routing: rendez-vous over structured overlay
    - each topic assigned unique ID (hashing)
    - broker of topic is the node with ID closest to topic's
    - multicast tree for each topic (broker as root)
    - EN(e) = h(e), SN(s) = h(S), i.e., EN=SN for a given topic
  - subscriptions: routed to broker
    - updating tree structure to include new subscriber
    - higher nodes in tree seldom changed
  - publishing event routed to broker directly
    - broker starts diffusion of notification through the topic tree
    - until all leaf nodes (subscribers) are reached

Pub-Sub Systems

- **Scribe**
  - topic rendez-vous node
  - event diffusion tree with levels

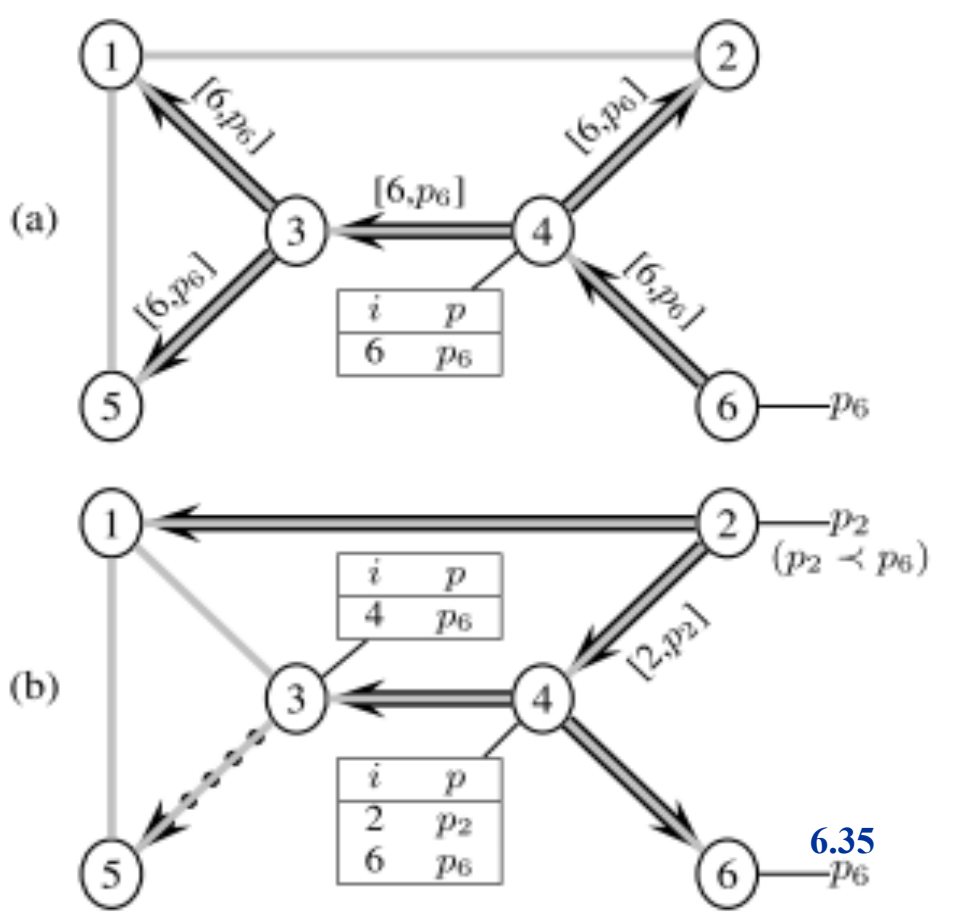


Pub-Sub Systems

- **SIENA**
  - uses broker network for routing over TCP or UDP layer
  - filtering-based content-based routing (influential work)
    - for subcription propagation and event delivery
  - optimizations:
    - subscription covering/containment
      - s1 contains s2 iff all events matching s2 also match s1
      - subscription updates only forwarded
        - if broker has not already propagated a containing subscription
    - subscription refreshing protocol triggered by brokers
    - advertisements (publisher advertisements)
      - publishers declare set of events they are going to produce
      - considered in routing paths, futher reduce set of involved brokers

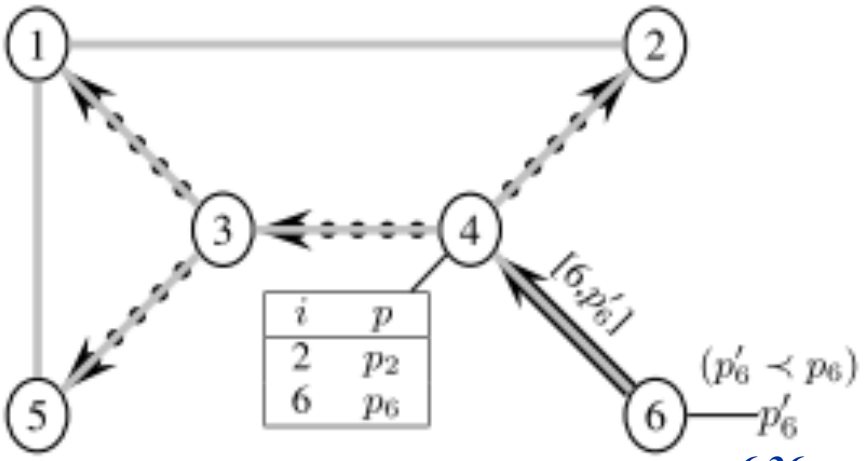
Pub-Sub Systems

- **SIENA**
  - subscription propagation
    - Receiver Advertisement Protocol (RA)
  - subscription containment
    - (subscription covering)
    - p6 covers p2
    - no need to propagate s2
      - over to b3 and beyond



Pub-Sub Systems

- **SIENA**
  - subscription/adress inflation
  - when updated subscriptions by one client
    - are covered by its early subscriptions
    - no forwarding takes place
  - in the long run
    - increasing number of events
    - wastefully forwarded
    - via broker routes directing to
    - no-longer subscribing nodes

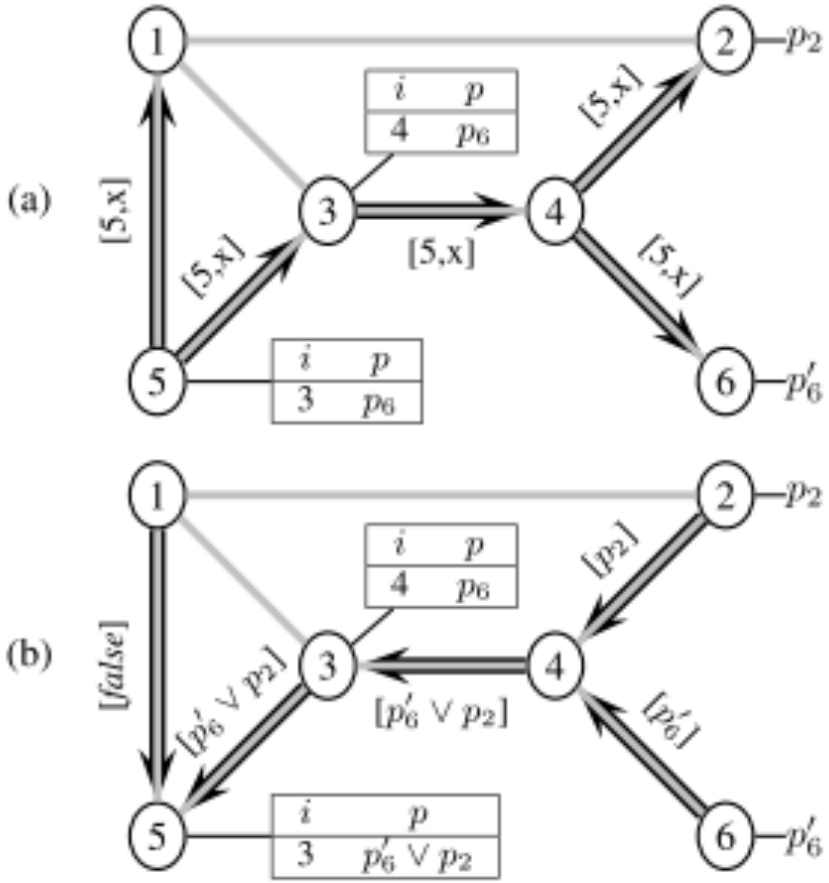


Pub-Sub Systems

- **SIENA**
  - Sender Request / Update Replies (SR/UR) protocol
    - periodical subscription update
    - triggered by broker nodes to *pull* subscription information
    - tackles address inflation
  - receiving brokers:
    - leaf-broker:
      - replies immediately with subscriptions
    - non-leaf:
      - assembles UR
      - combining its subscriptions with
      - those received from downstream
      - waits until all replies or timeout and replies upstream

Pub-Sub Systems

- **SIENA**
  - Sender Request / Update Replies (SR/UR) protocol



Summary

- **Introduction**
- **Elements in Pub-Sub**
- **Subcription Models**
- **Architecture Model**
  - transport, infrastructure, routing, matching
- **Distributed Event Routing**
  - flooding, selective (filtering/rendez-vous), gossip,...
- **Example Systems**
  - Scribe, Siena, ...

Pub-Sub Systems

- **SIENA**
  - Sender Request / Update Replies (SR/UR) protocol
  - Opportunistic Update Reply Processing
    - allows caching of UR replies to previous SR
    - brokers may reuse URs to SRs sent by other brokers **iff**
      - set of downstream nodes from caching broker...
      - equal to downstream nodes for initial requester...
      - at the same link
    - e.g., b4 caches UR from b6
    - reuses as reply to SR from b2
      - nodes from b5 to b6 via b4
      - same nodes as
      - from b2 via b4 (i.e., b6)

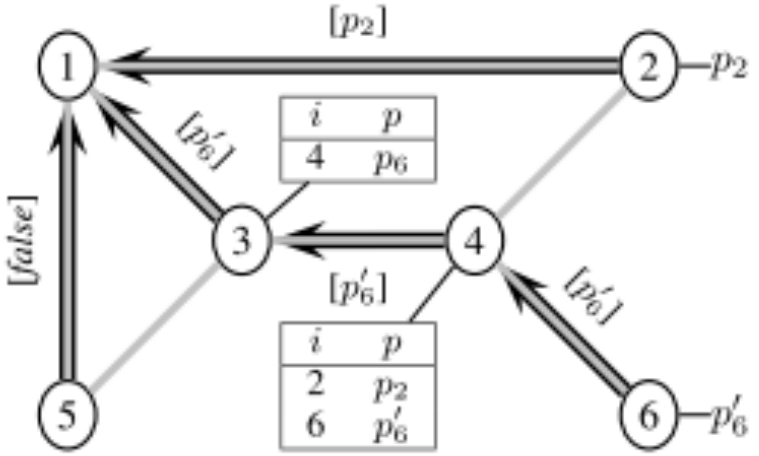


Fig. 9. Opportunistic UR Processing