Publish-Subscribe Systems

- What is publish-subscribe
- Data model
- Subscription classification
  channel, topic, content, type - based
- Composite events
- Case Studies: Siena, SMC and Hermes
- Large-scale multibroker systems
- Event Routing
- Pub/Sub issues.

Acknowledgements

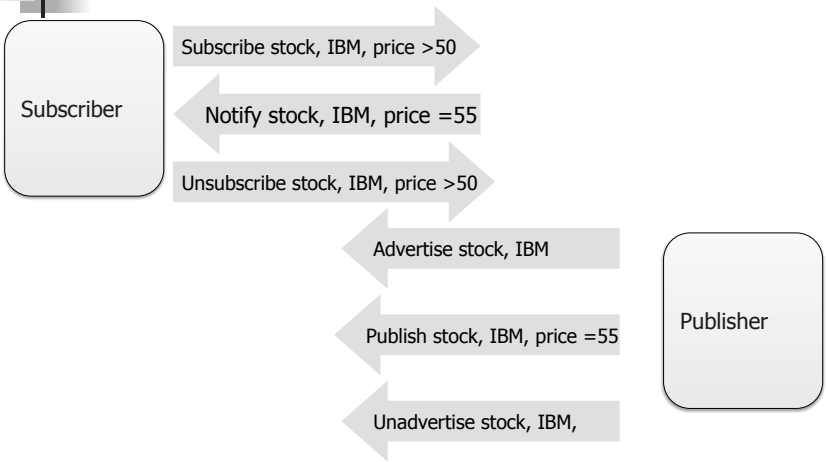
The slides are derived from tutorials by:

- Prof. Peter Triantafillou, University of Patras
- Antonio Carzaniga and Prof Alex Wolf

Characteristics of a Pub/Sub System

- **Information consumers** express their interests in information with ***subscriptions,*** identifying which items are of interest***.***
- **Information producers**, ***publish*** information by submitting publications (a.k.a. ***publication events*** or ***event notifications***).
- Information producers may ***advertise*** information about events they publish
- A pub/sub system:
  - Stores (and indexes appropriately) subscriptions
  - Processes events
  - Matches each produced event to the set of appropriate subscriptions.

Event Notification Protocol



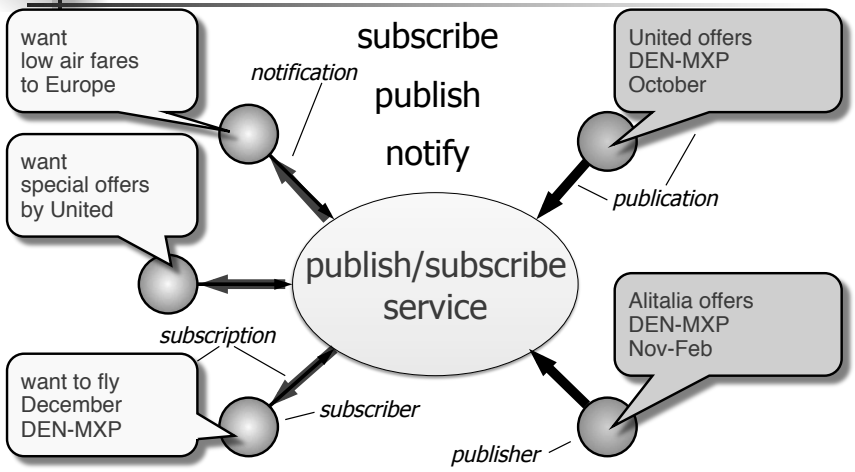
Database View of Pub/Sub

- Events correspond to **data** ("data-carrying events").
- Subscriptions correspond to **queries**:
  - ***Continuous***
  - Define predicates on attributes
- Fundamentally different model:
  - Instead of storing/indexing data and issuing queries to access it
  - Queries (subscriptions) are stored/indexed and incoming data (events) is matched against stored queries.
- The pub/sub system performs ***information filtering,***
  - Events are filtered (evaluating the predicates of stored subscriptions)
  - Filtered events are delivered to interested subscribers.

Pub/Sub Communication Model

- Akin to ***multicasting*** i.e. , 1-N communication
  - Events may be published to a large number of subscribers.
- ***Anonymous communication***
  - Subscribers do not name ("know") publishers and vice versa
  - Destination determined by receiver not sender
- ***Asynchronous communication***
  - Publish and subscribe do not block clients
- Decoupling of sender/receiver
  - Flexible Adaptation
  - Can easily add new publishers or subscribers – scalability
  - Reactive – computation driven receipt of message c.f. policies
  - Facilitates interaction between large numbers of entities

Publish/Subscribe Middleware Service



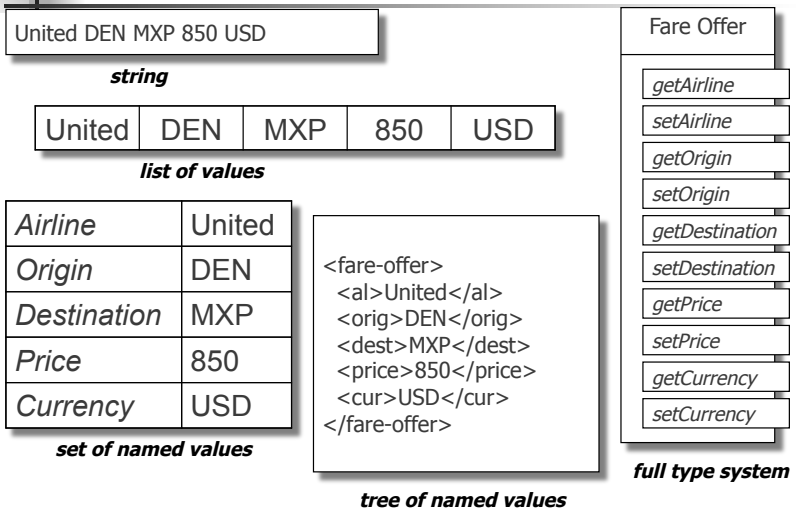
Applications

- Obligation Policy Triggers
  - Management systems, context changes in ubiquitous systems.
- Broadcasting of live data
  - RSS feeds (news reports, stock exchanges, …)
  - Satellite data, etc.
- Monitoring
  - The web: issue subscriptions for page updates, etc.
  - Network & system management – alarms etc.
  - Sensor networks
- Social Networking
  - Sharing events with friends
  - Twitter
- Multi-player Games

Basic Elements of a Pub/Sub Service Model

- Data model – message structure
  - structure
  - types
- Subscription model
  - filter language
  - scope/expressiveness – channel, topic, content, type based
  - subscription modifications
- General challenge
  - expressiveness versus scalability

Data Model



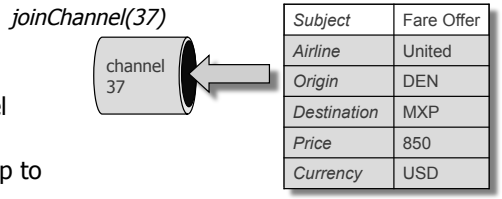
Subscription Model: Filter Language

- Operators are implied by the data model

string	prefix, suffix, substring, regular expression
list of values	value operators + indexing e.g. msg [2] = DEN, msg[5] = USD
named values	value operators + selection e.g. msg.Origin = DEN, msg.Currency = USD
tree	value operators + structure walk
full type	user-defined operators

Channel Subscriptions

- Channel
  - channel identifier
  - equality over channel identifier
  - channel ID could map to topics

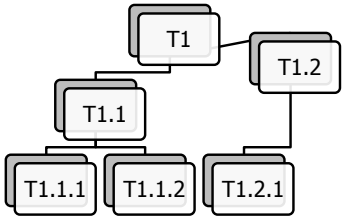


Topic-based Subscriptions

- Subject (or topic)
  - Designated subject field – descriptive name or keyword is minimum addressable info unit
  - General filter applied to designated field i.e. whole topic c.f. news groups
  - Topics can be hierarchically organised
  - Publications refer to a topic in the tree
  - Client subscribing for T1.1, gets all events 'naming' T1.1 including T1.1.1 & T.1.1.2

Subject	Fare Offer
Airline	United
Origin	DEN
Destination	MXP
Price	850
Currency	USD

subscribe("Fare")*



Content-based Subscriptions

- Content
  - All message fields
  - General filter (predicate) applied to all fields in message
  - Allow much finer granularity in addressing info units and thus much greater flexibility.
  - Subscriptions can be very 'expressive'

Subject	Fare Offer
Airline	United
Origin	DEN
Destination	MXP
Price	850
Currency	USD

subscribe("Origin=DEN, Price<900")

- An **atomic subscription** is a conjunction of a set of predicates on a number of attributes in a single event
- A **composite subscription** is a boolean combination of *atomic subscriptions* i.e. combining multiple events

Content-based Subscriber outline

```
public class FlightReq implements Serializable {
    public string destination, airline;
    public float price
}
public class FlightSubscriber implements Subscriber{
    public void notify (Event e) {
      bookFlight()
}}

string criteria = (airline == "United" AND price < 900 AND price > 300);
Subscriber sub = new FlightSubscriber();
EventService.subscribe(sub,criteria);
```


Type-Based Subscription

- Use Type-based language for defining events
- Clients can subscribe to different event types
- Event types can also be hierarchically organized in terms of inheritance
  - A subscription refers to a subtree root
  - Events referring to any subtype in that subtree are delivered to the subscription
    e.g. *stockQuote* and *stockRequests* are subtypes of *stock*. Subscribing to *stock* gets both types of events.
- Content-based filtering on event type attributes
- Type-based systems offer benefits for programming the system:
  - Type safety
  - Encapsulation

Composite Event Subscriptions

- Event correlation ➔ more abstract events
- Combine atomic events using composition operators:

a & b	occurs when both a and b occur, irrespective of order
(a ; b) ! c	a followed by b with no interleaving c
a ; b	a followed by b (only composite in Siena)
a | b	a or b occurs
a = b	a and b occur simultaneously

Example: If a person has entered a building and has not left within 2 minutes of a fire alarm, generate an *indanger* event for the person

entered (name) ; (fire + [2*min]) ! left (name)
when (entered.name == left.name))
notify.indanger (entered.name)

Composite Event Issues

- Used in Active Databases
- Complex implementation:
  - Time window in which to match events
  - What is simultaneous?
  - Need time synchronization for distributed composite event matching
  - Storing and then discarding events
- Sometimes implemented as a client which subscribes to required atomic events and publishes composite events.
- Supported by Hermes
  Composite Event Detection as a Generic Middleware Extension
  http://www.doc.ic.ac.uk/%7Epeter/manager/doc/IEEE_Network_Composite_Event_2004.pdf
- GEM: a generalized event monitoring language for distributed systems
  http://www.iop.org/EJ/article/0967-1846/4/2/004/ds7204.pdf

Example: Siena Data Model

- An event is a set of typed attributes

attribute ≡ (type, name, value)

```
string       class = travel/airlines/offers
date         starts = Jun
date         expires = Aug
string       origin = DEN
string       destination = MXP
string       carrier = United
float        price = 850
string       currency = USD
```


Example: Siena Filter Language

- A filter is a list of attribute constraints

attribute constraint ≡ (type,name,operator,value)

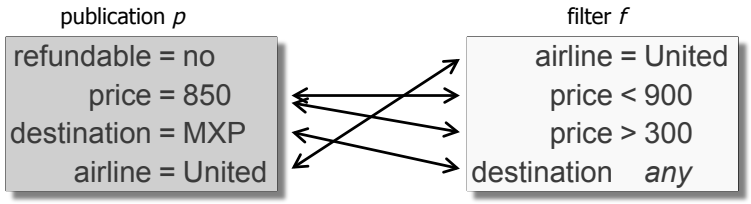
Prefix match

```
string       class >* travel/airlines/
date         starts <  Jul
date         expires >  Jul
string       origin =  DEN
string       destination = MXP
```

Example: Siena Matching Rules

- Matching is defined by covering relations

p matches f (or *f covers p*)



Siena Publisher

HelloWorld.java

```
import siena.*;

public class HelloWorld {
  public static void main(String [] args)
  throws SienaException {
    Siena siena = new ThinClient("tcp:host.domain.net:1234");
    Notification n = new Notification();
    n.putAttribute("greeting", "Hello World!");
    siena.publish(n);
  }
}
```


Siena Subscriber

Subscriber.java

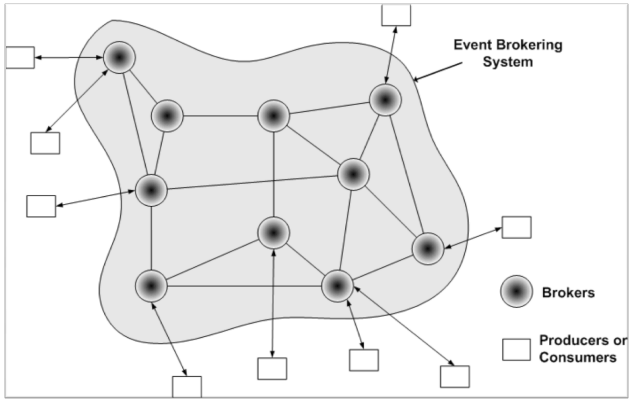
```
import siena.*;
public class Subscriber implements Notifiable {
  public void notify(Notification n) {
    System.out.println(n);
  }
  public static void main(String [] args)
  throws SienaException {
    Siena siena = new ThinClient(args[0]);
    Filter f = new Filter();
    f.addConstraint("greeting", >*, "Hello");
    siena.subscribe(f, new Subscriber());
  }
}
```

Prefix

Pub/Sub Processing Model

- Publisher is a client which advertises and publishes events
- Subscriber is a client which subscribes to events and receives matched notifications
- An event ***e*** is said to ***match*** a subscription ***s***
  - if and only if all attribute-value predicates in s are satisfied by the values carried by e.
- ***Brokers*** are servers in the infrastructure which store subscriptions, receive published events, match these to subscriptions and deliver matched notifications to subscribers.
- Matching entails ***3 phases***:
  - ***Subscription processing***: Indexing and storing subscriptions.
  - ***Event processing***: upon event arrival, access subscription indices and identify all matched subscriptions.
  - ***Event delivery***: deliver event to clients with matched subscriptions.
- Large-scale systems have a **network of brokers** which have to propagate subscriptions and events around the network

Network of Brokers



- Each broker has a local event matching engine

Large-scale Systems

- Subscriptions must be propagated through the network and stored at appropriate brokers.
- Events must be
  - Propagated through the network to 'meet' all relevant subscriptions
  - Delivered to the clients that issued the matched subscriptions.
- Objectives
  - Route only 1 event along common paths – minimize duplication
  - Minimize propagation and storage of subscriptions – generalized subscription covers specialized
    e.g. Price < 500 covers Price < 400
- How these are accomplished efficiently is the major challenge facing large-scale pub/sub systems.

Performance Trade-offs

- Favor event processing against subscription processing, or vice versa
- Two ends of the design spectrum:

- ***Event-friendly*** approach:
  - Bcast a subscription to all broker nodes;
  - Each broker stores subscription
  - Events sent to just one broker
- ***Subscription-friendly*** approach:
  - Send a subscription to just one broker node, where it is stored.
  - Bcast events to all brokers.

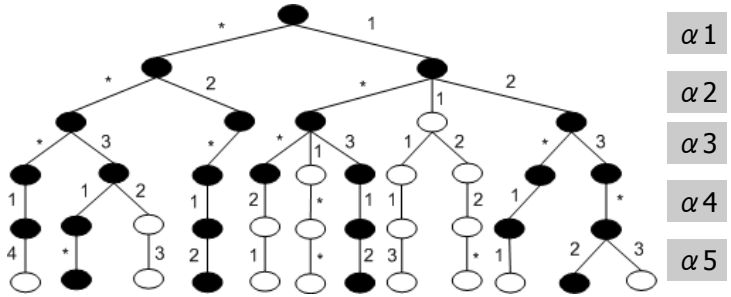
Flooding subscriptions or events should be avoided

Gryphon matching process

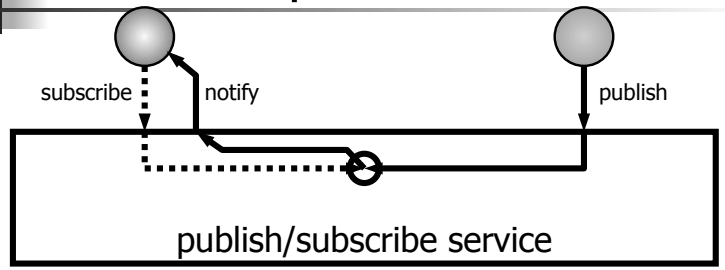
- A (parallel) tree is employed to index subscriptions.
- The tree has a number of levels equal to the number of attributes in the schema.
- Subscriptions build the tree, level by level, defining new branches, based on the values for their attributes.
- Subscriptions are stored at the tree leaves.
- Events traverse the tree, level by level, following the branches defined by event values and match all subscriptions at the leaves.

Gryphon Example

- Example system with a 5-attribute schema.
- Subscription < α1=1, α2=2, α3=3, α5=3> follows rightmost path.
- Event < α1=1, α2=2, α3=3, α4=1, α5=2> visits all dark nodes.

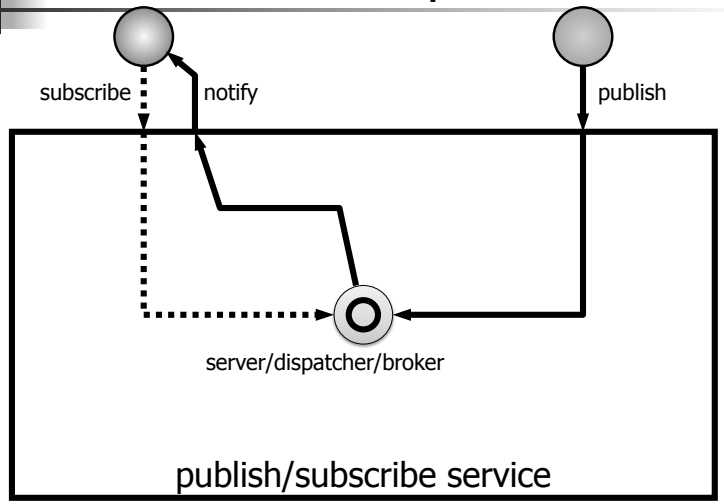


General Service Implementation



- Publications must meet subscriptions somewhere
- Service must decide whether publications match subscriptions
- Service must notify subscribers

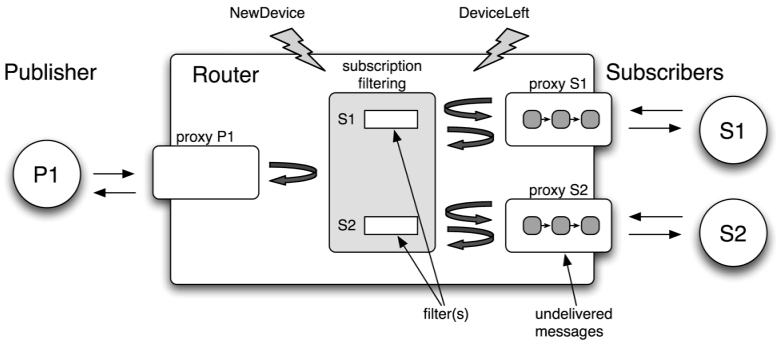
Centralized Implementation



Centralized SMC Event Service

- Content-based subscriptions with single router
- At-most-once, reliable event delivery.
- Caters for intermittent loss of wireless comms.
- To an individual recipient, events are delivered in the same order as received by the router.
- Quenchable publishers to minimise number of messages and power consumption i.e. do not publish if no active subscribers
- Supports heterogeneous communication.
- Implemented on small smartphone-like devices

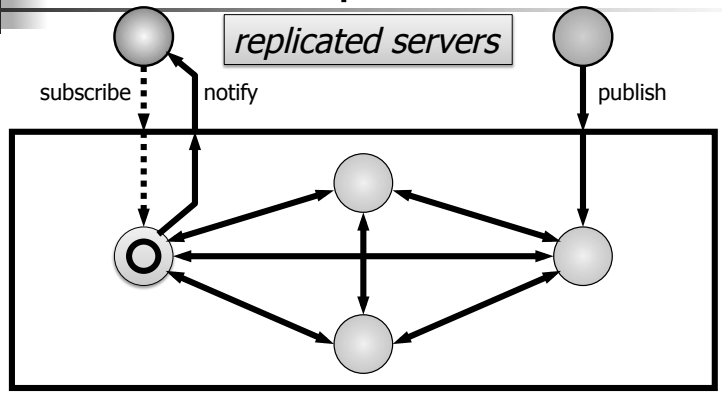
SMC Event Service Architecture



Evaluating Centralized Implementation

- ✓ Simple
  - every (client) application sends subscriptions and publications to a single server
  - the server maintains and evaluates all subscriptions locally
- × But not very "scalable"
  - one server handles every subscription and every message of every client
  - the server becomes a network bottleneck
  - all clients must trust the same server (privacy)

Naïve Extension: Broadcast Replication

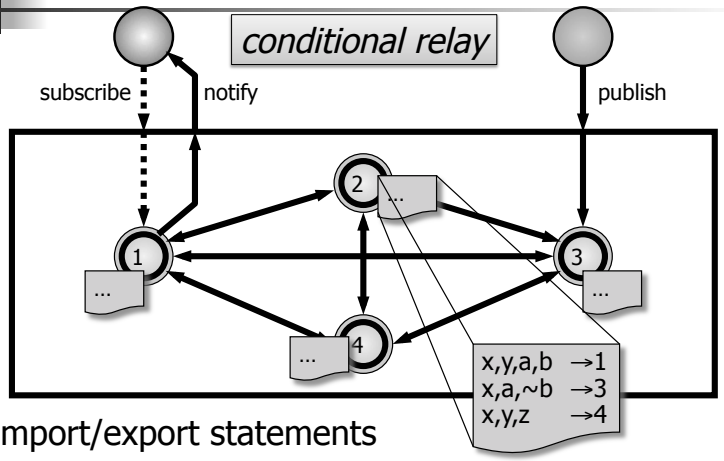


- Publications are relayed to every server
- Subscriptions remain local to their servers

Evaluating Broadcast Replication

- ✓ Simple
  - hopefully making use of network-level broadcast
- ✓ Reduces computation
  - only local subscriptions
  - same number of publications
- × Still not very scalable
  - little flexibility
  - little privacy
  - increased network traffic
- What publish/subscribe patterns might this suit?

Another Simple Extension: Federation

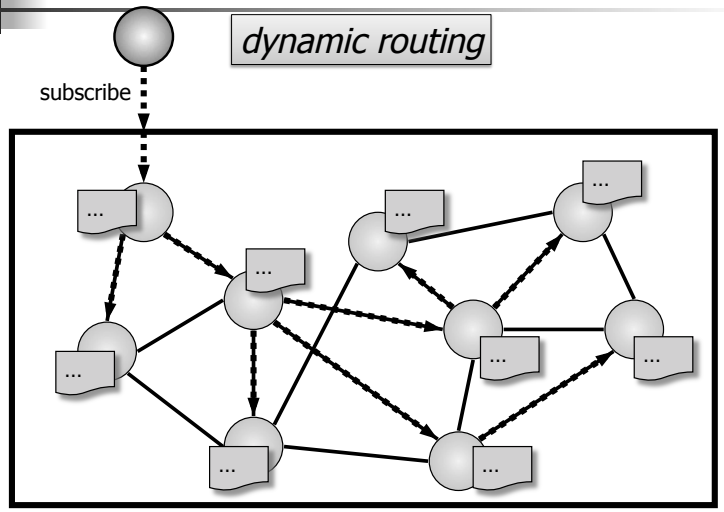


- Import/export statements
- Static, user-defined routes

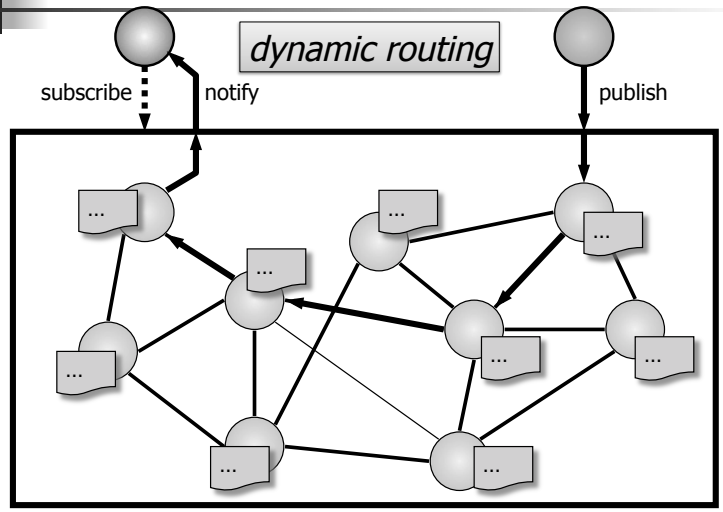
Evaluating Federation

- ✓ Reduces traffic
  - only imported/exported traffic goes through
- ✓ Improves privacy
- ✓ Reduces computation
  - local subscriptions plus import/export
- × Still not very scalable
  - little flexibility
  - suboptimal network usage
  - potential errors due to misconfiguration

Advanced: Fully Distributed Architecture



Advanced: Fully Distributed Architecture



Dynamic Routing

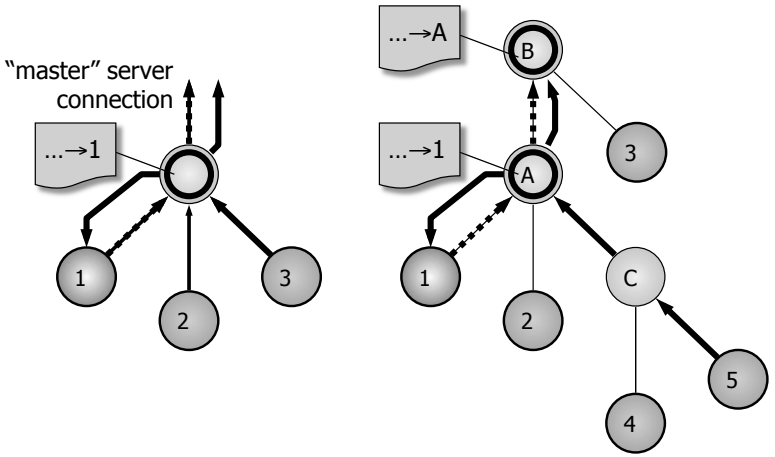
- Servers are interconnected through statically configured links
- Servers function as store-and-forward publication dispatchers
- Servers exchange subscription information according to a *routing protocol*
- Servers forward publications according to
  - subscription information
  - forwarding protocol

Evaluating Fully Distributed Architecture

- ✓ Increases reliability
  - exploiting multi-connected topologies
- ✓ Adapts automatically
  - to changes in applications (subscriptions)
  - to changes in network (topology)
- ✓ Reduces computation and communication
  - using optimized routing and forwarding
- × Requires complex protocols
  - to handle topological routing information
  - to handle content-based routing information

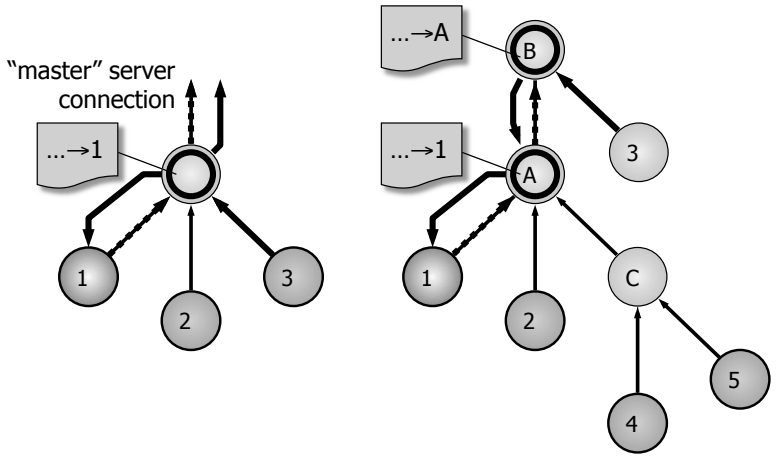
Example: Hierarchical Routing

simple extension of centralized server



Example: Hierarchical Routing

simple extension of centralized server

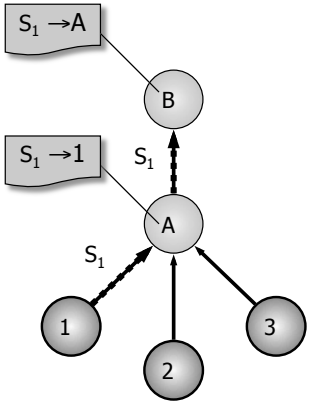


Evaluating Hierarchical Routing

- ✓ Reduces traffic and computation
  - "leaf" nodes process only local traffic and traffic that matches local subscriptions
    - no unnecessary traffic goes down the hierarchy
- ✗ Still not very scalable
  - root server processes every subscription and publication

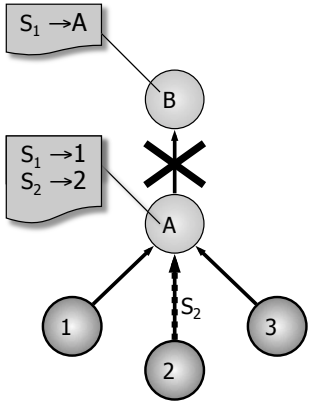
Hierarchical Routing Improved

combine and simplify subscriptions



Hierarchical Routing Improved

combine and simplify subscriptions

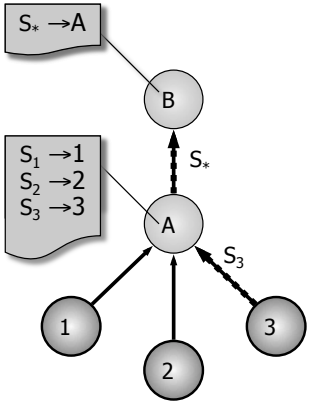


S_1 *covers* S_2

S_1=[airline="UA"]
S_2=[airline="UA", dest="MXP"]
⇒ do not propagate S_2

Hierarchical Routing Improved

combine and simplify subscriptions



S_1 *covers* S_2

S_1=[airline="UA"]
S_2=[airline="UA", dest="MXP"]
⇒ do not propagate S_2

S_1 ,S_2, and S_3 are *similar*

S_1=[airline="UA"]
S_2=[airline="UA", dest="MXP"]
S_3=[airline="AZ"]
⇒ propagate S_*=[airline=*any*]

Evaluating Improved Hierarchical Routing

- ✓ Reduce traffic and computation
  - only "new" subscriptions are propagated
  - servers store and process fewer, more generic subscriptions
- ✓ Open to a variety of heuristic optimizations
  - e.g., based on actual traffic profiles
- ✓ Applies to other architectures
  - concept of "content-based subnet address"

Siena System Fundamentals

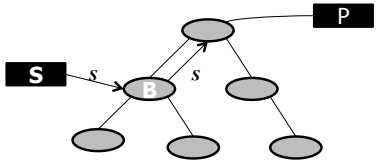
- Subscriptions are propagated in the network
- Whenever a broker ***B1*** forwards a subscription ***s*** to ***B2***, ***B2*** stores ***s*** and notes that ***s*** came from ***B1***.
- Whenever ***B2*** receives an event ***e*** that matches ***s***, it forwards ***e*** to ***B1***.
- Thus, ***event propagations follow the reverse paths setup by subscription propagations***.
- This basic approach, however, may still incur high overheads.

SIENA Improvements

- To improve performance, SIENA incorporated two influential concepts:
  - Publication ***advertisements***, and
  - Subscription ***subsumption.***
- ***Publication advertisements***
  - Describe events that will be published
  - Exploit the fact that (sometimes) publishers are aware of their publications.
  - The key idea is to
    - push this information into the network,
    - and exploit it during subscription propagation to place subscriptions only on the relevant nodes where events will be published

Siena Publication Advertisements

- Advertisements are propagated to form an ***advertisement tree,*** rooted at the publisher's broker.
- Whenever a subscription arrives at a node that is a member of the tree, it is sent 'up' the tree.
- Events from the publisher are sent 'down' the tree, following the reverse tree paths setup by matching subscriptions.
- A node may partake in many advertisement trees, and
- Some popular publications may be published from many different nodes



A publisher's P advertisement tree

Siena Subscription Subsumption

- Subscription subsumption (a.k.a. coverage) buys additional performance savings.
  - e.g. S1 subsumes S2 as Price 50-60 includes 52-55

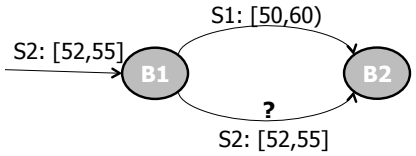
S1

Category	String	"Book"
Title	String	"Weird Oz"
Price	Float	[50, 60]

S2

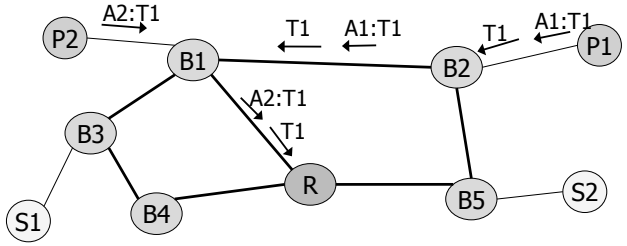
Category	String	"Book"
Title	String	"Weird Oz"
Price	Float	[52,55]

- Upon arrival of subscription S2 at B1, If B1 has already forwarded a subsuming subscription S1 to B2, then no need to forward S2 to B2 as well.
- Maintenance problems for changing or deleting subscriptions



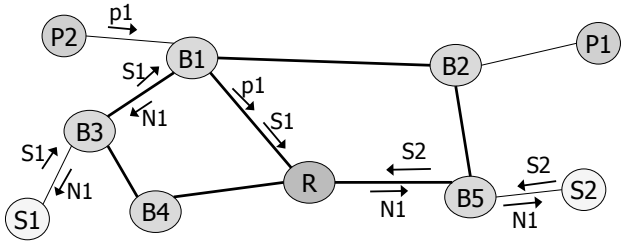
Hermes Rendezvous Nodes

- Publisher P1 hashes event type to select Rendezvous node R and registers type T1
- P2 advertises event A2 of type T1, so B1 sends it to R
- P1 advertise event A1 of type T1, which is covered by A2 so discarded by B1



Hermes Event Routing

- Subscriber S1 generated subscription S1 of Type T1 so it is routed to R
- S2 generates subscription S2 of type T1 which is also routed to R
- P2 generates publication p1 of Type T1 which follows path of adverts to R but also results in notifications N1 traversing reverse paths of matching subscriptions.
- Can have multiple event dissemination trees



Flooding vs Rendezvous

- With rendezvous, in general, fewer network nodes can be involved in subscription and event processing
  - Fewer messages, less bandwidth, less aggregate CPU cycles, less state, etc.
  - Possible load imbalances due to non-uniform data item popularity distributions.
- Rendezvous may require specific network support and characteristics, such as DHT lookup in order able to route towards a specific rendezvous node.

Pub/Sub Issues

- Hierarchical architectures for the broker network
  → avoid flooding; may incur only O(tree_depth) hops for subscription and some event processing.
- Hierarchical organizations for brokers per publisher
  → avoid single, heavily-loaded brokers for all traffic (e.g. root).
- Store extra routing state at brokers – paths setup by subscriptions and followed by events as in dynamic routing
- Further filtering of subscription traffic through exploitation of subsumption relationships among subscriptions.

Implementation Issues

- Extra state maintained at nodes
  - Additional hierarchies (parent/children)
  - Subscriptions and where they were forwarded from
  - Subsumed subscriptions and their relation

  Does not come for free !
- Nodes fail/recover: need to maintain
  - Hierarchical information
  - Subscription paths
- Subscriptions are updated and deleted
  - Subsumption relation and information needs maintenance.
- Popular events, with many publishers, even with advertisements and subscription-subsumption exploitation, can still lead to flooding-like performance.

Topic- vs Content-based

- Topic-based systems:
  - Simplicity, ease of programming and deployment,
  - High performance for event matching and delivery.
- Content-based systems:
  - Flexibility and expressiveness, when defining subscriptions
  - Increased complexity
  - Lower performance.

Attribute Type Support

- For equality operators, in subscription predicates, it is trivial to support both numerical and string types.
- If subscriptions (contents) are spread/replicated throughout the network (c.f. flooding-like approaches), it is also easy to support all attribute types.
- If rendezvous and/or network-independent approaches are used, it is more challenging to support advanced operators (such as ranges, >, <, etc for numerical and also prefix / suffix, / substring operators for strings.
  - How does one define a rendezvous node for "*abc*" ?

Available Pub/Sub Systems

- XML Blaster: Open Source XML event encoding with XPath expression subscription http://www.xmlblaster.org/
- IBM Microbroker MQTT Small footprint for sensor networks http://www.alphaworks.ibm.com/tech/rsmb
- Elvin Simple content based system http://www.elvin.org/
- Siena http://www.inf.unisi.ch/carzaniga/siena/software/index.html

References

- Online survey http://www.medianet.kent.edu/surveys/IAD04F-pubsubnet-shennaaz/Survey2.html#submodel
- P. Eugster et. al. The many faces of publish/subscribe, ACM Computing Surveys, Vol. 35, No. 2, June 2003, pp. 114–131.
- Siena: Design and evaluation of a wide-area event notification service, ACM Trans. on Computer Systems (TOCS), Vol19, No.3, Aug. 2001, PP 332-383 http://www.inf.unisi.ch/carzaniga/siena/
- Peter Pietzuch publications