Community-Aware Information Dissemination

EPFL

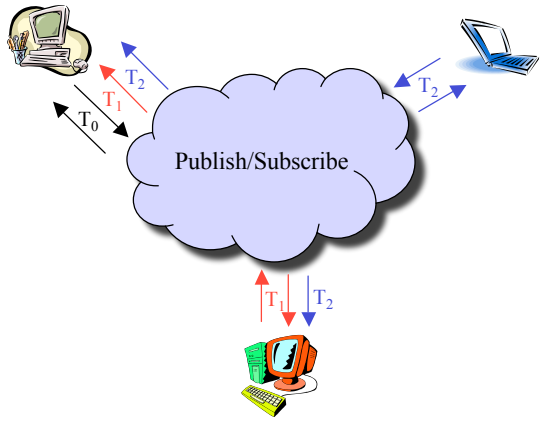
Distributed Programming Laboratory

Sébastien Baehni, Patrick Th. Eugster, Rachid Guerraoui, Oana Jurca

In *Proceedings of the 5th IEEE International Conference on Dependable Systems and Networks*, pages 233-242, June 2004 (original name: Data-Aware Multicast)

Motivation

Implement reliably the *type-based publish/subscribe* abstraction in a *decentralized* environment



Type hierarchy

T_0
T_1
T_2

Contribution

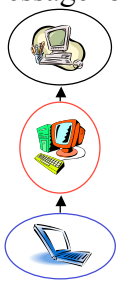
- *CAMCAST*
  - Exploits the *type hierarchy*
  - Memory complexity: $\ln(S_{T_i}) + c_{T_i} + z_{T_i}$
  - Message complexity: $S_{T_{max}} \ln(S_{T_{max}})$
  - Decentralized (no *broker*)
  - No *parasite* event
  - Tradeoff between reliability and message load

S_{T_i} = # processes in T_i

c_{T_i}, z_{T_i} cste

Type hierarchy

T_0
T_1
T_2



Roadmap

- ***Background***
  - Publish/Subscribe
  - Type-based Publish/Subscribe
  - Decentralized Systems
  - Related Work
- CAMCAST
  - Intuition
  - Evaluation

Background

- The problem specification relies on the knowledge of:
  - What is a *publish/subscribe* system?
  - What is a *type-based publish/subscribe* system?
  - What is a (de-)centralized system?

Roadmap

- ***Background***
  - ***Publish/Subscribe***
  - Type-based Publish/Subscribe
  - Decentralized Systems
  - Related Work
- CAMCAST
  - Intuition
  - Evaluation

Background

- Publish/Subscribe (Pub/Sub)
  - Two kinds of actors:
    - Subscribers (who want to receive information)
    - Publishers (who send the information)
  - Defines several properties:
    - Time decoupling
    - Space decoupling
    - Flow decoupling
  - Two different categories of Pub/Sub
    - Topic-based
    - Content-based

Background

- Topic-based Publish/Subscribe
  - Interests of subscribers based on topics (e.g., “car”)
  - Publishers send events on topics (e.g., “.car.BMW”)
- Content-based Publish/Subscribe
  - Interests of subscribers based on criteria (e.g., model = “BMW”, color = “red”, horsepower = “> 100”)
  - Publishers send plain information

Roadmap

- ***Background***
  - Publish/Subscribe
  - ***Type-based Publish/Subscribe***
  - Decentralized Systems
  - Related Work
- CAMCAST
  - Intuition
  - Evaluation

Background

- Type-based Publish/Subscribe
  - Interests of subscribers based on types (e.g., “interface Car {…}”)
  - Publishers send objects (e.g., new BMW(“red”,140);)
- Advantages
  - Both topic-based or content-based
  - Well-suited for systems implemented in an object-oriented language
    - One structure to maintain
    - No mapping between the high/low-level structures
    - Preserves type-safety (no bad cast)
- Disadvantages
  - How to filter objects?
  - A subscription to a type implies being interested in all the subtypes as well
  - Interoperability between types?

Roadmap

- ***Background***
  - Publish/Subscribe
  - Type-based Publish/Subscribe
  - ***Decentralized Systems***
  - Related Work
- CAMCAST
  - Intuition
  - Evaluation

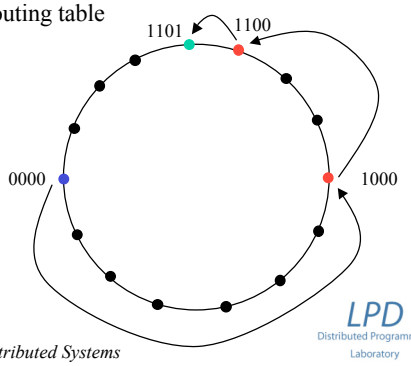
Background

- Centralized System (e.g., HTTP server)
  - Relies on one single entity to send/receive messages
  - Easy to maintain, to make assumptions on the system
  - Single point of failure, potential bottleneck
- Broker-based System (e.g., Usenet, Siena, "Gnutella Super Peers")
  - Relies on a set of well known servers to disseminate the messages
  - Still possible to administrate, to make assumptions on the system
  - No single point of failure
  - Some nodes (processes) have to perform more work
- Peer-to-Peer (P2P) System (e.g., Pastry, Tapestry, Chord)
  - Each process do the same amount of work
  - Completely decentralized
  - Very difficult to administrate, very few assumptions possible to make
  - Very difficult to shutdown
  - Bootstrap ?

Background

- Peer-to-Peer (P2P) Distributed Hashtable (DHT) Systems
  - Each process is responsible for storing one or more information
  - A mapping between the process and its information is created with a hash function
    - The key corresponds to the return value of the hash function
    - The value corresponds to the information stored by the process
  - The processes also store a routing table

Key	IP
0001	101….
0010	128….
0100	134….
1000	145….

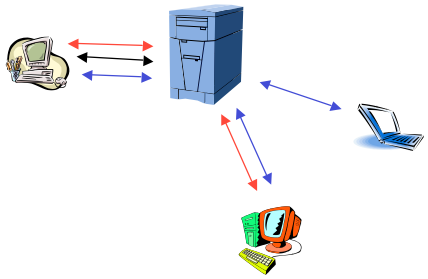


Roadmap

- ***Background***
  - Publish/Subscribe
  - Type-based Publish/Subscribe
  - Decentralized Systems
  - ***Related Work***
- CAMCAST
  - Intuition
  - Evaluation

Background

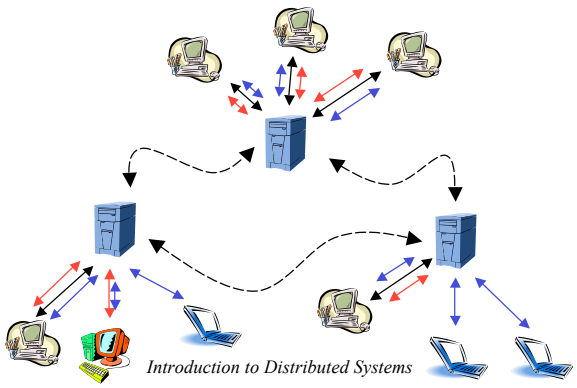
- Centralized Systems (e.g., JMS, Joram, MSMQ, CORBA)
  - Single point of failure
    - Replication means overhead
  - High load on the main server
  - Not type-based, not taking into account the type or topic hierarchy (except Joram)



Type hierarchy

Background

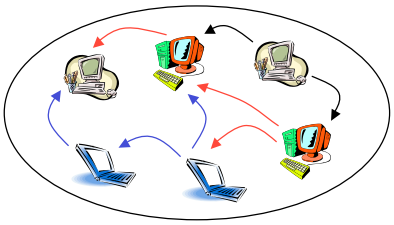
- Broker-based systems
  - Centralized or semi-centralized
  - The brokers have to filter the events
  - NNTP (Network News Transfer Protocol)
    - Processes must subscribe to all topics of a topic hierarchy

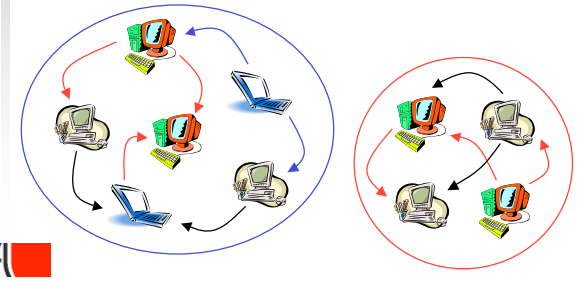


Type hierarchy

Background

- Gossip-based algorithms (e.g., b. multicast, lpbcast, Scamp)
  - Broadcast
    - Parasite events
    - Feasible?
  - Multicast
    - Multiple membership tables
    - Filtering overhead



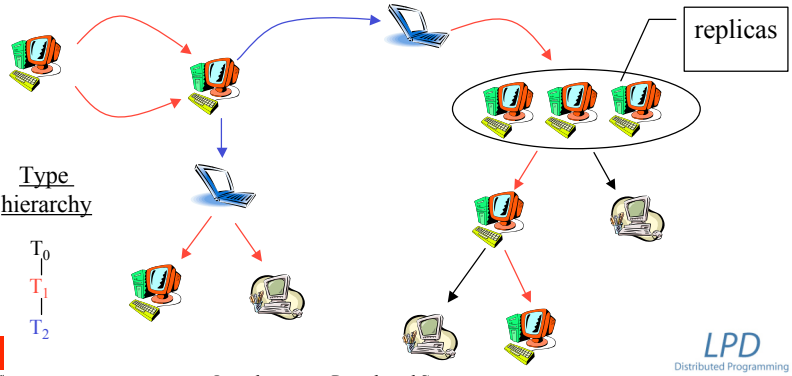


Type hierarchy

T_0
T_1
T_2

Background

- P2P Multicast (e.g., Bayeux, Scribe, HiScan)
  - Spanning trees
    - Single point of failures
    - Replication overhead
  - Parasite events

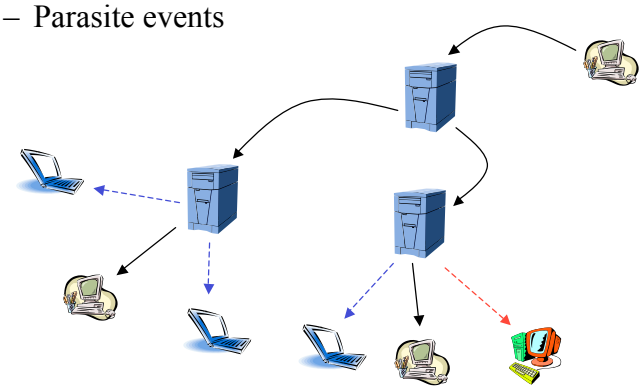


Type hierarchy

T_0
T_1
T_2

Background

- Content-based publish/subscribe (e.g., Gryphon, Siena, Hermes, PMcast)
  - Relies on brokers (filtering of events)
  - Parasite events

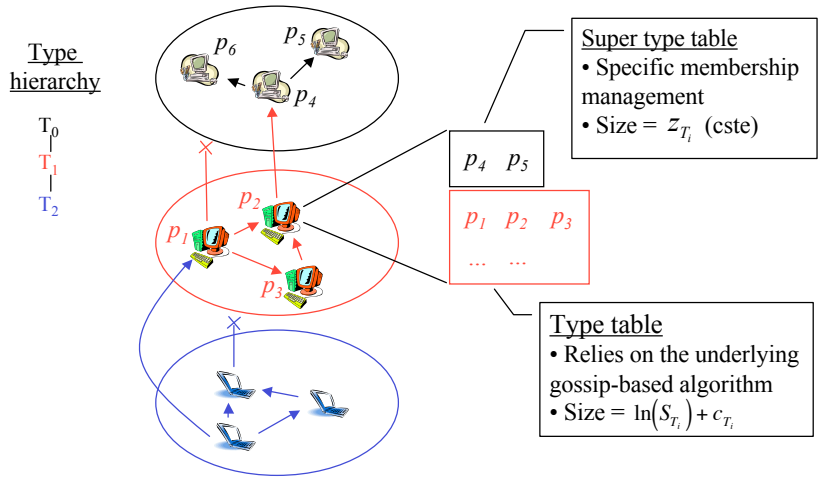


Type hierarchy

Roadmap

- Background
  - Publish/Subscribe
  - Type-based Publish/Subscribe
  - Decentralized Systems
  - Related Work
- ***CAMCAST***
  - ***Intuition***
  - Evaluation

Intuition



Algorithm

- Membership
  - Super-type table
    - Initialization
      - Bootstrapping technique (weak. conn. overlay network, …)
      - Table initialized as soon as a process interested in the direct super type is found
      - If no process is found, look in the super super community (and so on recursively)
    - Updates (pro-active algorithm)
      - If *k* super processes are down, the current process asks one of the alive super processes for a new view
      - New super type table gossiped in the community
  - Type table
    - Based on the underlying gossip-based membership algorithm

Algorithm

- Dissemination (gossip-based)
  - Every process (receiving the message the first time)
    - Gossips the message in its community
    - With a probability pit_{T_i}, propagates the message to its super community
  - Reliability
    - Gossip-based in each community
    - Depends on the type of interest, but can be controlled

Roadmap

- Background
  - Publish/Subscribe
  - Type-based Publish/Subscribe
  - Decentralized Systems
  - Related Work
- ***CAMCAST***
  - Intuition
  - ***Evaluation***

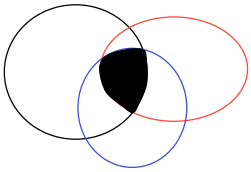
Evaluation

- Based on SCAMP [1]
  - Subscription
    - Initial contact
    - New subscription received
      - Forwarded to other processes of the view
    - Forwarded subscription received
      - With a probability of 1/(1+n) keep the new subscriber
      - Otherwise forward the new subscription to a random process in the view
  - Unsubscription
    - Done using a gossip-based mechanism
  - Network partition
    - Heartbeat messages
  - Graph rebalancing
    - Leases

[1] I. Gupta, A.-M. Kermarrec, A. J. Ganesh. Efficient and Adaptive Epidemic-style Protocols for Reliable and Scalable Multicast. *In IEEE Transactions on Parallel and Distributed Systems*, 2005

Evaluation

- Gossip-based algorithm based on SCAMP
  - Event disseminated to $\ln(n)+c$ induces a reliability of $e^{-e^{-c}}$
- Three alternative approaches (all based on SCAMP)
  - Gossip-based broadcast
    - One community for all processes
  - Gossip-based multicast
    - One community per type
  - Hierarchical SCAMP (HiSCAMP)
    - One community per type
    - Communities regrouped together

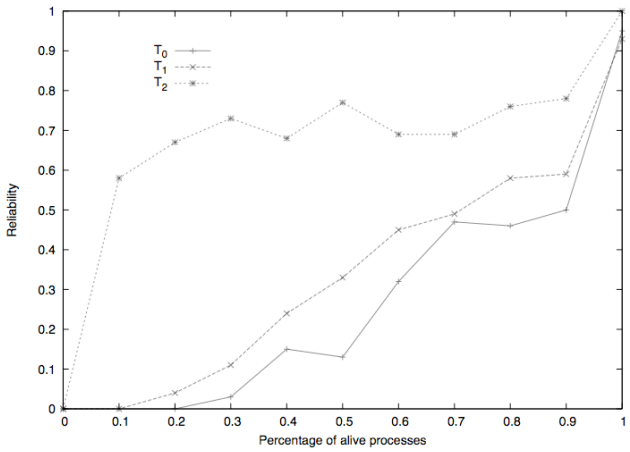


Evaluation

	Message Complexity	Reliability	Memory Information	Latency
Gossip-based Broadcast	$O(n\ln(n))$	$e^{-e^{-c}}$	$\ln(n)+c$	$O(R_n)$
Hierarchical gossip-based	$O(S_{T_{\max}}\ln(S_{T_{\max}}))$	$e^{-Ne^{-cT_i}-e^{-c2}}$	$\ln(S_{T_i})+c_{T_i}+\ln(N)+c_2$	$O(R_{S_{T\max}})$
Gossip-based Multicast	$O(S_{T_{\max}}\ln(S_{T_{\max}}))$	$\prod_{i=t-1}^{j} e^{-e^{-cT_i}}$	$\sum_{i=t-1}^{j}(\ln(S_{T_i})+c_{T_i})$	$O(R_{S_{T\max}})$
CAMCAST	$O(S_{T_{\max}}\ln(S_{T_{\max}}))$	$\prod_{i=t-1}^{j}(e^{-e^{-cT_i}}pit_{T_i})$	$\ln(S_{T_i})+c_{T_i}+z_{T_i}$	$O(R_{S_{T\max}})$

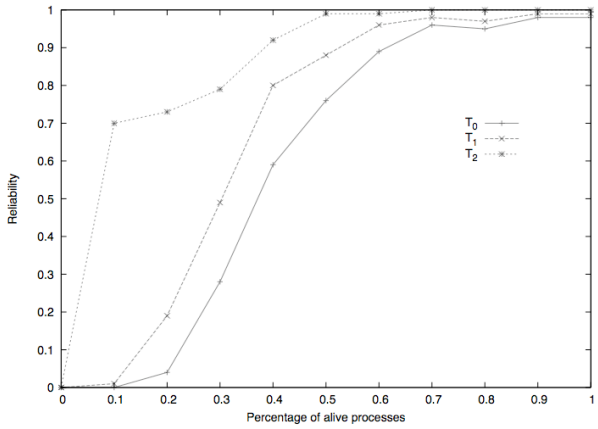
Evaluation

- Reliability (no membership algorithm)



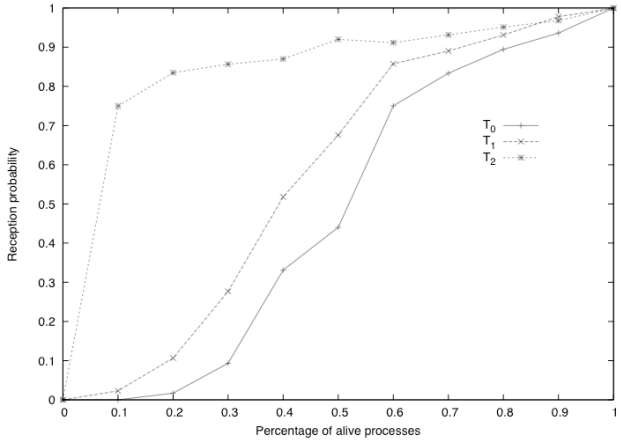
Evaluation

- Reliability (weak membership algorithm)



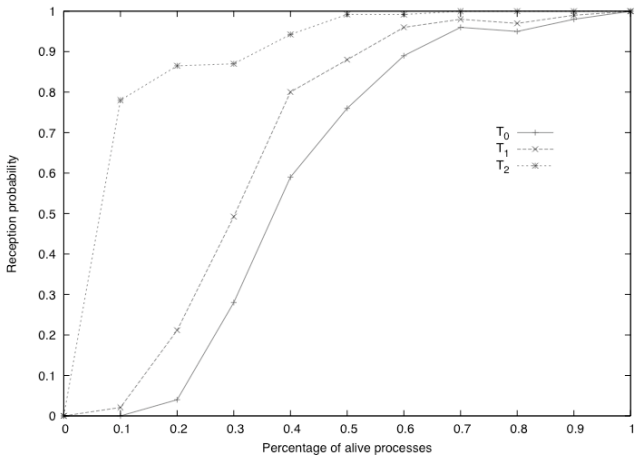
Evaluation

- Probability for a non-crashed process to receive a publication (no membership algorithm)



Evaluation

- Probability for a non-crashed process to receive a publication (weak membership algorithm)



Evaluation

- Latency (nb. rounds) and parasite events
  - Results shown for 1 publication

		$T_j, T_{j-1}, \ldots T_0$		
		1000,100,10	100 (j=2)	100 (j=4)
Latency	CAMCAST	9	9	13
	Gossip-based Broadcast	5	4	4
	PMcast	7	5	6
Parasite Events	CAMCAST	0	0	0
	Gossip-based Broadcast	11200	1700	3700
	PMcast	453	359	615

Summary

- CAMCAST enhances traditional gossip-based multicast algorithms to support type-based publish/subscribe
- It exploits the type hierarchy to ensure that there is no parasite event, low memory complexity and tunable reliability
- What about duplicates?
- Cost of the membership algorithm?
- Time decoupling?