

# Anonymous Publish/Subscribe in P2P Networks

A.K. Datta<sup>‡</sup>

M. Gradinariu<sup>\*</sup>

M. Raynal<sup>\*</sup>

G. Simon<sup>†\*</sup>

<sup>\*</sup> IRISA, Université Rennes 1, France

{raynal, mgradina, gsimon}@irisa.fr

<sup>†</sup> France Telecom R & D, Issy Moulineaux, France

gwendal.simon@rd.francetelecom.com

<sup>‡</sup> School of Computer Science, UNLV

datta@cs.unlv.edu

## Abstract

*One of the most important issues to deal with in peer-to-peer networks is how to disseminate information. In this paper, we use a completely new approach to solving the information dissemination problem. Our approach uses the publish/subscribe paradigm. The publish/subscribe method is the most inclusive strategy to establish communication between the information providers (publishers) and the information consumers (subscribers). We give a formal definition of publish/subscribe systems. We then use the publish/subscribe communication paradigm to design deterministic protocols (topic and content-based) for peer-to-peer networks. Our protocols are designed on top of an innovative information dissemination scheme, and can cope with the anonymity and mobility of both publishers and subscribers, weak-connectivity, and polarization, which are some of the characteristics of peer-to-peer networks. Moreover, in our solutions, every node could play a role of both publisher and subscriber. The algorithms are designed completely independent of the underlying routing substrates. The key advantage of our protocols is that they are scalable without additional re-organization cost. To the best of our knowledge, this is the first time the content-based subscription has been addressed in peer-to-peer networks.*

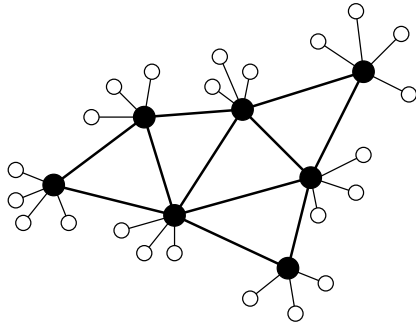
## 1 Introduction

Peer-to-peer systems have recently become a popular medium to share high volume of data. As these systems distribute the cost of sharing data — disk space for storing files and bandwidth for transferring them — across the peers in the network, they are scalable without using any

powerful or expensive servers. Designing a suitable scheme for information dissemination in peer-to-peer systems is an important and challenging area of research.

Publish/subscribe is a paradigm used to establish communication between the information providers and information consumers [4]. This scheme differs from the traditional point-to-point model in a number of ways. The communication method in the publish/subscribe scheme is anonymous, asynchronous, and multi-casting. Moreover, it can quickly adapt to dynamic environment.

**Peer-to-Peer Networks.** A peer-to-peer system is a dynamic and scalable set of processors (also referred as *peers*). In a peer-to-peer system the peers could join or leave the system at any time. The main characteristics of the peer-to-peer systems are the ability to pool together and harness large amounts of resources, self-organization, load-balancing, adaptation and fault-tolerance. The peer-to-peer systems can be categorized into two classes based on their degree of centralization. We call them *pure* peer-to-peer and *super-peer* networks. In a *pure* peer-to-peer system (e.g., Gnutella [9] or Freenet [8]), all peers have equal roles. The nodes have identical capabilities and responsibilities, and all the communications are symmetric. A *super-peer network* (Morpheus [11]) operates like a pure peer-to-peer network except that each peer in a super-peer network is connected to a set of clients. That is why, the peers in this system are called super-peers. Since the number and type of clients per super-peer can vary, the super-peer networks are not symmetric. Also, the peers do not need to be of similar capabilities. Figure 1 shows an example of a super-peer network. Every super-peer (represented by black nodes) are connected to a set of clients (represented by white nodes). When a client wants to submit a query to the network, it sends the query only to its super-peer. The research in peer-to-peer networks focussed on improving the search



**Figure 1. A Super-peer Network.**

efficiency by designing good search protocols (CAN [12], Pastry [14], Chord [15], Tapestry [17]). **Publish/Subscribe**

**Systems.** Intuitively, the Publish/Subscribe system is a two player game. The publishers are information providers or producers of events or notifications. The subscribers have the ability to express their interests in an event or a pattern of events, and the system provides them with every event fired by a publisher matching their registered interest. Publish/subscribe systems can be classified as of two types — *topic-based* and *content-based* — based on how the subscribers describe their interests. The topic-based publish/subscribe is similar to the notion of groups [6]. In the topic-based scheme [1], events are marked based on a fixed set of topics/subjects designated by the system. Each event is sent to one of the groups by its publisher. A user subscribes to one or more groups, and receives all the events published to the subscribed groups. In the content-based subscription systems [2, 4], the subscribers can refine their subscriptions by choosing filtering criteria along multiple dimensions without requiring the pre-definition of groups.

The implementation of a publish/subscribe system for fixed and mobile networks is based on the broker concept. The broker is the element of a network responsible for routing events between publisher and subscribers. A broker receives events posted by publishers and matches them against a set of subscriptions. There are two approaches (centralized and distributed) to implementing a publish/subscribe system in the context of both fixed and mobile systems. In the centralized approach, every new event is sent to a unique broker in the system which is responsible for matching the event against all the subscriptions in the system. Efficient matching techniques are presented in [2]. The version suitable for the mobile networks (presented in [10]) assumes that only the publisher and subscribers are allowed to be part of the mobile network while the broker is always available and part of a fixed network. The distributed approach uses a set of brokers. Each broker is responsible for a part of the subscriptions. A source event can publish a message to any broker which then forwards

the event to all other brokers in the system. In another implementation, the event is forwarded along the edges of a tree rooted at the originating broker. An efficient solution to the distributed content-based publish/subscribe system is proposed in [4]. In this system, upon receiving an event, a broker instead of sending the message to all the brokers first determines (by running some matching algorithm) which of its neighboring brokers should receive the event. This allows the (originating) broker to send the event to selective brokers only. This approach is specially efficient in networks where the addition and deletion of the subscribers are very rare. This approach requires every broker to have a complete knowledge of the network of brokers and the set of subscriptions.

The publish/subscribe schemes in the peer-to-peer networks are constructed on top of two popular object location and routing substrates, Pastry and CAN. In the Pastry system, each peer has a unique identifier. The routing in a Pastry system is implemented using a greedy approach. Given a message and a key, the message is routed to the Pastry node that is numerically closest to the key. With concurrent node failures, eventual delivery is guaranteed unless  $l/2$  (where  $l$  equals 16) or more nodes adjacent to each other fail simultaneously. The SCRIBE system [7] builds a multi-cast tree per group on top of a Pastry overlay and relies on Pastry in order to optimize the routes from the root to each group member. The CAN (content addressable network) design is based on a virtual  $d$ -dimensional Cartesian coordinate space. The space is dynamically partitioned among all the nodes in the system. Each CAN node maintains a coordinate routing table that holds the IP address and virtual coordinate zone of each of its neighbors in the coordinate space. Nodes use their routing tables to route messages towards their destination by using simple greedy forwarding to the neighbor with coordinates closest to the destination coordinates. The CAN multi-cast [13] does not build multi-cast trees. The messages are flooded to all nodes in a CAN overlay network. Multi-groups are supported by creating a separate CAN overlay per group.

The major drawback of the two previous publish/subscribe systems (discussed in the previous paragraph) is the high maintenance cost. In peer-to-peer networks, the multi-cast trees or separate CAN overlays are expensive tools. The cost increases with the increase of the number of groups in the network and the dynamic topology changes (addition/deletion of nodes). They are not easily scalable because they need to be re-organized after every change of network topology. Moreover, the two proposed schemes are not portable. They rely on a particular routing substrate, and address only the topic-based subscriptions.

**Our contributions.** We present anonymous schemes for both topic-based and content-based publish/subscribe systems in peer-to-peer networks. We propose a formal model

for the organization and information diffusion in the peer-to-peer networks. We also give a formal definition of the publish/subscribe system. Our schemes are characterized by a high portability since they are completely independent of the underlying routing substrates. Our contributions are three fold: First, our schemes are well-adapted to scalable systems without compromising any subscription criteria or network reorganization. Second, we maintain the anonymity of the distributed system — in order to maintain the network structure, we need only local information. Third, our solutions are fully decentralized, modular, and self-organizing, thus making them appropriate for practical implementations.

**Outline of the paper.** Section 2 presents our model for the organization of a peer-to-peer system. In Section 3, we introduce a new and fair scheme for information diffusion in peer-to-peer networks. In Section 4, we propose a formal definition for the publish/subscribe systems, followed by our solutions to the publish/subscribe problem in peer-to-peer networks.

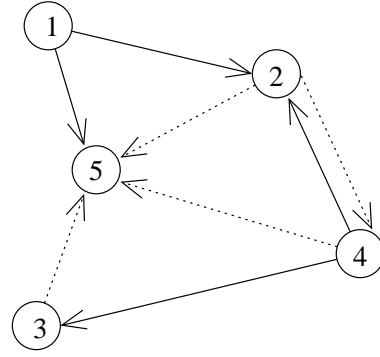
## 2 Logical Organization of Peer-to-Peer Networks

A peer-to-peer system is an asynchronous network subject to topology changes. Processors, also referred to as peers, can join or leave the system at any time. Processors and links can fail temporarily (transient faults) or permanently (crash failures). A processor in a classical peer-to-peer network submits queries and receives results (data) in return. The data shared in a peer-to-peer system could be of any type.

We model the organization of a peer-to-peer network as a logical multi-layer system, each logical layer  $l$  being a weakly connected graph, also referred to as the communication graph at Layer  $l$ . In order to connect to a particular layer  $l$ , the processors execute an underlying connection protocol. A processor  $p$  is called *active* at a layer  $l$  if there exists at least one processor  $q$  which is connected at  $l$  and aware of  $p$ . A logical link between two processors  $p$  and  $q$  at a layer  $l$  could be in one of the following states: *up* (the two processors are aware of each other at  $l$ ), *down* ( $p$  and  $q$  are not aware of each other at  $l$ ), and *forming* (at least one of the processors has initiated the connection protocol for  $l$ ). The logical *neighbors* of a processor  $p$  at a layer  $l$  (denoted by  $\mathcal{N}^l(p)$ ) are the set of processors  $q$  such that the logical link  $(p, q)$  is up.

Note that a processor  $i$  may belong to several layers simultaneously. So,  $i$  may have different sets of neighbors at different logical layers. The network presented in Figure 2 has two logical layers. The links of the two layers are distinguished in the figure by using two types of lines. The

logical neighbors of Processor 5 in Layer 1 are Processors 2, 3, and 4, and its neighbor at Layer 2 is Processor 1.



**Figure 2. Logical Multi-layer Network. The links of Layers 1 and 2 are represented by solid and dotted lines, respectively.**

A logical layer  $l$  at time  $t$  is characterized by

- the active nodes at  $l$  at time  $t$ , denoted by  $V^l(t)$ ;
- the logical links up at time  $t$  at  $l$ , denoted by  $E^l(t)$ ;
- the logical orientation of the communication graph,  $G^l(t) = (V^l(t), E^l(t))$ , denoted by  $\mathcal{R}^l$ .

Note that since we consider peer-to-peer networks in this paper, the communication graph(s) at time  $t$  may be different from that at time  $t + 1$ .

Processors may run different algorithms at different layers. The state of a processor  $p$  at layer  $l$  at time  $t$  is given by the values of  $p$ 's variables and the state of its adjacent logical links at time  $t$ , and is denoted by  $State(V^l(t))$ . The configuration of a layer  $l$  at time  $t$  consists of the state of the active processors in  $l$  at  $t$ , the communication graph  $(G^l(t) = (V^l(t), E^l(t)))$ , and the logical orientation of the communication graph,  $\mathcal{R}^l$ . Formally,  $c_t^l = (State(V^l(t)), \{(G^l(t) = (V^l(t), E^l(t)), \mathcal{R}^l)\})$ .

The configuration of a multi-layer network at time  $t$  is represented by the state of all active processors at time  $t$ , the set of communication graphs corresponding to all the layers, and the logical orientation for all logical layers  $l \in \mathcal{L}$ , where  $\mathcal{L}$  is a finite set of logical orientations. Formally,  $c_t = (c_t^{l_1}, \dots, c_t^{l_k})$ , where  $c_t^{l_i}$  is the configuration of layer  $l_i \in \mathcal{L}$  at time  $t$ .

The system transitions correspond to the topological changes or some internal actions executed by some processors. A transition is labeled with the labels of the layers  $l$  involved. A system execution is a maximal sequence of transitions.

We now show that if every layer in the system satisfies a property  $\mathcal{P}$  and the layers are pairwise independent (the

actions executed at a layer do not influence any other layer), then the multi-layer system also satisfies  $\mathcal{P}$ .

**Definition 2.1 (Pairwise Independent Layers)** Let  $S$  be a system logically organized in a set  $\mathcal{L}$  of logical layers. Two layers  $l_1$  and  $l_2$  in  $\mathcal{L}$  are called pairwise independent if no action executed by a process at  $l_1$  involves Layer  $l_2$ .

**Theorem 2.1 ([3])** Let  $S$  be a logical multi-layered system with pairwise independent layers,  $\mathcal{L}$  the set of layers, and  $SP^l$  the specification of the algorithm executed at layer  $l \in \mathcal{L}$ .  $S$  satisfies  $\bigwedge_{l \in \mathcal{L}} SP^l$ .

**Corollary 2.1** Let  $S$  be a logical multi-layer system with pairwise independent layers, and  $SP$  the specification of the algorithms executed at every layer. Then  $S$  satisfies  $SP$ .

### 3 A New approach to Logical DAGs

The publish/subscribe algorithms proposed in this paper assume that the underlying communication graph is a directed acyclic graph (DAG). We will show later in this section how to maintain the acyclicity of the graph. The use of logical DAGs in the peer-to-peer systems has a few advantages. The maintenance of the structure needs only the local information, i.e., the state of a processor and its neighbors. Hence, the network is scalable without any additional cost of re-organization. Moreover, this structure logically breaks the peer-to-peer network symmetry by designating two types of nodes: “sink nodes” (see Definition 3.3 below) and “non-sink nodes”. This distinction is used only to implement the information dissemination. Typically, it is difficult to avoid network flooding to disseminate information in peer-to-peer systems since all the nodes have the same role. In our work, we avoid this symmetry by maintaining the DAG which contains two distinct types of nodes, sink and non-sink. In our proposed algorithms, only one of the two types of nodes (the sink nodes) are enabled to diffuse information. The non-sink nodes will have to wait until they become sink nodes to disseminate information. Our DAG orientation scheme ensures that all non-sink nodes will eventually become sink nodes.

In this section, we define the logical orientation of edges to maintain a DAG. Every processor maintains two variables: an identifier ( $lid$ ) and an integer variable ( $val \in \{0, 1, 2\}$ ). We assume that the processor identifiers are unique in their neighborhood.

**Definition 3.1** The relation  $\prec$  is defined as follows:

$$x \prec y \Leftrightarrow y = (x + 1) \bmod 3$$

**Definition 3.2** Let  $G(V, E)$  be a communication graph. The logical orientation of the edges  $\rightarrow$  is defined as follows: the edge  $(p, q)$  is logically oriented from  $q$  to  $p$  ( $q \rightarrow p$ ) iff  $(val_p \prec val_q)$  or  $(val_p = val_q \wedge lid_p < lid_q)$ .

**Definition 3.3 (Sink Node)** Let  $G(t)$  be the communication graph at time  $t$ . A processor  $p$  is called a sink node if

$$\forall q \in \mathcal{N}_p : q \rightarrow p$$

By Definition 3.2, the network oriented according to the relation  $\rightarrow$  contains at least one sink processor.

### 3.1 Edge Re-orientation in a Logical DAG

In this section, we present a new link re-orientation scheme which can cope up with the node additions and deletions. We first present a scheme (later referred to as the mechanism of information dissemination) which assumes that the network is logically organized as one virtual layer. (Later we show how to extend this scheme for multi-layer organization.) Our scheme is based on the edge reversal idea of [5]. Only the sink processor is privileged to execute a task. After the sink processor finishes its task, its adjacent edges are re-oriented to ensure the fairness among all neighboring processors.

The idea of the re-orientation algorithm is simple. A sink processor is enabled to execute a rule of LLRO (Module 1), which re-orient its adjacent edges toward its neighbors. The edge reversal is implemented by changing  $val$  and sometimes,  $lid$ . If all neighbors of the sink processor have the same value  $v$  of  $val$ , then the sink sets its  $val$  to  $(v + 1) \bmod 3$  (see Rule  $\mathcal{R}_1$ ). Hence, all adjacent edges are re-oriented towards its neighbors.

---

**Module 1 LLRO: Logical Link Re-orientation Scheme in a One-Layer Network (Processor  $i$ )**

---

**Parameters :**

$\mathcal{N}(i)$ : set of neighbors;  
 $val_i \in \{0, 1, 2\}$ : integer;

**Functions :**

$sink(i) : \forall j \in \mathcal{N}(i), j \rightarrow i$   
 $choose_{id} : returns the first identifier  $id$  available in  $\mathcal{N}_i$  and larger than the maximum of the identifiers of neighbors  $j$  verifying  $val_i \prec val_j$ .$

**Actions :**

$\mathcal{R}_1 : if\ sink(i) \wedge (\forall j \in \mathcal{N}(i), val_j = val_i)$   
 $val_i = (val_i + 1) \bmod 3;$

$\mathcal{R}_2 : if\ sink(i) \wedge (\exists j \in \mathcal{N}(i), val_i \prec val_j)$   
 $val_i = max_{j \in \mathcal{N}(i)}(val_j);$   
 $lid_i = choose_{id};$

---

If the sink processor  $i$  has at least one neighbor  $j$  such that  $val_i \prec val_j$ , then  $i$  sets its  $val$  to the maximal value of  $val$  of all its neighbors by executing the first part of  $\mathcal{R}_2$ . Note that this action does not reverse the edges of the

neighbors with  $lid_j > lid_i$ . That is taken care of by the second part of  $\mathcal{R}_2$ . In this part of  $\mathcal{R}_2$ ,  $lid_i$  is adjusted to make  $lid_i > lid_j$ . In summary, after the execution of  $\mathcal{R}_1$  or  $\mathcal{R}_2$ , the sink nodes reverse their adjacent edges towards their neighbors.

**Lemma 3.1** *Let  $G(t)$  be the communication graph at time  $t$  and  $G(t+1)$  the communication graph after the execution of Module 1 by a sink processor  $p$ . If  $G(t)$  is acyclic, then  $G(t+1)$  is acyclic.*

The following lemma proves two important properties of LLRO (Module 1) — starvation freedom (i.e., every processor eventually executes its actions) and liveness (i.e., a processor executes its actions infinitely often).

**Lemma 3.2** *Let  $e$  be an execution of LLRO algorithm (Module 1). Every processor executes its algorithm infinitely often even in the presence of topology changes (i.e., the addition and deletion of processors).*

**Note 3.1** *Note that since the underlying neighborhood maintenance protocol updates the list of neighbors, even if the network becomes partitioned, the edge reversal scheme works with no additional cost in every individual partition.*

The algorithm presented as Module 2 generalizes the orientation scheme for networks organized in multiple layers. It is trivial to prove (see Theorem 2.1) that the generalized scheme preserves the properties of the single layer scheme (shown as Lemmas 3.1 and 3.2).

---

**Module 2** LLRM - Logical Link Reorientation Scheme in a Multi-layer Network (Processor  $i$ ).

---

**Parameters :**

$\mathcal{L}$ : the set of layers  $i$  belongs to;  
 $Val = \{val_i^l \in \{0, 1, 2\} \mid l \in \mathcal{L}\}$ : set of integers,  
 0 by default;  
 $\mathcal{N} = \{\mathcal{N}^l(i) \mid l \in \mathcal{L}\}$ : the set of  
 neighbors of  $i$  for all levels in  $\mathcal{L}$ ;

**Relation:**

$\forall l \in \mathcal{L}$ ,  $\rightarrow^l$  is relation  $\rightarrow$  with respect to  $val^l$   
 and identifiers;

**Function :**

$sink^l(i) : \forall j \in \mathcal{N}^l(i), j \rightarrow^l i$

**Action :**

$\mathcal{R} : if \exists l \in \mathcal{L}, sink^l(i)$   
 Execute Module 1 with parameters  $val_i^l$   
 and  $\mathcal{N}^l(i)$

---

## 3.2 Peer Connections and DAG Maintenance

In this section, we present modules which maintain the acyclicity of the communication graph in spite of node additions and deletions. The goal is to make sure that when a new processor  $p$  joins the system, the orientation of the newly created edges between  $p$  and its neighbors must not break the acyclicity of the logical orientation of the communication graph. We present a connection mechanism which avoids cycle creation and tries to minimize the number of critical points<sup>1</sup>.

We assume that a processor  $p$  starts the connection phase with  $lid_p$  different from all its connection points<sup>2</sup>. But, no value for the variable  $val$  is used in order to define the orientation of the new edges. The first step of the connection algorithm sets the value of  $val$ . The processor  $p$  sends a message  $request\_val$  to a processor  $p_0$  connected at  $l$ . Module 3 shows the actions taken by a processor upon receiving this message.

If  $p_0$  is a sink node, it responds by sending two messages: a message  $respond\_val$  carrying a value  $val$  such that  $p \rightarrow p_0$  is maintained and a message  $link\_up\_OK$ . If  $p_0$  is not a sink node, it forwards the message  $request\_val$  to a neighbor  $p_1$  with  $p_0 \rightarrow p_1$ . The message forwarding continues until the message reaches a sink node  $p_i$ .

---

**Module 3** Receiving a message  $request\_val$  from a processor  $j$  (Processor  $i$ )

---

**Parameters :**

$\mathcal{N}(i)$ : set of neighbors;  
 $val_i \in \{0, 1, 2\}$ : integer;

**Functions:**

$choose\_Neighbor()$  = choose a neighbor  $p_k$  with  
 $p_i \rightarrow p_k$   
 $send(message, proc)$  : sends the message  $message$  to  
 processor  $proc$

**Actions :**

$i$  receives  $request\_val$  from  $j$   
 if  $sink(i)$   
   if  $lid_j < lid_i$   
      $send(respond\_val (val_i + 1) \bmod 3, p_j)$   
   else  
      $send(respond\_val val_i, p_j)$   
      $send(link\_up\_OK, p_j)$   
 else  
    $p_{forward} = choose\_Neighbor()$   
    $send(request\_val, p_{forward})$

---

<sup>1</sup>A point is called critical or articulation point if its failure disconnects the network.

<sup>2</sup>A processor to connect to a peer-to-peer network should be aware of one or more processors (referred to as connection points) already in the system

When  $p$  receives a message  $respond\_val$  from  $p_i$ , it initializes its  $val$ , then sends  $request\_link\_up$  to all processors  $p_j$  with  $j \in \{0, \dots, i-1\}$ . The processors  $j$  wait until they become sink. Until then, the edge  $(p_j, p)$  is maintained in the state forming by both  $p$  and  $p_j$ . When a processor  $p_j$  becomes a sink node, it responds by  $link\_up\_OK$  if  $p \rightarrow p_j$  or by  $link\_down$ , otherwise (see Module 4).

---

**Module 4 CC - Connection Creation (Processor  $i$ )**

---

**Parameters :**

$\mathcal{N}(i)$ : set of neighbors;  
 $val_i \in \{0, 1, 2\}$ : integer;  
 $forming_i$ : list of processors  $p$  with  $(i, p)$  forming;

**Functions:**

$sink(i) : \forall j \in \mathcal{N}(i), j \rightarrow i$   
 $send(message, proc) : \text{sends the message } message \text{ to processor } proc$

**Actions :**

$i$  receives  $respond\_val$  from  $j$   
 if  $sink(i)$   
 $\forall p \in forming_i$ :  
 if  $val_i < val_p \vee (val_p = val_i \wedge lid_i < lid_p)$ ;  
 $send(link\_up\_OK, p)$ ;  
 else  
 $send(link\_down, p)$ ;

---

**Lemma 3.3** *Let  $G(t)$  be the communication graph at time  $t$ . Assume that  $G(t)$  is acyclic. Let  $(p_i, p_j)$  be an up link and  $G(t+1)$  the communication graph after the creation of  $(p_i, p_j)$ . Then  $G(t+1)$  is acyclic.*

**Note 3.2** *When a processor leaves the system, it does not affect the acyclicity of the communication graph.*

## 4 Hybrid Publish/Subscribe Scheme

In Section 2, we presented a multi-layer peer-to-peer network model. Then in Section 3, we designed a logical DAG maintenance protocol to disseminate information in these networks. In this section, we use the multi-layer model and the DAG to design deterministic publish/subscribe algorithms for the peer-to-peer systems. We discuss both single topic and multiple topic algorithms below. Most importantly, we formally define the properties of a publish/subscribe system and provide solutions for both topic-based and content-based subscriptions. We are not aware of any formal definition of the publish/subscribe system. One of the unique features of our scheme is that a processor can be both a publisher and a subscriber. Hence, we call this scheme a hybrid publish/subscribe scheme.

### 4.1 Definition of Publish/Subscribe

A publish/subscribe system should satisfy the following three properties:

**Event Publication Liveness:** If a publisher publishes an event (or a news)  $m$  on a topic  $t$ , then  $m$  is eventually delivered to every live subscriber for the topic  $t$ .

**Publish/Subscribe Validity:** If a process delivers an event (or a news)  $m$  on a topic  $t$ , then  $m$  has been published by some publisher.

**Publisher Liveness:** Every publisher can publish infinitely often.

An anonymous publish/subscribe system should also verify the following additional property:

**Publish/Subscribe Anonymity:** The information is diffused in the network in an anonymous way.

### 4.2 Topic Based Publish/Subscribe

In a publish/subscribe system, subscribers subscribe to some specific categories of events. Every processor maintains a list  $\mathcal{L}$  of topics of which it is a member. It subscribes to a single or multiple categories in a single or multiple topic systems, respectively. The subscriptions are modeled using links in the communication graphs, each graph  $G^l$  representing the publish/subscribe system for the topic  $l \in \mathcal{L}$ .

#### 4.2.1 Publishing Algorithm

Our logical DAG orientation model guarantees the existence of at least one sink in the communication graphs at any time. The sink processors behave as privileged processors to send or forward any information regarding any event of the publish/subscribe system. They are allowed to send messages that are generated locally and received from the neighbors. After sending the messages, they re-orient their adjacent edges (Module 1). A non-privileged (i.e., not sink) processor can receive some messages, but is not allowed to forward the information until it becomes a sink. Hence, only the sink processors are points of diffusion, thus avoiding the network flooding.

The publish/subscribe scheme (shown as Module 5) uses two sets of buffers and two communication primitives, **send()** and **receive()** to disseminate all the events to the interested subscribers using the underlying oriented (DAG) communication graph. The two buffers are called input and local buffers. In Module 5,  $Input\_Messages^l$  and  $Local\_Messages^l$  refer to the messages stored in the input and local buffers, respectively. The input buffer is used to record all the messages received from the neighbors in layer  $l$ , and the local buffer saves the messages generated locally.

---

**Module 5** Publish/subscribe Scheme (Processor  $i$ )

---

**Volatile Variables:**

$Input\_Messages_i^l$ : list of messages received from  $i$ 's neighbors, initially empty;  
 $Local\_Messages_i$ : list of messages generated locally;  
 $forming_i^l$ : list of processors  $p$  with  $(i, p)^l$  forming;

**Primitives:**

$receive(Input\_Messages_i)$ : returns the list of messages received from the input buffer;  
 $collect(Local\_Messages_i)$ : reads the local buffer and returns messages in  $Local\_Messages_i$ . The invocation of this primitive empties the local buffer;  
 $send(New\_Messages_i)$ : sends the messages from  $New\_Messages_i$ ;

**Action:**

$\mathcal{R}1$  : *if*  $sink^l(i)$   
    execute CC (Module 4) with  $forming_i^l$   
     $receive(Input\_Messages_i^l)$ ;  
     $collect(Local\_Messages_i^l)$ ;  
     $send(Input\_Messages_i^l \cup Local\_Messages_i^l)$ ;  
    execute LLRO (Module 1)

---

The **receive()** primitive when invoked reads the input buffer and returns the messages collected between the previous and current invocation of **receive()**. Processors write their own (i.e., generated locally) messages in their local buffer. A processor calls **send()** to send all its neighbors the messages returned by the receive primitive and the ones produced locally (stored in the local buffer).

**Lemma 4.1** *A published message  $m$  reaches all active processors in a connected communication graph  $G$  in a finite time.*

### 4.2.2 Subscription Algorithm

We define a particular layer  $l_0 \in \mathcal{L}$  at which all processors must be connected. When a processor joins the system, it first executes the connection primitives (Modules 3 and 4) for the layer  $l_0$ . This layer has two main functionalities. It facilitates announcement of the creation of new topics (new layers). It also implements the subscription to another layer. **New topic declaration.** When a processor  $p$  wants to create a new topic, it announces the topic to all processors. It uses the layer  $l_0$  to publish a message in which it describes the new layer  $l_{new}$  with  $l_{new} \notin \mathcal{L}$  and the topic of this layer. This message is propagated to all processors connected at layer  $l_0$ . Eventually, by Lemma 4.1, all processors in the network receive this message.

**Connection to a layer.** When a processor  $p$  wants to connect to a layer  $l$ , it first needs to know a processor connected at  $l$ . To find one of these processors,  $p$  relies on the layer

$l_0$ . It publishes a message *query\_processor*  $l$  to know some processors already connected at  $l$ . When a processor  $q$  connected at  $l$  receives this message, it stops the propagation of this message and sends contact information to  $p$ . Hence, the processor  $p$  can execute the usual connection primitives for layer  $l$  using  $q$  as the connection point (Modules 3 and 4).

## 4.3 Content Based Publish/Subscribe

In the content-based publish/subscribe systems, the nodes are characterized by a subscription predicate. Since the subscription may be different for two arbitrary nodes in the network, it is impossible to implement the content-based publish/subscribe in a pure peer-to-peer network. The minimal peer-based topology necessary in order to perform a content-based publish/subscribe scheme is a super-peer network. (See [16] for more details on the design of super-peer networks.) Super-peers are connected to each other as processors in our multi-layer DAG network. Moreover, super-peers act as servers to subsets of clients (or peers). In this section, *cluster* refers to a super-peer and all peers connected to it.

### 4.3.1 Subscription Algorithm

**Peers subscription.** In a content-based publish/subscribe scheme, clients can choose the filtering criteria for the messages they want to receive. Our super-peer organization operates as the traditional client-broker scheme, where clients inform their associated broker about their subscription criteria.

When a peer  $p$  joins the system, it first connects to a peer  $p_c$  which informs  $p$  about its membership to a cluster  $Cl_{p_s}$  with super-peer  $p_s$ . Then,  $p$  requests to connect to  $p_s$  (Modules 3 and 4). Peer  $p$  defines a predicate  $c$  using variables representing a set of topics  $T_c$ .  $p$  submits the set of criterion  $c$  and its associated topics to  $p_s$ . The super-peer  $p_s$  computes the common set between topic  $T_c$  and the topics to which  $p_s$  and its neighbors are already connected. If in the neighborhood of  $p_s$ , there is another super-peer  $p'_s$  which has more common topics with  $p$ , then  $p_s$  proposes to  $p$  to join the cluster of the super-peer  $p'_s$ ,  $Cl'_{p'_s}$ . Thus, the network self-organizes such that the clients with common connection criterion (or interests) join a cluster.

If a peer  $p$  modifies, adds, or removes its criteria or part of the criteria, it informs its super-peer immediately about these modifications. Using the same mechanism as described in the previous paragraph, the super-peer may propose to another super-peer which better matches the new criterion of  $p$ .

**Super-peers subscription.** A super-peer acts with the peers in its cluster like a broker in a traditional publish/subscribe scheme. In addition, it also plays the role of a peer in the

super-peers network, organized as a multi-layer DAG topic-based publisher/subscriber.

A super-peer  $p_s$  belonging to the cluster  $Cl_s$  must store the set of topics  $t_i$  associated with the criteria  $c_i$  defined by each peer  $p_i \in Cl_s$ . Thus, using the set of topics  $T_s = \bigcup t_i$ , the super-peer  $p_s$  determines the set of layers  $L_s$  at which it must be connected to in order to receive the messages that could satisfy the subscriptions of the peers  $p_i$ .

When a super-peer  $p_s$  receives a message  $m$  from the layer  $l_i \in L_s$ , it first determines the set of peers  $P_i$  whose subscription criteria depend on the topic  $t_i$  associated to  $l_i$ . If  $m$  satisfies the criteria defined by the peer  $p_i \in P_i$ , it sends  $m$  to  $p_i$ .

### 4.3.2 Publishing Algorithm

For peers, the publishing action simply consists of sending messages to their super-peer. When a super-peer  $p_s$  receives a message  $m$ , it sends  $m$  immediately to all peers  $p \in Cl_{p_s}$  whose criteria match with  $m$ . Then, it determines the layer  $l_m$  associated with the topic of  $m$ . If  $p_s$  is connected at layer  $l_m$ , it stores  $m$  in its buffer of new messages and publishes it when it becomes a sink. If  $p_s$  is not connected to  $l_m$ , it can decide to subscribe to  $l_m$ , even if no peer is interested in the topic. This can be an efficient decision if the peers frequently send messages for this layer. Otherwise, it uses the information about the neighbors' layers to send  $m$  to another super-peer connected to  $l_m$ .

## 5 Conclusions

In this paper, we presented the first anonymous publish/subscribe communication paradigm in peer-to-peer networks. We formally defined the logical network organization and the publish/subscribe problem. We presented a new information dissemination scheme which uses only local information and copes with the system scalability at no additional re-organization cost. Moreover, we provided algorithms for the topic and content-based publish/subscribe system in the context of peer-to-peer networks. An interesting future research direction is defining complexity or performance metrics and the method to evaluate them related to the publish/subscribe problem, especially in peer-to-peer networks. We plan to measure the delay between a news publication and its reception which is obviously dependent on network topology changes. It may also be useful to measure the impact of the system self-organization on the publish/subscribe process.

## References

[1] K. Aguilera and R. Strom. Efficient atomic broadcast using deterministic merge. In *Proceedings of the Nineteenth Annual*

*ACM Symposium on Principles of Distributed Computing (PODC'00)*, pages 208–218, 2000.

[2] M. Aguilera, R. Strom, D. Sturman, M. Astley, and T. Chandra. Matching events in a content-based subscription system. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing (PODC'99)*, pages 53–61, 1999.

[3] E. Anceaume, A. K. Datta, M. Gradinariu, and G. Simon. Publish/subscribe scheme for mobile networks. *ACM/POMC*, page to appear, 2002.

[4] G. Banavar, T. Chandra, B. Mukherjee, and J. Nagarajao. An efficient multicast protocol for content based publish subscribe systems. In *Proceedings of the 19th International Conference on Distributed Computing Systems (ICDCS'99)*, 1999.

[5] V. Barbosa and E. Gafni. Concurrency in heavily loaded neighborhood-constrained systems. *ACM Transactions on Programming Languages and Systems*, 11(4):562–584, 1989.

[6] K. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):36–53, 1993.

[7] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected areas in Communications*, 20(8):100–111, 2001.

[8] Freenet. Freenet website. <http://freenet.sourceforge.net>.

[9] Gnutella. Gnutella website. <http://gnutella.wego.com>.

[10] Y. Huang and H. Garcia-Molina. Publish/subscribe in a mobile environment. *ACM Int. Workshop on Data Engineering for wireless and mobile access (MOBIDE'01)*, pages 27–34, 2001.

[11] Morpheus. Morpheus website. <http://www.musiccity.com>.

[12] S. Ratnasamy, P. Francis, M. Handley, and R. Karp. A scalable content-addressable network. *ACM SIG/COMM*, pages 161–172, 2001.

[13] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application level multicast using content addressable networks. *Proc. of the Third International Workshop on Networked Group Communication*, pages 14–29, 2001.

[14] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large scale peer-to-peer systems. *Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, 2001.

[15] I. Stoica, R. Morris, D. Karger, K. M., and B. H. Chord. A scalable peer-to-peer lookup service for internet applications. *ACM SIG/COMM*, pages 149–160, 2001.

[16] H. Yang, B. Garcia-Molina. Designing a super-peer network. *Technical report Stanford Database group (<http://dbpubs.stanford.edu/pub/2002-13>)*, 2002.

[17] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. *Technical Report UCB/CSD-01-1141, Computer Science U.C. Berkeley*, 2001.