Content-Based Publish/Subscribe Systems

Haiying Shen

Abstract The application and deployment of publish/subscribe systems have developed significantly over the past years. A publish/subscribe system is a powerful paradigm for information dissemination from publishers (data/event producers) to subscribers (data/event consumers) in large-scale distributed networks. Publish/subscribe systems have been used in a variety of applications ranging from personalized information dissemination to large-scale and critical monitoring. This chapter provides a survey on current content-based publish/subscribe systems. It first introduces the publish/subscribe systems, then presents a survey of current systems based on three classification criteria: subscription model, routing and topology. It details different publish/subscribe systems in the centralized category and distributed category including multicast-based systems and Distributed hash table (DHT)-based systems. Finally, it concludes the chapter with concluding remarks and open issues.

1 Introduction

In the past few years, with the tremendous development of Internet and rapid growth of information, more and more Internet applications require information dissemination among a large number of widely scattered entities. In this environment, thousands or even millions entities are distributed globally and their locations and behaviors may vary. The large-scale, dynamic and geographically spread features of the environment requires a scalable, efficient and reliable technique for information dissemination. The rigid and static individual point-to-point and synchronous communications are not able to meet the requirements. Publish/subscribe (pub/sub) systems has been receiving increasing attention for the loosely coupled form of interaction it provides in large scale settings [65]. A pub/sub system [47] enables information

Haiying Shen
University of Arkansas, Fayetteville, AR, USA, e-mail: hshen@uark.edu

X. Shen et al. (eds.), *Handbook of Peer-to-Peer Networking,*

DOI 10.1007/978-0-387-09751-0_49,

dissemination from *publishers* (data/event producers) to *subscribers* (data/event consumers) in large-scale distributed networks.

The first pub/sub system was the "news" subsystem in the Isis Toolkit and was described in [19]. This pub/sub technology was invented by Frank Schmuck, who probably should get the credit as the first person to ever invent a fully functional pub-/sub solution [2]. Since then, significant research work has been devoted to developing efficient and scalable pub/sub systems. Pub/sub systems have been applied to a wide range of group communication applications including software distribution, Internet TV, audio or video-conferencing, virtual classroom, multi-party network games, distributed cache update, distributed simulation and shared white-boards. It can also be used in even larger size group communication applications, such as broadcasting and content distribution. Such applications in our daily lives include news and sports ticker services, real-time stock quotes and updates, market tracker, and popular Internet radio sites [16].

A pub/sub system is composed of many nodes distributed over a communication network. In such a system, clients are autonomous entities that exchange information by publishing events and by subscribing to the classes of events they are interested in. Clients are not required to communicate directly among themselves but are rather decoupled: the interaction occurs through the nodes of the pub/sub system that coordinate themselves in order to route information from publishers to subscribers [6]. Figure 1 shows a high-level view of a pub/sub system. In the system, publishers produce information and subscribers consume information. Specifically, publishers publish information in the form of events and subscribers express their interests in an event or a pattern of events in the form of subscription filters. A data event specifies values of a set of attributes associated with the event. The subscriptions can be very expressive and specify complex filtering criteria by using a set of predicates over event attributes. When a pub/sub system receives an event published by a publisher, it matches the event to the subscriptions and delivers the event to the matched subscribers. A subscriber installs and removes a subscription from the pub/sub system by executing the subscribing and unsubscribing operations respectively.

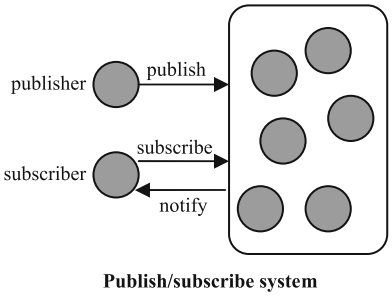

Fig. 1 A high-level view of a pub/sub system

Processes in pub/sub systems are clients of an underlying *notification service* and can act both as producers and consumers of messages, called event notifications or notifications for short. A notification is a message that describes an event. Notifications are injected into the event system via a `publish()` call rather than being published towards a specific receiver. They are conveyed by the underlying notification service to those consumers which have registered a matching subscription with `subscribe()`. Subscriptions describe the kind of notifications consumers are interested in.

A variety of content-based pub/sub systems have been proposed. The pub/sub systems can be classified into groups according to three criteria: subscription model, routing and topology. Based on the expressive power of subscription models, pub-/sub systems can be classified into three categories: topic-based, content-based and type-based. According to routing solutions for the notification service, the pub/sub systems can be categorized into filter-based approaches [8, 26, 27, 36, 73, 90] and the multicast-based approaches [8, 73, 83, 104]. Based on the system topology, current pub/sub systems can be classified into centralized [88] and distributed [25, 28, 29, 102]. The distributed systems can be further classified into broker-based and Distributed Hash Table (DHT)-based systems. DHT systems are also called structured peer-to-peer (P2P) systems, which is one type of P2P systems.

Table 1 Classification of pub/sub systems

Classification criteria	Categories	
Subscription model	Topic-based	
	Content-based	
	Type-based	
Routing	Filter-based	
	Multicast-based	
Topology	Centralized	
	Decentralized	
	Broker-based	DHT-based

This chapter is dedicated to providing the reader with a complete understanding of content-based pub/sub systems. Table 1 shows the classification of pub/sub systems based on the three different classification criteria. We will introduce the pub/sub systems based on the three classification methods.

The rest of this chapter is organized as follows. In Section 2, we present the pub-/sub systems based on subscription models. In Section 3, we present the content-based pub/sub systems based on routing models, and introduce multicast techniques. Section 4 details different pub/sub systems according to system topology, and discusses various challenges in modelling the systems. Meanwhile, we present a number of examples for the content-based pub/sub systems discussing their goals,

properties, strategies and classification. Finally in Section 5, we conclude this chapter with discussion about a number of open issues for building pub/sub systems.

2 Subscription Models

Different ways for specifying the subscribers' interest result in distinct variants of the pub/sub systems. The subscription models that appeared in the literature are characterized by their expressive power: highly expressive models offer subscribers the possibility to precisely match their interest, i.e., to receive only the events they are interested in [6]. In this section we briefly review the most popular pub/sub subscription models: topic-based model, content-based model and type-based model.

2.1 Topic-Based Systems

In the topic-based pub/sub systems, each event belongs to a certain topic (also referred to as group, channel or subject). Subscribers express their interest in a particular subject and they receive all the events published within that particular subject. Each topic corresponds to a logical channel ideally connecting each possible publisher to all interested subscribers. Network multicasting and diffusion trees can be used to disseminate events to interested subscribers. The topic-based model has been the solution adopted in all early pub/sub systems. Examples of systems that fall under this category are TIB/RV [72], SCRIBE [39], Bayeux [116], CORBA Notification Service [5], ISIS [18] and iBus [72] as well as commercial products Tibco [98] and Vitria [4].

Topic-based pub/sub systems take only coarse-grained subscriptions. The main drawback of the topic-based model is the very limited expressiveness it offers to subscribers. Consequently, a subscriber has to receive all events pertinent to a subject though the subscriber might be interested in only a subset of the events. In addition, topic-based systems provide limited choices of subscriptions. To address problems related to low expressiveness of topics, as indicated in [6], a number of solutions are exploited in pub/sub implementations. For example, the topic-based model is often extended to provide hierarchical organization of the topic space, instead of a simple flat structure such as in [13, 72]. A topic can be then defined as a sub-topic of an existing topic. Events matching the sub-topic will be delivered to all users subscribed to both the topic and sub-topic. Implementations also often include convenience operators, such as wildcard characters, for subscribing to more than one topic with a single subscription. Another method for enhancing expressiveness of the topic-based model is the filtered-topic variant [3, 5], where a further filtering phase is performed once the message is received based on the content of the message. Messages that do not satisfy the filter are not delivered to the application.

2.2 Content-Based Systems

In contrast to topic-based systems, content-based systems allow fine-grained subscriptions by enabling restrictions on the event content. In the content-base pub-/sub systems, notifications typically consist of a number of attribute/value pairs. A subscription may include an arbitrary number of attribute names and filtering criteria on their values. Only those events satisfying all the predicates are delivered to the subscriber. Hence, content-based systems increase subscription selectivity by allowing subscriptions have multiple dimensions [40]. Examples of content-based systems include Gryphon [1, 8, 73, 95], SIENA [26, 27], JEDI [36], LeSubscribe [80], Hermes [77, 78], Elvin [88], Rebeca [48, 49, 70], and CPAS [9]. In content-based pub/sub systems, the matching of subscriptions and publications is based on content and no prior knowledge is needed. Subscriptions in content-based pub/sub systems are more expressive. Subscribers express their interest by specifying conditions over the content of events they are interested in. In other words, a subscription is a request formed by a set of constraints composed through disjunction or conjunction operators. Possible constraints depend on the attribute type and the subscription language. Most subscription languages comprise equality and comparison operators as well as regular expressions. Therefore, these systems are more flexible and useful since subscribers can specify their interests more accurately using a set of predicates. The subscriber need not have to learn a set of topic names and their content before subscribing. The main challenge in building such systems is to develop an efficient matching algorithm that scales to millions of publications and subscriptions.

The complexity of the subscription language affects the complexity of matching operation. Therefore, it is not common to design subscription languages making requests more complex than those in conjunctive form such as those in [20, 23]. The work in [70] presents a complete specification of content-based subscription models. In content-based pub/sub systems, events are distinguished by the properties of the events instead of predefined criterion (i.e., topic name). Thus, the correspondence between publishers and subscribers is on a per-event basis. The difference with a filtered-topic model is that events that do not match a subscriber can be filtered out in any point in the system rather than on the receiver. For these reasons, the higher expressive power of content-based pub/sub comes at the cost of a higher overhead for calculating the set of interested subscribers for each event [26, 45].

2.3 Type-Based Systems

In type-based systems such as Echo [41], XMessage [92] and the work in [43, 44, 46], pub/sub variant events are objects belonging to a specific type, which can encapsulate attributes as well as methods. In a type-based subscription, the declaration of a desired type is the main discriminating attribute. That is, type-based pub/sub systems occupy the middle-ground between coarse-grained topic-based systems and fine-grained content-based systems. In terms of aforementioned models, a type-

based pub/sub system is in the middle, by giving a coarse-grained structure on events (like in topic-based) on which fine-grained constraints can be expressed over attributes (like in content-based). For example, in XMessages [92], a publisher and a subscriber can either interact directly with each other, exchanging events or use an XMessage channel that allows multiple publishers and listeners to communicate asynchronously. Publishers and subscribers initially use lookup table to get the reference of XMessage channel for building the connection. Then, they use SQL-like query to filter messages from this channel based on the content of messages. Thus, XMessage is type-based. Messages in XMessage may be defined as any XML content that needs to be transmitted between source and sink while events are also XML strings but have the typed fields.

3 Filter-Based and Multicast-Based Pub/Sub Systems

A main aspect in a pub/sub system is event dispatching in which matched events are routed to subscribers. According to routing solutions, the pub/sub systems can be largely categorized into two classes [24]: the filter-based approaches [8, 26, 27, 36, 73, 90] and the multicast-based approaches [8, 73, 83, 104]. In the filter-based approaches, routing decisions are made through successive content-based filtering at all nodes along the path from source to destination. Every pub/sub server in the path matches the event with remote subscriptions from other servers, and then forwards it towards directions that lead to matching subscriptions. This approach can achieve high efficiency, but at the cost of expensive subscription information management and high processing load at pub/sub servers.

In the multicast-based approach, certain multicast groups are determined before event transmission. For each event, one group is determined at the publisher, and the event is then multicasted to that group. In this method, some nodes in the routing path receive the events they are not interested in. The network efficiency of this approach is often highly sensitive to the data types and the distributions of events and subscriptions in the application.

Recently, much research effort has been devoted to the distributed pub/sub systems. The architecture designs include SIENA [26, 27], Gryphon [8, 73, 95], JEDI [36], Rebeca [48, 49, 70], Elvin [89], Ready [55], and Herald [22]. Most of these systems adopt the filter-based routing approach. For example, in JEDI, a hierarchical interconnection topology is proposed in which a server is only informed of subscriptions from servers in its sub-tree. Events are always forwarded up the hierarchy regardless of the interests in other parts of the network.

Pub/sub systems relying on multicast for event dispatching need content-based matching to discover the events and subscriptions. Event dispatching in a pub/sub system is similar to the traditional multicasting. The only difference is that the addresses of the message receivers are known in multicasting, while in pub/sub systems the receivers need to be determined by content-based matching. The matching problem has been studied for various data types and event schemes [8, 10, 47, 93].

Many pub/sub systems rely on multicast for notification service. That is, a publisher forwards events to many subscribers who subscribe to the publisher.

Many overlay-based multicast systems are proposed in the recent years, such as Narada [87], Bayeux [116], NICE [16] and Scribe [39]. Multicast protocols can be classified into centralized-based and distributed-based. Examples of centralized methods include HBM [84] and ALMI [74]. The distributed multicast implementations can be classified according to a number of criteria. We list the criteria and classifications in the following.

Collaboration techniques. There are two multicast architectures: P2P architectures and proxy (i.e., broker)-based architectures [35]. A P2P architecture pushes the functionality to nodes participating in the multicast group so that each node maintains the state of those groups that it is participating, while a proxy-based architecture lets an organization that provides value and services deploy proxies at strategic locations on the Internet. End node attaches itself to proxies near them, and receive data using plain unicast, or any available multicast media.

Distribution. Two types of multicast according to information distribution are tree-based and flooding [31] (including enhancement of flooding method such as gossip and random walking). The flooding approach such as CAN-based multicast [82] creates a separate overlay network per multicast group and leverages the routing information already maintained by a group's overlay to broadcast messages within the overlay. The tree approach, such as Scribe [39] and Bayeux [116], uses a single overlay and builds a spanning tree for each group, on which the multicast messages for the group are propagated.

Overlay network construction. Currently proposed multicast protocols are either built from scratch or based on an overlay network substrate such as Pastry [85], CAN [81] or Tapestry [113]. Examples of the former category include Narada [35] and NICE [16] and the latter include Scribe [39] based on Pastry, Bayuex [116] based on Tapestry and CAN-based multicast [82] based on CAN. According to the taxonomy of overlay multicast provided in [42], the former category can be further classified into two classes. (1) *Direct tree construction.* Members choose their parents from the members that they know. Protocols such as Yoid [51], BTP [59], Overcast [61] TBCP [68], HMTP [109], NICE [16] and ZIGZAG [99] use this way to construct trees. (2) *First mesh construction, second tree construction.* That is, first efficient meshes are constructed, then trees are constructed out of the meshes by certain routing algorithms. Such examples include Narada [35], Gossamer [32] and Delaunay triangulation [63]. The overlay network substrate category can be further classified into generalized hypercube such as Scribe [39] and Bayuex [116], and Cartesian Hyperspace such as CAN-based multicast [82] according to overlay network construction.

These proposals use two different techniques to design self-organizing multicast in order to improve the scalability of multicast. (1) *Neighbor mapping based on members' assigned addresses.* For example, CAN-based multicast [82] assigns logical addresses from cartesian coordinates on an n-dimensional torus. Delaunary Triangulations [63] assigns points to a plane and determines neighbor mappings corresponding to the Delaunay triangulation of the set of points. (2) *Organizing*

members into hierarchies of clusters. Nice [16] and Kudos [60] are such instances. Kudos constructs a two level hierarchy with a Narada like protocol at each level of the hierarchy. Banerjee et al. [16] constructs a multi-level hierarchy, which does not involve the use of a traditional routing protocol.

Building a broker-based network is the most common approach for designing a distributed notification service. Each broker communicates with its neighbor using for subscription and publication. A P2P overlay network for multicast is a logical application level network that is built on top of a general network layer like IP unicast. The nodes that are part of the overlay network can route messages between each other through the overlay network. There is an overhead associated with using a logical network for routing since the logical topology does not necessarily mirror the physical topology. However, more sophisticated routing algorithms can be used and deployed since routing is implemented at the application level.

4 Centralized and Distributed Pub/Sub Systems

Content-based pub/sub systems operate either in a centralized manner or a decentralized manner. In a centralized pub/sub system, a centralized server stores all the subscriptions, maps events to the subscriptions, and delivers events to the matched subscribers. The main component of this architecture is the event dispatcher. This component records all subscriptions in the system. When a certain event is published, the event dispatcher matches it to all subscriptions in the system. When the incoming event verifies a subscription, the event dispatcher sends a notification to the corresponding subscriber.

Keeping a global image of subscriptions makes it easy for the sever to find matched subscribers, avoiding unnecessary event delivery. However, the server can easily be overloaded in a large-scale system with thousands or even millions of clients. In addition, such systems suffer from the problem of single point of failure. Thus, centralized pub/sub systems cannot provide high scalability and reliability, which prevents it from being applied to large-scale applications such as global video-conferencing. A distributed pub/sub system [26, 101] is a promising alternative driven by a variety of large-scale communication applications. The main difficulty in building distributed content-based systems is the design of an efficient distributed matching algorithm. Distributed content-based systems can be further classified into broker-based and DHT-based. Broker-based systems such as SIENA [26] depend on a small number of trusted brokers connected by a high bandwidth network [96]. The broker-based systems improve the scalability and reliability of the centralized systems to a certain extend by distributing load among a number of brokers. However, a failure of one broker may lead to a large number of state transfer operations during recovery. Thus, the systems also may not provide very high scalability and reliability in a large-scale environment.

To address the problems, more and more pub/sub systems resort to DHTs [81, 85, 94, 113] due to their high scalability, reliability, fault-tolerance and self-organizing.

DHTs have successfully been used in a number of application domains, such as distributed file systems [7, 37, 71, 86]. Most pub/sub systems, such as Scribe [39], relying on DHTs are topic-based because of DHTs' mapping policy between data and nodes. Recently, much research has been conducted in building content-based pub/sub systems on top of P2P systems [9, 77, 78, 96, 97, 106–108, 111, 114, 115].

4.1 Centralized Pub/Sub Systems

Traditional centralized systems [3, 7, 21, 33, 54, 57, 58, 64, 76, 88] use a centralized server that stores all the subscriptions in the system. The centralized server maps events to the subscriptions, and delivers events to the matched subscribers who are interested in the events. As indicated in [96], centralized systems have the advantage of retaining a global image of the system at all times, enabling intelligent optimizations during the matching process [11, 21, 47, 64, 76]. For example, Fabret et al. [47] proposed data structures and application-specific caching policies and query processing to support high rates of subscriptions and events in the system. Specifically, they used the data structures including a set of indexes, a predicate bit vector and a cluster vector to achieve efficient event matching that is based on clustering and maximizes temporal and spatial locality. However, restrictions have to be placed on subscriptions such that they must contain at least one equality predicate, sacrificing flexibility and expressiveness of subscriptions. Major disadvantages of centralized systems are the lack of scalability and fault-tolerance.

Elvin [50, 88] is a "pure" notification service in which producers send notifications to the service, which in turn sends them to consumers. The notifications describe events using a set of named attributes of simple data types and consumers subscribe to a "class" of events using a boolean subscription expression. When a notification is received at the service from a producer, it is compared to the consumers' registered subscription expressions and forwarded to those whose expressions it satisfies. Once producers are freed of the responsibility to direct notifications, the determination of the significance of a state change becomes less important: they can notify any potentially interesting information, and rely on the notification service to discard notifications of no (current) interest to consumers. While large volumes of unused notifications may be useful from a user's perspective, they consume network bandwidth. To overcome this problem, Elvin includes a quenching mechanism which allows producers to discard unneeded notifications without sending them to the server. In order to support organization-wide notification, the implementation of the notification service must cater for many client applications. A single Elvin server can effectively service thousands of clients (producers or consumers) and evaluate hundreds of thousands of notifications per second on moderate hardware platforms. Further, additional servers can be configured in a federation, sharing the load of notification delivery, providing wide-area scalability and ensuring fault-tolerance in the face of individual server failures.

Hanson et al. [58] introduced an algorithm for finding the matching predicates that is more efficient than the standard algorithm when the number of predicates is large. The authors focus on equality and inequality predicates on totally ordered domains. This algorithm is well-suited for database rule systems, where predicate-testing speed is critical. A key component of the algorithm is the interval binary search tree. It is designed to allow efficient retrieval of all intervals such as range predicates that overlap a point, while allowing dynamic insertion and deletion of intervals. Later on, Hanson et al. [57] proposed a way to develop a scalable trigger system. It is achieved with a trigger cache to use main memory effectively, and a memory-conserving selection predicate index based on the use of unique expression formats called expression signatures. A key observation is that if a very large number of triggers are created, many will have the same structure, except for the appearance of different constant values. When a trigger is created, tuples are added to special relations created for expression signatures to hold the trigger's constants. These tables can be augmented with a database index or main-memory index structure to serve as a predicate index. The design presented also uses a number of types of concurrency to achieve scalability, including token (tuple)-level, condition-level, rule action-level, and data-level concurrency.

Farsite [7] is a serverless distributed file system that logically functions as a centralized file server but whose physical realization is dispersed among a network of untrusted desktop workstations. Farsite is intended to provide both the benefits of a central file server (a shared namespace, location transparent access, and reliable data storage) and the benefits of local desktop file systems (low cost, privacy from nosy sysadmins, and resistance to geographically localized faults). Farsite provides file availability and reliability through randomized replicated storage; it ensures the secrecy of file contents with cryptographic techniques; it maintains the integrity of file and directory data with a Byzantine-fault-tolerant protocol; it is designed to be scalable by using a distributed hint mechanism and delegation certificates for pathname translations; and it achieves good performance by locally caching file data, lazily propagating file updates, and varying the duration and granularity of content leases. Pub/sub matching algorithms work in two phases. First, predicates are matched and then matching subscriptions are derived. Based on Ashayer et al.'s [11] observation that the domain types over which predicates are defined are often of fixed enumerable cardinality in practice, Adya et al. developed a table-based look-up scheme for fast predicate evaluation that finds all matching predicates for each type with one table lookup. They further proposed two DBMS-based matching algorithms and compare the better one with a special purpose pub/sub matching algorithm implementation. Their work showed that for application scenarios that require large subscription workloads and process many events, a DBMS-based solution is not a feasible alternative.

Petrovic et al. proposed S-ToPSS semantic pub/sub system that provides semantic matching [76]. For instance, the system returns notifications about "vehicles" or "automobiles" to a client who is interested in a "car" based on the semantics of the terms. The authors described three approaches, each adding more extensive semantic capability to the matching algorithms. The first approach allows a

matching algorithm to match events and subscriptions that use semantically equivalent attributes-synonyms. The second approach uses additional knowledge about the relationships (beyond synonyms) between attributes and values to allow additional matches. More precisely, it uses a concept hierarchy that provides two kinds of relations: specialization and generalization. The third approach uses mapping functions which allow definitions of arbitrary relationships between schema and attribute values.

Liu et al. [64] pointed out that most existing pub/sub systems cannot capture uncertainty inherent to the information in either subscriptions or publications. In many situations, it is difficult to derive exact knowledge of subscriptions and publications. Moreover, especially in selective information dissemination applications, it is often more appropriate for a user to formulate his/her search requests or information offers in less precise terms, rather than defining a sharp limit. To address these problems, the authors proposed a new pub/sub model based on possibility theory and fuzzy set theory to process uncertainties for both subscriptions and publications.

Burcea et al. [21] identified the factors that affect the performance of a distributed pub/sub architecture supporting mobility; formalized mobility algorithms for distributed pub/sub systems and developed and evaluated optimizations that reduce the costs associated with supporting mobility in pub/sub systems. They focused on the "unicast" traffic generated to support mobile users, as opposed to the regular "multicast" traffic used for event dissemination to stationary clients.

4.2 Distributed Broker-Based Pub/Sub Systems

Content-based pub/sub allows fine-grained expressiveness of subscription, and thus is a more attractive solution for content dissemination. However, the design for content-based pub/sub systems is faced with two challenges that affect the performance of a content-based pub/sub network directly. The first challenge is the matching between subscriptions and events. Unlike the traditional multicast system where the addresses of destinations are known, the communication in content-based pub/sub systems is based on the content of event publications and subscriptions. Thus, it is important to match the subscribers' subscriptions and publishers' events to identify the addresses of destinations. After the destinations are determined, the events need to be routed to the destinations. As indicated in [25], traditional group-based muticast techniques [35] cannot be readily used to route event to all destinations. This is because content-based subscriptions are usually highly diversified, and different events may satisfy the interests of widely varying sets of servers. In the worst case, the number of such sets can be exponential to the network size (2^n where n is the number of servers), and it is impractical to build a multicast group for each such set. The second challenge is how to efficiently route the matched events to the destinations. Therefore, an architecture design should efficiently match an event to subscriptions and meanwhile reduce the nodes participating in routing. In the last few years, a variety of broker-based pub/sub systems have been proposed in order

to provide efficient and scalable pub/sub services. Broker-based systems depend on a small number of trusted brokers connected by a high bandwidth network. Brokers form an application level overlay and each broker stores subset of all subscriptions in the system. The overlay is managed by an administrator based on technical or administrative constraints. Examples of the broker-based pub/sub systems include SIENA [26–29], Gryphon [8, 73, 95], JEDI [36], Rebeca [48, 49, 70], Ready [55], Herald [22], MEDYM [25], Kyra [24], EDN [103] and link matching [15]. In the following, we present the details of the systems.

Kyra

To improve event routing efficiency, Cao and Singh [24] proposed Kyra routing scheme that uses content clustering to create multiple pub/sub networks each of which is responsible for a subset of the content space. The goal of Kyra is to reduce the implementation cost of the filter-based approach while still maintaining comparable network efficiency. Cao and Singh studied two major existing approaches for content-based pub/sub systems: filter-based approach, which performs content-based filtering on intermediate routing servers to dynamically guide routing decisions, and multicast-based approach, which delivers events through a few high-quality multicast groups that are pre-constructed to approximately match user interests. These approaches have different trade-offs in the routing quality achieved, the implementation cost and system load generated. The proposed Kyra carefully balanced these trade-offs by combining the advantages of content-based filtering and event space partitioning in the existing approaches to achieve better overall routing efficiency. The main idea is to construct multiple smaller routing networks, so that filter-based routing is implemented in each one with lower cost. Server load is reduced because each Kyra server is guaranteed to only participate in a small number of routing networks. This is achieved through strategically "moving" subscriptions between servers to improve content locality. Therefore, the effectiveness of Kyra is independent of data characteristics of pub/sub applications. Detailed simulation results show that Kyra significantly reduces the storage, processing and network traffic loads on pub/sub servers, while achieving network efficiency close to that of the filter-based approach. Kyra also balances routing load across the pub/sub service network.

SIENA

SIENA [26, 27] builds a symmetric spanning tree and each pub/sub server can be a publisher or subscriber. It selects the notifications that are of interest to clients and then delivers those notifications to the clients via access points. Mainly, SIENA addresses a key design challenge of maximizing expressiveness in the selection mechanism without sacrificing scalability of the delivery mechanism. SIENA focuses on the aspects that fundamentally affect scalability and expressiveness. In particular,

SIENA has data model for notifications, the covering relations that formally define the semantics of the data model, the distributed architectures, and the processing strategies to exploit the covering relations for optimizing the routing of notifications. This work shows that the hierarchical architecture is suitable with low densities of clients that subscribe (and unsubscribe) very frequently, whereas the P2P architecture performs better when the total cost of communication is dominated by notifications. In situations where there are high numbers of ignored notifications (i.e., notifications for which there are no subscribers), the P2P architecture is also superior to the hierarchical architecture.

Based on SIENA, Carzaniga et al. [29] proposed a forwarding algorithm in content-based pub/sub networks. Forwarding in such a network amounts to evaluating the predicates stored in a router's forwarding table in order to decide to which neighbor router the message should be sent. The proposed algorithm is based on the general structure proposed for Le Subscribe systems and takes advantage of their fixed or limited number of output interfaces. A forwarding table is conceptually a map from predicates to interfaces of neighbor nodes where a predicate is a disjunction of filters, each one being a conjunctions of elementary conditions over the attributes of a message. The design of a forwarding algorithm involves the design of a forwarding table and of its processing functions. The proposed forwarding algorithm accelerates the decision making in situations where there are large numbers of predicates and high volumes of messages.

Later on, Carzaniga et al. [28] further proposed a routing scheme that can propagate predicates and necessary topological information in order to maintain loop-free and possibly minimal forwarding paths for messages. The routing scheme uses a combination of a traditional broadcast protocol and a content-based routing protocol. This scheme consists of a content-based layer superimposed over a traditional broadcast layer. The broadcast layer handles each message as a broadcast message, while the content-based layer prunes the broadcast distribution paths, limiting the propagation of each message to only those nodes that advertised predicates matching the message. To implement this two-layer scheme, a router runs two distinct routing protocols: a broadcast routing protocol and a content-based routing protocol. The first protocol processes topological information and maintains the forwarding state necessary to send a message from each node to every other node. The second protocol processes predicates advertised by nodes, and maintains the forwarding state that is necessary to decide, for each router interface, whether a message matches the predicates advertised by any downstream node reachable through that interface. This second protocol is based on a dual "push-pull" mechanism that guarantees robust and timely propagation of content-based routing information.

Gryphon

Gryphon [8, 73, 95] organizes a pub/sub network into a single-source tree and proposes a link matching algorithm to forward events towards directions of matching subscriptions. In Gryphon, the flow of streams of events is described via an

information flow graph. The information flow graph specifies the selective delivery of events, the transformation of events, and the generation of derived events as a function of states computed from event histories. For this, Gryphon derives from and integrates the best features of distributed communications technology and database technology. The Gryphon approach augments the pub/sub paradigm with the following features: content-based subscription, in which events are selected by predicates on their content rather than by pre-assigned subject categories; event transformations, which convert events by projecting and applying functions to data in events; event stream interpretation, which allows sequences of events to be collapsed to a state and/or expanded back to a new sequence of events; and reflection, which allows system management through meta-events.

MEDYM

MEDYM [25] focuses on the problem of efficiently delivering events from the servers where they are published to the servers with matching subscriptions. In MEDYM, a matcher node matches an event to the subscriptions and obtains a destination list of the matched subscribers. Then, the event delivery message containing the destination list is routed through a dynamically generated dissemination tree with the help of topology knowledge. MEDYM does not rely on static overlay networks for event delivery. Instead, an event is matched against subscriptions early at the publishing server to identify destinations with matching subscriptions, and then sent to destination through a dynamically constructured multicast tree. This architecture achieves low computation cost in matching and high network efficiency in routing. MEDYM is distinguished by its dynamic multicast scheme to support the diversified routing need in pub/sub networks.

HYPER

HYPER [112] is a hybrid approach capable of minimizing both the matching and forwarding overhead within the pub/sub network and the delay experienced by clients receiving the content. It identifies a number of virtual groups by exploring common subscription interests among clients, and messages for each virtual group are only matched once at the group entry point. In addition, for each virtual group, the content delivery tree embedded in the underlying pub/sub network can benefit from short cutting forwarding-only paths.

EDN

EDN [103] partitions the content space subject to the restriction that the schema is fixed. For equality test, the attribute IDs and values are hashed to generate a key to locate the server managing it. For inequality tests, EDN uses an R-tree to decide

offline how to assign subscriptions to processors, and requires each processor to maintain a complete map of this assignment. This approach is limited to small-scale systems with a fixed set of subscriptions, and it is also unclear whether it works efficiently for high dimensional content space.

Rebeca

In Rebeca [49, 70], the notification service relies on a network of brokers, which forward notifications according to filter-based routing tables. The topology of the system is constrained to be an acyclic and connected graph for simplicity reasons. The edges are point-to-point connections, forming an overlay network. This model simplifies the implementation and reasoning about communication characteristics. As indicated in [105], the major advantage of these systems is that the routing tables can direct the flow of notifications to only interested nodes. Each broker maintains a routing table which includes content-based filters. When routing, a notification only goes down a link if it is matched by a corresponding filter. The simplest form of routing is simple routing: active filters are simply added to the routing tables with the link they originated from. However, this makes the routing table sizes grow linearly with the number of subscriptions. Two methods can be used to address this problem. The first improvement method is to check and combine filters that are equal. In the second improvement method, if no cover can be found in a given set of filters, merging can be used to create new filters that cover existing ones. Only the resulting merged filter is forwarded to neighbor brokers, where it covers and replaces the base filters.

Later, Fiege et al. [48] pointed out that many works on notification services and many concrete systems such as Siena [26, 27] and JEDI [36] have informal semantics. In addition, in these systems, subscriptions are selected out of all published notifications without distinguishing producers. Any further distinctions are necessarily hard-coded into the communicating components, mixing application structure and component implementation and thereby defeating the very feature of event-based systems of loose coupling. To provide methodological support building pub/sub systems, Fiege et al. presented Rebeca modular design and implementation of an event system which supports scopes and event mappings, two new and powerful structuring methods that facilitate engineering and coordination of components in pub/sub systems. They give a formal specification of scopes and event-mappings within a trace-based formalism adapted from temporal logic.

Link Matching

Banavar et al. [15] proposed a multicast protocol, called link matching, within a network of brokers in a content-based pub/sub system, thereby showing that content-based pub/sub can be deployed in large or geographically distributed settings. With this protocol, each broker partially matches events against subscribers at each hop

in the network of brokers to determine which brokers to send the message. Further, each broker forwards messages to its subscribers based on their subscriptions. Basically, the matching is based on sorting and organizing the subscriptions into a parallel search tree data structure, in which each subscription corresponds to a path from the root to a leaf. The matching operation is performed by following all those paths from the root to the leaves that are satisfied by the event. This data structure yields a scalable algorithm because it exploits the commonality between subscriptions as shared prefixes of paths from root to leaf. There is no additional information appended to the message headers in the method that match an event again all subscriptions. Further, at most one copy of a message is sent on each link. The disadvantages of the flooding approach are avoided as the message is only sent to brokers and clients needing the message.

Subscription Summaries

Triantafillou and Economides [101, 102] contributed the notion of subscription summaries, a mechanism appropriately compacting subscription information. They developed the associated data structures and matching algorithms. The proposed mechanism can handle event/subscription schemata that are rich in terms of their attribute types and powerful in terms of the allowed operations on them. The summarization structures of a broker's subscriptions and accompanying algorithms which operate on the summary structures match incoming events to the brokers with relevant subscriptions and maintain the subscriptions in the face of updates. The authors presented an algorithm to efficiently propagate subscription summaries to brokers. They also proposed an algorithm for the efficient distributed processing of incoming events, utilizing the propagated subscription summaries to route the events to brokers with matched subscriptions. They showed that the proposed mechanism is scalable with the bandwidth required to propagate subscriptions increasing only slightly even at huge-scales. The mechanism is significantly more efficient, up to orders of magnitude, depending on the scale, with respect to the bandwidth requirements for propagating subscriptions.

4.3 Distributed DHT-Based Pub/Sub Systems

DHT overlay networks [67, 69, 81, 85, 91, 94, 113] is a class of decentralized systems in the application level that partition ownership of a set of objects among participating nodes, and can efficiently route messages to the unique owner of any given object. Based on DHT overlay networks, a number of application level multicast systems have been proposed that can be used for topic-based pub/sub systems as well as content-based pub/sub systems. Examples of such systems include Scribe [39] based on Pastry, Bayeux [116] based on Tapestry and CAN-based multicast [82] based on CAN. Many content-based pub/sub systems based on DHTs

have been proposed [9, 14, 56, 77, 78, 96, 97, 100, 106–108, 111, 114, 115]. DHT-based pub/sub systems inherit the distinguished features of DHT overlay networks including scalability, efficiency, reliability, fault-tolerance, self-organizing from the underlying DHT infrastructure.

4.3.1 Introduction of DHT Overlay Networks

A P2P system consists of peers that act as servers as well as clients in order to make full use of resources. Because of dynamic connections and decentralization characteristic, P2P systems have certain mechanisms to ensure efficient connection and communication. Such mechanisms include those handling nodes join, leave and failure, allocating files to the nodes, etc. In the system, no node is more important than any other and the nodes can communicate with each other. Each node maintains the location information of some other nodes. A node can send message to a chosen node or broadcast the message to several other nodes. Based on overlay topology, P2P systems can be classified into unstructured P2P systems and DHT systems (i.e., structured P2P systems). Unstructured P2P overlay networks such as Gnutella [53] and Freenet [52] do not have strict control over the topologies, and they do not assign responsibility for data to specific nodes. On the contrary, DHT overlay networks have strictly controlled topologies and the data placement and lookup algorithms are precise.

DHT overlay networks is a class of decentralized systems in the application level that partition ownership of a set of objects among participating nodes, and can efficiently route messages to the unique owner of any given object. The DHT overlay networks include Chord [94], CAN [81], Tapestry [113], Pastry [85], Kademlia [69], Symphony [67] and Cycloid [91]. In DHT overlay networks, each object is stored at one or more nodes selected deterministically by a uniform hash function. Specifically, each object or node is assigned an ID (i.e., key) that is the hashed value of the object or node IP address using consistent hash function [62]. An object is stored in a node whose ID closest or immediately succeeds to the object's ID, which is called the object's owner. Though these DHT systems have great differences in implementation, they all support a hash-table interface of `put(key,value)` and `get(key)` either directly or indirectly. `put(key,value)` stores an object in its owner node, and `get(key)` retrieves the object. Queries for the object will be routed incrementally to the node based on the P2P routing algorithm. Each node maintains a routing table recording $O(\log N)$ neighbors in an overlay network with N hosts. These structured systems are highly scalable as it make very large systems feasible; lookups can be resolved in $\log N$ overlay routing hops. DHT overlay networks are widely used for data sharing application. Different from pub/sub systems, content-delivery DHT overlay networks distribute data among nodes, and efficiently forward a data request to the data owner. DHTs' efficient data location enables efficient multicast communication. In addition, DHT overlay networks make pub/sub systems resilient in a dynamic environment where nodes join and leave continuously.

Chord

Chord uses a one-dimensional circular key space. The node responsible for the key is the node whose identifier most closely follows the key numerically; that node is called the key's successor. Each node in Chord maintains two sets of neighbors: a successor list of k nodes that immediately follow it in the key space and a finger list of $O(\log n)$ nodes spaced exponentially around the key space. The ith entry of the finger list points to the node that is 2^i away from the present node in the key space, or to that node's successor if that node is not alive. Therefore, the finger list is always fully maintained without any null pointer. Routing correctness is achieved with these two neighbor lists. A `lookup(key)` is, except at the last step, forwarded to the node closest to, but not past, the key. The path length is $O(\log n)$ since every lookup halves the remaining distance to the destination.

Pastry and Tapestry

Plaxton et al. [79] developed perhaps the first routing algorithm that could be scalably used for P2P systems. Tapestry and Pastry use a variant of the algorithm. The approach of routing based on address prefixes, which can be viewed as a generalization of hypercube routing, is common to all theses schemes. The routing algorithm works by correcting a single digit at a time in the left-to-right order. If node with ID 12345 receives a lookup query with key 12456, which matches the first two digits, then the routing algorithm forwards the query to a node which matches the first three digits (e.g., node 12467). To do this, a node needs to have, as neighbors, nodes that match each prefix of its own identifier but differ in the next digit. For each prefix (or dimension), there are many such neighbors (e.g., node 12467 and node 12478 in the above case) since there is no restriction on the suffix, i.e., the rest bits right to the current bit. This is the crucial difference from the traditional hypercube connection pattern and provides the abundance in choosing neighbors and thus a high fault resilience to node absence or node failure. In addition to these neighbors, each node in Pastry also contains a leaf set, which is the set of $|L|$ numerically closest nodes (half smaller, half larger) to the present node's ID, and a neighborhood set which is the set of $|M|$ geographically closest nodes to the present node.

CAN

CAN chooses its keys from a d-dimensional toroidal space. Each node is identified by a binary string and is associated with a region of this key space, and its neighbors are the nodes that own the contiguous regions. Routing consists of a sequence of redirections, each forwarding a lookup to a neighbor that is closer to the key. CAN has a different performance profile than the other algorithms; nodes have

$O(d)$ neighbors and path-lengths are $O(dN^{1/d})$ hops. Note that when d=log$N$, CAN has $O(\log N)$ neighbors and $O(\log N)$ path length like the other algorithms.

4.3.2 Early DHT-Based Pub/Sub Systems

Most initially proposed DHT-based pub/sub systems such as Scribe [39] and Bayeux [116] are essentially topic-based pub/sub systems. They do not directly support content-based pub/sub services. The systems employ rendezvous node model. A subscription or an event is mapped to a rendezvous node using the DHT key allocation policy. The rendezvous node disseminates events to subscribers using application level multicast. Systems built on Chord and Pastry map each multicast group number to a specific node and then have it act as a rendezvous node for that group. Joining a group means to lookup the rendezvous node and have the nodes on the lookup path record the route back to the new members. Systems built on CAN have the rendezvous node act as an entry point to a distinct overlay network composed only of the group members.

Scribe

Scribe is a scalable application level multicast infrastructure built on top of Pastry. Scribe relies on Pastry to create and manage groups and to build efficient multicast trees for the dissemination of messages to each group. In addition, Scribe provides best-effort reliability guarantees. Scribe is fully decentralized: all decisions are based on local information, and each node has identical capabilities. Each node can act as a multicast source, a root of a multicast tree, a group member, a node within a multicast tree, and any sensible combination of the above. Any Scribe node may create a group; other nodes can then join the group, or multicast messages to all members of the group. Scribe provides best-effort delivery of multicast messages, and specifies no particular delivery order. A node can create, send messages to, and join many groups. Groups may have multiple sources of multicast messages and many members. Scribe can support simultaneously a large numbers of groups with a wide range of group sizes, and a high rate of membership turnover.

A node creates a group with groupId. The groupId can be the hash value of the group's textual name concatenated with its creator's name. The rendezvous point of a group is the owner of the groupId of the group. Scribe creates a multicast tree, rooted at the rendezvous point, to disseminate the multicast messages in the group. The multicast tree is created using a scheme similar to reverse path forwarding [38]. Specifically, a `join` message is routed by Pastry towards the group's rendezvous point. Each node along the route checks its list of groups to see if it is currently a forwarder; if so, it accepts the node as a child, adding it to the children table. Otherwise, it creates an entry for the group, and adds the source node as a child in the associated children table. It then becomes a

forwarder for the group by sending a `join` message to the next node along the route from the joining node to the rendezvous point. The original message from the source is then terminated. To enhance reliability, Scribe arranges each non-leaf node in the tree periodically sends a heartbeat message to its children. Furthermore, `forwardHandler(msg)` is invoked by Scribe before the node forwards a multicast message to make sure that parents can successfully forward the message.

SplitStream

SplitStream [30] is an application level multicast system built from Scribe for high-bandwidth data dissemination. Scribe works well only when the interior nodes are highly available. It poses a problem for application level multicast in P2P cooperative environments where peers contribute resources in exchange for using the service. SplitStream addresses this problem by striping the content across a forest of interior-node-disjoint multicast trees that distributes the forwarding load among all participating peers. For example, it is possible to construct efficient SplitStream forests in which each peer contributes only as much forwarding bandwidth as it receives. Furthermore, with appropriate content encodings, SplitStream is highly resilient to failures because a node failure causes the loss of a single stripe on average. To balance forwarding load over participating nodes with heterogeneous bandwidth constraints, SplitStream splits content into k stripes each of which corresponds to a Scribe multicast tree.

Bayeux

Bayeux [116] is another architecture for application layer multicast, where the end-hosts are organized into a hierarchy as defined by the Tapestry overlay location and routing system [113]. Similar to Scribe, Bayeux assigns a unique ID to each topic by using the tuple that uniquely names a multicast session (i.e., topic), and a secure one-way hashing function (such as SHA-1 [62]) to map it into a 160 bit identifier. The owner of the ID becomes the rendezvous point for this topic and the root node of the multicast tree. Clients that want to join a session must know the unique tuple that identifies that session. They can then perform the same operations to generate the file name, and query for it using Tapestry. For each topic, a multicast tree that is rooted at the rendezvous point is created by combining the paths from each subscriber to the rendezvous point. A level of the hierarchy is defined by a set of hosts that share a common suffix in their host IDs. These searches result in the session root node receiving a message from each interested listener, allowing it to perform the required membership operations. The events associated with the topic are disseminated along the corresponding multicast tree starting from the root. Such a technique was proposed by Plaxton et al. [79] for locating and routing to named objects in a network. Therefore, hosts in Bayeux maintain $O(b \log_b N)$ state

and end-to-end overlay paths have $O(\log_b N)$ application level hops (b is a small constant).

CAN-based Multicast

CAN defines a virtual d-dimensional Cartesian coordinate space, and each overlay host owns a part of this space. Ratnasamy et al. [82] leveraged the scalable structure of CAN to define an application layer multicast scheme, in which hosts maintain $O(d)$ state and the path lengths are $O(dN^{1/d})$ application level hops, where N is the number of hosts in the network. The CAN-based multicast scheme is capable of scaling to large group size without restricting the service model to a single source. Extending the CAN framework to support multicast comes at trivial additional cost, and obviates the need for a multicast routing algorithm because of the structured nature of CAN topologies. Given the deployment of a distributed infrastructure such as a CAN, the CAN-based multicast scheme offers the dual advantages of simplicity and scalability.

Reach

Reach [75] employs the rendezvous model, in which each node serves as a rendezvous point for those subscriptions with suffix matching the node's identifier. At a high level, the rendezvous service is the means by which subscriptions are stored in the network, and by which published messages are directed to "find" the subscriptions they match. This rendezvous node is then an entry point into a "subset tree" of nodes hosting other, more general subscriptions, and thus to which this message should also be routed. This tree is implemented in such a way that it offers join-and-leave flexibility and maximum efficiency as the nodes in the tree are nearby neighbors in the overlay. Reach employs a semantic overlay network and uses a Hamming-distance based routing scheme. Hamming-based encoding scheme defines an identifier hierarchy in which, a parent identifier contains at least all the attributes of a child identifier. This hierarchy is a fundamental concept in Reach and is the basis for content-based multicasting.

HOMED

HOMED [34] maintains a semantic overlay where each node's identifier is derived from its subscriptions. HOMED is suitable for large-scale pub/sub. HOMED prefers a mesh-like structure rather than a tree for a reliable and adaptive event dissemination tree. Moreover, it arranges a node to neighbor with the nodes whose interests are similar to its interest in the overlay network so that only interested nodes participate in disseminating an event. To ease construction and routing, HOMED organizes the overlay network based on the interest digest of each node rather than the complex

selection predicate. HOMED can be used not only for flexible topic or type-based systems by nature, but also as a routing substrate for highly selective content-based systems. In HOMED, an event is delivered along the path of a binomial tree. Also, the subscribe/unsubscribe overhead is limited to $O(\log N)$.

4.3.3 DHT and Content Based Pub/Sub Systems

DHT systems are oblivious to the content of a file and use a uniform hash function on files' keys to distribute the files among the different peers. A file's key is the file name or the keyword that can distinguish the file. Therefore, on the one hand, DHTs provide exact-matching service. On the other hand, equality predicates and range predicates are expected when specifying subscriptions in pub/sub systems. Thus, to use DHT substrates for content-based pub/sub systems, a mechanism is needed that helps to distribute subscriptions and events among DHT nodes based on data content.

To tackle this problem, the works in [14, 56, 100] regard a subscription as a number of attributes and ranges. These works use each of the attributes and range constraints as a key to map the subscription to a number of overlay nodes. The single individual mapping for each attribute and value may lead to low scalability, especially when a subscription has many attributes and value ranges. To resolve the problem, some works [9, 77, 78, 96, 106–108, 111, 114, 115] use a scheme to derive a key or a small number of keys from a subscription for the mapping, while other works [97] combine the filter-based routing in broker-based model with the routing in DHT model.

Chord-based Systems

Triantafillou et al. [100] introduced one of the first content-based approximations where Chord DHT is employed as reliable routing infrastructure, so that they do not build a specific pub/sub overlay. The system distributes subscriptions on the Chord nodes based on the keys produced by hashing the attribute and its values. To do so, they employ the rendezvous model, in which a subscription is stored in a number of nodes based on the keys. If the subscription specifies a range over an attribute, the subscription would be stored on a number of nodes by hashing the attribute and each of its possible values within this range. Such systems suffer from the lack of scalability on high-dimensional contexts where a subscription has many attributes and values. The main drawback is that subscription installation and update are expensive due to the large number of nodes and messages potentially involved.

Later, Baldoni et al. [14] proposed a similar approach but, in this case, they used a particular mapping of events and subscriptions to keys from the DHT key space, instead of per-attribute mappings. They introduced a general form mapping that does not depend on the stored subscriptions which is called stateless mapping. It eliminates the need to propagate the knowledge about currently stored

subscriptions. Specifically, the authors proposed three different methods for mapping pub/sub subscriptions and events to overlay keys: attribute-split, key space-split and selective-attribute. Furthermore, in order to increase the efficiency of the proposed solution, they proposed to enrich the existing overlay networks with one-to-many primitives, as well as to extend the infrastructure with notification buffering and range discretization capabilities.

Meghdoot

Meghdoot [56] is designed to adapt to highly skewed data sets, which is typical of real applications. Built upon CAN, Meghdoot adapts content-based pub/sub systems to DHT networks in order to provide scalable content delivery mechanisms while maintaining the decoupling between the publishers and the subscribers. Meghdoot stores subscriptions in a zone according to the coordinate determined by event attribute values. To do this, Meghdoot extends the traditional 1D-dimensional CAN to 2D-dimension CAN and relaxes the restrictions on subscriptions. A subscription defines a rectangular region in the D-attribute content space bounded by the minimal and maximal value specified. Unspecified attributes take the whole value range. The hyperrectangle is projected to a point in a 2D-dimension CAN constructed from the minimal and maximal values of the D-dimension rectangle. An event is then mapped to a rectangle in the 2D space, and the mapping is performed in a manner such that the rectangle covers all subscription points relevant to the event. This novel approach reduces the subscription matching problem into a range query operation in CAN. Considering skewed distributions of subscriptions and events in a real application, Meghdoot addresses the load balancing issue by zone splitting and zone replication. However, though it can support range subscriptions, it is still confined to numerical attributes and also can not handle skewed distributions efficiently. In addition, Meghdoot requires that the overlay dimension must be proportional to the number of event attributes, which may lead to very high DHT key space.

Scribe-based System

Tam et al. [96] proposed a content-based pub/sub system built from Scribe. In the approach, topics are automatically detected from the content of subscriptions and publications through the use of a schema, which is a set of guidelines for selecting topics. The schema is application-specific and can be provided by the application designer after some statistical analysis. The schemas are similar to database schemas used in RDBMS. This approach significantly increases the expressiveness of subscriptions compared to purely topic-based systems. However, this scheme does not fully provide the query semantics of a traditional content-based system. Queries are not completely free-form but must adhere to a predefined template. The system places some restrictions on subscriptions and thus sacrifices expressiveness in subscriptions. Moreover, issues of fault-tolerance in subscription storage have yet to

be explored in the system, although fault-tolerance in DHT routing and multicast routing can be transparently handled by Pastry and Scribe, respectively.

Hermes

Hermes [77, 78] is an event-based middleware architecture that follows a type- and attribute-based pub/sub model. Hermes uses Pastry DHT routing substrate for installing content based filters close to the publishers. The Cambridge Event Architecture (CEA) [12, 66] is an event-based middleware that supports proper event typing. Hermes follows its approach by associating every event and subscription with an event type that is type-checked at runtime. A scalable routing algorithm using an overlay routing network is developed that avoids global broadcasts by creating rendezvous nodes. Fault-tolerance mechanisms that can cope with different kinds of failures in the middleware are integrated with the routing algorithm, resulting in a scalable and robust system.

CPAS

Considering node cooperation and multi-attribute feature of subscription, Ahullò et al. [9] proposed CPAS, which employs the rendezvous model in order to meet both, events and subscriptions. The system defines a certain set of nodes from the DHT as rendezvous nodes. The rendezvous nodes are responsible of matching events against subscriptions and starting the notification process. Additionally, these rendezvous nodes are selected deterministically, so that the node in DHT responsible for a given key then becomes the rendezvous node. Due to the DHT properties, the chosen node will be globally agreed upon by all nodes. Thus, every node can use the P2P routing substrate to send messages to this rendezvous node. The rendezvous model enables the system to avoid the construction of a specific overlay to disseminate events in a proper way. CAPS employs an order preserving hash function (OPHF) to deterministically map conjunctive predicates from every subscription into a set of keys and every event into a key, in order to deal naturally with multi-dimensional domains, and multiple sources cooperating within the system.

Ferry

Ferry [114] provides a preliminary study of exploiting the embedded trees in DHTs to deliver events. It is designed based on Chord and aims to host any and many content-based pub/sub services. That is, any pub/sub service with a unique scheme can run on top of Ferry, and multiple pub/sub services can coexist on top of Ferry. For each pub/sub service, Ferry does not need to maintain or dynamically generate any dissemination tree. Instead, it exploits the embedded trees in the underlying DHT to deliver events. Ferry can support a pub/sub scheme with a large number of

event attributes. Specifically, a subscriber chooses an attribute from all attributes of a subscription whose consistent hash value is equal to or most immediately precede the subscriber's ID. It then maps the subscription to a rendezvous node based on the consistent hash value. Thus, a tree is formed by the underlying DHT links thereby imposing no additional construction of maintenance cost. When a node wants to publish an event, the event is first directed to the rendezvous node where the event is matched to the subscriptions. Once those subscriptions matching the event are identified, the event is then delivered to the corresponding subscribers by using Ferry's event delivery algorithm. In the delivery algorithm, all the event delivery messages to those subscribers who share common ancestor nodes on the tree are aggregated into one single message along the path from the root node to their lowest common ancestor node. To deal with skewed distribution of subscriptions and events, Ferry uses one-hop subscription push and attribute partitioning to balance load. In the one-hop subscription push algorithm, a rendezvous node pushes the subscriptions corresponding to one of the nodes' neighbors in its routing table to the neighbor. In the attribute partitioning algorithm, a value range is partitioned into a number of ranges.

Eferry and HyperSub

Eferry [108], HyperSub [107] and the work in [106] are enhanced systems based on Ferry. The objective of Eferry [108] is to ensure an appropriate amount of rendezvous point nodes in the system and load distribution among them. Eferry achieves this goal with three methods: (1) a novel subscription installation algorithm to choose certain rendezvous point nodes which are evenly distributed in the ID space. (2) ID space partitioning and attributes grouping schemes designed to flexibly adjust the amount of rendezvous point nodes as well as their load. (3) a self-adaptive load balancing algorithm with dynamic ID space split-merge to make sure that no node is unduly loaded. HyperSub [107] and the work in [106] use a locality-preserving hashing mechanism to partition and map the content space to nodes. Subscriptions and events are mapped to the corresponding nodes for efficiently matching. The systems have an efficient event delivery algorithm which exploits the embedded trees in the underlying DHT to deliver events to the corresponding subscribers. In addition, the systems have light-weighted load balancing mechanisms to adjust the load among peers. The load balancing mechanism includes space mapping rotation, content space transformation and dynamic subscriptions migration algorithms.

PRESS

PRESS [115] distinguishes itself from Ferry by proposing a new architecture that aims to preserve subscription locality in subscription management, minimize event matching load, balance load across nodes, and offer efficient and scalable

event delivery. The framework of PRESS is based on the three key mechanisms: Subscription Organization Mechanism (SOM), Publication and Matching Mechanism (EPMM) and Event Delivery Mechanism (EDM). SOM uses K-D tree techniques [17] to organize subscriptions in a hierarchical tree manner, and stores the subscriptions only on leaf nodes. SOM preserves locality of subscriptions, i.e., similar/relevant subscriptions are stored on a (or a small number of adjacent) leaf node(s). Each leaf node is responsible for roughly the same number of subscriptions, ensuring load balance across leaf nodes. SOM layers the tree structure on top of a DHT, by which each tree node is hosted by a DHT node and the tree inherits fault-resilience and self-organizing properties of the underlying DHT. Subscription installation is a process of tree navigation from the tree root to the corresponding leaf node(s). The subscription installation may involve multiple overlay hops since the tree spans the DHT overlay, thereby incurring high latency. In addition, every installation goes through the root, creating a potential bottleneck. Hence, PRESS uses K-D tree-lookaside cache at client/subscriber side to alleviate the problems. EPMM allows event publishers to publish an event along the K-D tree to the leaf node that stores the subscriptions relevant to the event. The leaf node then matches the event to the subscriptions and starts delivering the event to the matched subscribers. Similar to subscription installation, event publication could incur high publication latency and create a potential bottleneck on the tree root node. To alleviate the problems, the K-D tree-lookaside cache is employed at the client/publisher side. EDM is virtually maintenance-free. It exploits embedded trees inherent in the underlying DHT to deliver events, thereby eliminating the cost of multicast-tree construction and maintenance. After a leaf node matches an event to the subscriptions stored on it, the leaf node multicasts the event through the corresponding DHT links of its DHT host node. The event is then disseminated along the embedded tree rooted at the DHT node hosting the leaf node, and finally reaches each subscriber. EDM aggregates messages along event dissemination paths, thus reducing the number of event delivery messages and bandwidth consumption. Moreover, exploiting DHT links for event delivery, EDM has three major advantages: (1) The underlying DHT maintenance messages could be piggybacked onto the event delivery messages to reduce the DHT maintenance cost. (2) Proximity neighbor selection in the underlying DHT, as a means of improving routing performance, makes event dissemination along the embedded tree proximity-aware, achieving efficient event delivery performance. (3) The fault-tolerance and self-organizing nature of DHT overlays makes event delivery along the DHT links resilient to node/link failures.

Brushwood-based System

The content-based pub/sub model has been adopted by many services to deliver data between distributed users based on application-specific semantics. Two key issues in such systems, the semantic expressiveness of content matching and the scalability of the matching mechanism, are often found to be in conflict due to the complexity associated with content matching. To address this problem,

Zhang et al. [111] presented a content-based pub/sub architecture based on Brushwood P2P matching trees [110]. The authors indicated that the content-based systems have more complex subscription structures that impede the workload partition than topic-based systems due to three reasons. The first reason is the high dimensionality of the content space in which a setting involves a large number of attributes. The second reason is type flexibility which means that attributes may have various types that require different filtering tests. The third reason is skewed data distribution, which could create a load imbalance in the system that throttles the scalability. The system achieves scalability by partitioning the responsibility of event matching to self-organized peers while allowing customizable matching functionalities. Specifically, the authors proposed a P2P architecture that achieves high scalability and generality. The architecture addresses the expressiveness problem with a modular matching tree structure. This tree organizes the subscriptions into hierarchical groups based on their similarity. It supports flexible schemas and multiple attribute types in subscriptions and events, and allows customization of new attributes and filtering types. This matching tree is distributed in a P2P system where each peer processor manages a small fragment of the tree. They maintain the distributed tree by peer-wise communications without global coordination. Events can enter the system from any processor. A decentralized tree navigation algorithm is used to forward the events to those tree fragments that may contain matching subscriptions. In experiments, the proposed system demonstrates high scalability. Specifically, the distributed event matching only visits a small number of processors, processors maintain a small amount of state about peers, and the workload is well-balanced across the processor set.

Combination of Rebeca and Chord

The system proposed in [97] is another content-based pub/sub system built on top of a dynamic Chord P2P overlay network. Both filter updates (e.g., due to subscribing and un-subscribing) and event routing use a broadcasting algorithm. The main advantage of the proposed system is the unique combination of the high expressiveness of content-based filters in Rebeca and the scalability and fault tolerance of Chord P2P system. It helps to remove the single bottleneck and point-of-failure of using only one tree for notifications and filter updates. To avoid introducing routing cycles within a more general redundant graph, the system selects for each notification a spanning subtree of the entire graph. However, to balance the network congestion and reduce single points of failure, the system uses a different tree for every broker. That is, each broker is at the root of its own distinct tree for delivering a published notification. This allows the system to use a generalization of the pub/sub routing strategy. During routing, the system provides a test to assure forwarding is only along those edges which are in the subtree. To provide the routing algorithm with an understanding of how to select the edges for a subtree, the system incorporates a topology component. Furthermore, the system has two components that maintain the structure of the graph and the filters to enhance system robustness when brokers

change and fail. Separating the components ensures that the network self-organizes to maintain the optimal topology and can survive simultaneous failure of up to half of its nodes. Because the system delivers via binomial trees, message delivery paths are logarithmically bounded.

Table 2 illustrates a survey of current pub/sub systems based on the classifications.

Table 2 Survey of pub/sub systems.

Centralized systems	**Distributed systems**			
	Broker-based		**DHT-based**	
Content-based	**Topic-based**	**Content-based**	**Topic-based**	**Content-based**
TIB/RV [72]	TIB/RV [72]	SIENA [26–29]	Scribe [39]	Meghdoot [56]
CORBA-NS [5]	JEDI [36]	Gryphon [8, 73, 95]	Bayuex [116]	Hermes [77, 78]
Narada [87]		Rebeca [48, 49, 70]	NICE [16]	CPAS [9]
Elvin [50]		Kyra [24]	SplitStream [30]	Ferry [114]
Farsite [7]		MEDYM [25]	Reach [75]	Eferry [108]
S-ToPSS [76]		Ready [55]	HOMED [34]	HyperSub [107]
JMS [3]		Herald [22]		PRESS [115]
		EDN [103]		Brushwood-based [111]

5 Summary and Challenges

In the last years, a growing attention has been paid to the pub/sub communication paradigm as a means for disseminating events through distributed systems on wide-area networks. This chapter has provided a detailed introduction of pub/sub systems, and has examined all aspects of pub/sub systems including their goals, properties, strategies and classification. To survey and compare different pub/sub systems, we introduced three classification criteria: subscription model, routing and topology. Based on the subscription model, the pub/sub systems can be classified into topic-based, content-based and type-based. Based on routing, the pub/sub systems can be classified into filter-based and multicast-based. Based on topology, the pub/sub systems can be classified into centralized-based and distributed-based, which is further classified into broker-based and DHT-based. A comprehensive review of research works of pub/sub systems focusing on distributed networks has been presented, along with an in-depth discussion of their pros and cons.

We conclude this chapter with discussion about a number of open issues for building pub/sub systems.

- **Tradeoff between the accuracy and efficiency.** Topic-based pub/sub systems cannot provide high accuracy since a node may receive events it is not interested

in. On the other hand, highly fined-grained content-based systems lead to high cost for mapping between subscriptions and events as well as node communication. A mechanism that can combine the advantages of both types while overcoming their drawbacks is expected.

- **Proximity.** Mismatch between logical proximity abstraction derived from overlay networks, and physical proximity information in reality is a major obstacle for the deployment and performance optimization issues for pub/sub applications. Most current pub/sub systems fail to take into account the proximity to reduce the node communication cost.
- **Heterogeneity.** With the increasing emergence of various end devices equipped with networking capability, coupled with the diverse network technology development, the heterogeneity of participating nodes of a practical pub/sub system is pervasive. Their distinct properties, including computing ability, differ greatly and deserve serious consideration for the construction of a real efficient widely-deployed application. Most current pub/sub system considering load balance fail to take into account the heterogeneity.
- **Mobility.** With the increasing popularity of wireless communication networks and mobile handheld devices, it becomes an inevitable trend that the pub/sub systems will be applied to the mobile wireless networks. Currently, there are few works devoted to the development of a pub/sub system in a mobile environment. One challenge is how to deal with node mobility.

References

1. Gryphon web site. http://www.research.ibm.com/gryphon/.
2. Publish/subscribe. http://en.wikipedia.org/wiki/Publish/subscribe.
3. Sun microsystems. Java Message Service API, Sun Microsystems. 2003.
4. Vitria. http://www.vitria.com/.
5. Object management group. corba notification service specification, version 1.0.1. omg document formal/2002-08-04, 2002.
6. S. Scipioni, A. Corsaro, and L. Querzoni. Quality of service in publish/subscribe. Technical report, Università di Roma La “Sapienza”, 2006.
7. A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceus, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. FARSITE: Federated, avaiable, and reliable storage for an incompletely trusted environment. In *Proc. of the Fifth USENIX Symposium on Operating Systems Design and Implementation*, December 2002.
8. M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *Proc. of the Eighteenth ACM Symposium on Principles of Distributed Computing*, 1999.
9. J. P. Ahullò, P. G. Lòpez, and Antonio F. G. Skarmeta. Caps: Content-based publish/subscribe services for peer-to-peer systems. In *Proceedings of 2nd International Conference on Distributed Event-Based Systems (DEBS)*, July 2008.
10. M. Altinel and M. Franklin. Efficient Filtering of XML Documents for Selective Dissemination of Information. *VLDB Journal*, pages 53–64, 2000.
11. G. Ashayer, H. K. Y. Leung, and H. A. Jacobsen. Predicate matching and subscription matching in publish/subscribe systems. In *Proc. of Workshop on Distributed Event-Based Systems (DEBS)*, pages 539–546, 2002.

12. J. Bacon, A. Hombrecher, C. Ma, K. Moody, and W. Yao. Event storage and federation using odmg. In *Proc. of the 9th Int. Workshop on Persistent Object Systems (POS9)*, pages 265–281, Sept. 2000.
13. S. Baehni, P. Th. Eugster, and R. Guerraoui. Data-aware multicast. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN)*, pages 233–242, 2004.
14. R. Baldoni, C. Marchetti, A. Virgillito, and R. Vitenberg. Content-based publish-subscribe over structured overlay networks. In *Proc. ICDCS*, pages 437–446, July 2005.
15. G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *Proceedings of the 19th IEEE ICDCS*, pages 262–272, June 1999.
16. S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proc. of ACM SIGCOMM'02*, pages 205–217, 2002.
17. J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.
18. K. P. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):36–53, Dec 1993.
19. K. P. Birman and T. A. Joseph. Exploiting virtual synchrony in distributed systems. *Operating Systems Review*, pages 123–138, 1987.
20. S. Bittner and A. Hinze. On the benefits of non-canonical filtering in publish/subscribe systems. In *Proceedings of the International Workshop on Distributed Event-Based Systems (ICDCS/DEBS)*, 2005.
21. I. Burcea, V. Muthusamy, M. Petrovic, H. A. Jacobsen, and E. de Lara. Disconnected operations in publish/subscribe. *Proc. of IEEE Mobile Data Management*, 2004.
22. L. F. Cabrera, M. Jones, and M. Theimer. Herald: Achieving a global event notification service. In *Proc. of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, May 2001.
23. A. Campailla, S. Chaki, E. M. Clarke, S. Jha, and H. Veith. Efficient filtering in publishsubscribe systems using binary decision diagrams. In *Proceedings of The International Conference on Software Engineering*, pages 443–452, 2001.
24. F. Cao and J. P. Singh. Efficient event routing in content-based publish/subscribe service networks. In *Proceedings of INFOCOM*, volume 2, pages 929–940, March 2004.
25. F. Cao and J. P. Singh. MEDYM: match-early and dynamic multicast for content-based publish-subscribe service networks. In *Proceedings of the 4th international workshop on distributed event-based systems*, pages 370–376, 2005.
26. A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Achieving scalability and expressiveness in an Internet-scale event notification service. In *Proc. of ACM Symp. on Principles of Distributed Computing (PODC)*, pages 219–227, 2000.
27. A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.
28. A. Carzaniga, M. J. Rutherford, and A. L. Wolf. A routing scheme for content-based networking. In *Proceedings of IEEE INFOCOM*, pages 918–928, March 2004.
29. A. Carzaniga and A. L. Wolf. Forwarding in a content-based network. In *Proceedings of ACM SIGCOMM*, pages 163–174, 2003.
30. M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in cooperative environments. In *Proc. of the 19th ACM Symp. on Operating Systems Principles (SOSP-19)*, October 2003.
31. M. Castro, M. B. Jones, A-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman. An evaluation of scalable application-level multicast built using peer-to-peer overlays. In *Proc. of IEEE Conference on Computer Communications (INFOCOM'03)*, March 2003.
32. Y. Chawathe. Scattercast: An architecture for internet broadcast distribution as an infrastructure service. ph.d. thesis. Technical report, University of California, Berkeley, 2000.
33. J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A scalable continuous query system for Internet databases. In *Proceedings of the 2000 ACM SIGMOD*, pages 379–390, 2000.

34. Y Choi, K. Park, and D. Park. HOMED: a peer-to-peer overlay architecture for large-scale content-based publish/subscribe systems. In *Proceedings of the third international workshop on distributed event-based systems (DEBS)*, pages 20–25, May 2004.
35. Y. Chu, S. Rao, and H. Zhang. A case for end system multicast. In *Proceedings of ACM SIGMETRICS'2000*, January 2000.
36. G. Cugola, E. D. Nitto, and A. Fuggetta. The JEDI Event-based Infrastructure and its Application to the Development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, 2001.
37. F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stocia. Wide-area cooperative storage with CFS. In *Proc. of the 18th ACM Symp. on Operating Systems Principles (SOSP-18)*, October 2001.
38. Y. K. Dalal and R. Metcalfe. Reverse path forwarding of broadcast packets. *Communications of the ACM*, 21(12):1040–1048, Dec. 1978.
39. P. Druschel, M. Castro, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. In *IEEE Journal on Selected Areas in Communications*, 2002.
40. V. S. W. Eide, F. Eliassen, O. Lysne, and O. Granmo. Extending content-based publish/-subscribe systems with multicast support. Technical report, Simula Research Laboratory, 2003.
41. G. Eisenhauer. The ECho event delivery system. Technical Report GITCC-99-08, College of Computing, Georgia Institute of Technology, June 1999. http://www.cc.gatech.edu/tech reports.
42. A. El-Sayed, V. Roca, I. Rhone-Alpes, and L. Mathy. A survey of proposals for an alternative group communication service. *IEEE Network magazine.*, 2003.
43. P. T. Eugster and R. Guerraoui. Content-based publish/subscribe with strucutural reflection. In *Proc. of the 6th USENIX Conf. on Object-Oriented Technologies and Systems (COOTS01)*, Jan 2001.
44. P. T. Eugster, R. Guerraoui, and J. Sventek. Type-based publish/subscribe. Technical report, EPFL, Lausanne, Switzerland, June 2000.
45. P. Th. Eugster, P. Felber, R. Guerraoui, and S. B. Handurukande. Event Systems: How to Have Your Cake and Eat It Too. In *Proceedings of the International Workshop on Distributed Event-Based Systems (DEBS)*, 2002.
46. P. Th. Eugster, R. Guerraoui, and Ch. H. Damm. On Objects and Events. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications*, 2001.
47. F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In *Proceedings of ACM SIGMOD*, volume 30, pages 115–126, 2001.
48. L. Fiege, G. Mühl, and F. Gärtner. Modular event-based systems. *The Knowledge Engineering Review*, 17(4):55–85.
49. L. Fiege, G. Mühl, and F. Gärtner. A Modular Approach to Building Event-Based Systems. In *Proceedings of the ACM Symposium on Applied Computing*, 2002.
50. G. Fitzpatrick, T. Mansfield et al. Instrumenting and Augmenting the Workaday World with a Generic Notification Service called Elvin. In *Proc. of European Conference on Computer Supported Cooperative Work (ECSCW)*, 1999.
51. P. Francis. Yoid: Your own internet distribution. Technical report, ACIRI, 2000. http://www.aciri.org/yoid/.
52. The freenet home page. freenet.sourceforge.net, www.freenetproject.org.
53. Gnutella home page. http://www.gnutella.com.
54. J. Gough and G. Smith. Efficient recognition of events in a distributed system. In *Proc. of the 18th Australasian Computer Science Conference*, 1995.
55. R. Gruber, B. Krishnamurthy, and E. Panagos. The architecture of the READY event notification service. In *Proceedings of the 19th Middleware Workshop*, 1999.

56. A. Gupta, O. D. Sahin, D. Agrawal, and A. E. Abbadi. Meghdoot: content-based publish/-subscribe over P2P networks. In *Proceedings of the 5th International middleware conference of ACM/IFIP/USENIX*, pages 370–376, Oct. 2005.
57. E. N. Hanson, C. Carnes, L. Huang, M. Konyala, L. Noronha S. Parthasarathy, J. B. Park, and A. Vernon. Scalable trigger processing. In *Proceedings of the 15th ICDE*, pages 266–275, 1999.
58. E. N. Hanson, M. Chaabouni, C.-H. Kim, and Y.-W. Wang. A predicate matching algorithm for database rule systems. In *Proc. of SIGMOD*, 1990.
59. D. A. Helder and S. Jamin. End-host multicast communication using switch-tree protocols. In *In Proceedings of the Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems (GP2PC)*, 2002.
60. S. Jain, R. Mahajan, D. Wetherall, G. Borriello, and S. D. Gribble. Scalable self-organizing overlays. technical report uw-cse 02-02-02. Technical report, University of Washington, 2002.
61. J. Jannotti, d. Gifford, K. Johnson, and M. Kaashoek. Overcast: Reliable multicasting with an overlay network. In *Proc. of the Fourth USENIX Symposium on Operating Systems Design and Implementation*, October 2000.
62. D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. *Proceedings of the 29th Annual ACM Symposium on Theory of Computing(STOC)*, pages 654–663, May 1997.
63. J. Liebeherr and M. Nahas. Application-layer multicast with delaunay triangulations. In *Global Internet Symposium, IEEE Globecom 2001 Conference*, 2001.
64. H. Liu and H. A. Jacobsen. Modeling uncertainties in publish/subscribe. In *Proc. of Conf. on Data Engineering*, 2004.
65. Y. Liu and B. Plale. Survey of publish subscribe event systems. Technical report, Indiana University, 2003.
66. C. Ma and J. Bacon. Cobea: A corba-based event architecture. In *Proc. of the 4th USENIX Conf. on O-O Tech. and Systems*, pages 117–131, Apr. 1998.
67. G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed Hashing in a Small Wold. In *Proc. of the 4th USENIX Symposium on Internet Technologies and Systems (USITS'03)*, 2003.
68. L. Mathy, R. Canonico, and D. Hutchison. An overlay tree building control protocol. In *3rd International Workshop Networked Group Communications*, 2001.
69. P. Maymounkov and D. Mazires. Kademlia: A Peer-to-peer Information Systems Based on the XOR Metric. In *Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, 2002.
70. G. Muhl. Generic Constraints for Content-Based Publish/Subscribe. In *Proceedings of the 6th International Conference on Cooperative Information Systems (CoopIS)*, 2001.
71. A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen. Ivy: A read/write peer-to-peer file system. In *Proc. of the Fifth USENIX Symposium on Operating Systems Design and Implementation*, December 2002.
72. B. Oki, M. Pfluegel, A. Siegel, and D. Skeen. The information bus - an architecture for extensive distributed systems. In *Proceedings of the ACM Symposium on Operating Systems Principles*, December 1993.
73. L. Opyrchal, M. Astley, R. E. Strom J. Auerbach, G. Banavar, and D. C. Sturman. Exploiting ip multicast in content-based publish- subscribe systems. In *Proc. of Middleware*, 2000.
74. D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An application level multicast infrastructure. In *Proc. of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS'01)*, March 2001.
75. G. Perng, C. Wang, and M. K. Reiter. Providing content-based services in a peer-to-peer environment. In *Proceedings of the third international workshop on distributed event-based systems (DEBS)*, pages 74–79, May 2004.

76. M. Petrovic, I. Burcea, and H. A. Jacobsen. S-ToPSS: Semantic Toronto publish/subscribe system. In *Proc. of Conf. on Very Large Data Bases*, pages 1101–1104, 2003.
77. P. R. Pietzuch and J. Bacon. Peer-to-peer overlay broker networks in an event-based middleware. In *Proc. of Workshop on DEBS*, 2003.
78. P. R. Pietzuch and J. M. Bacon. Hermes: A Distributed Event-Based Middleware Architecture. In *Proceedings of 1st International Workshop on Distributed Event-Based Systems (DEBS)*, pages 611–618, July 2002.
79. C. Plaxton, R. Rajaraman, and A. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. of ACM SPAA*, June 1997.
80. R. Preotiuc-Pietro, J. Pereira, F. Llirbat, F. Fabret, K. Ross, and D. Shasha. Publish/subscribe on the web at extreme speed. In *Proc. of ACM SIGMOD Conf. on Management of Data*, 2000.
81. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of ACM SIGCOMM'01*, pages 329–350, 2001.
82. S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *In Proceedings of NGC*, 2001.
83. A. Riabov, Z. Liu, J. Wolf, P. Yu, and L. Zhang. Clustering Algorithms for content-based publication-subscription systems. In *Proc. of ICDCS*, 2002.
84. V. Roca and A. El-Sayed. A host-based multicast(hbm) solution for group communications. In *1st IEEE International Conference on Networking(ICN01)*, July 2001.
85. A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms(Middleware)*, 2001.
86. A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale persistent peer-to-peer storage utility. In *Proc. of the 18th ACM Symp. on Operating Systems Principles (SOSP-18)*, October 2001.
87. A. Rowstron, P. Druschel, and M. Castro. Scribe: The design of a large-scale event notification infrastructure. In *Proc. of the 3th Int. Workshop on Networked Group Communications*, 2001.
88. B. Segall and D. Arnold. Elvin has left the building: a publish/subscribe notification service with quenching. In *Proceedings of AUUG*, pages 243–255, sep. 1997.
89. B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps. Content Based Routing with Elvin4. In *Proceedings of AUUG2K*, June 2000.
90. R. Shah, R. Jain, and F. Anjum. Efficient Dissemination of Personalized Information Using Content-Based Multicast. In *Proceedings of IEEE Infocom*, 2002.
91. H. Shen, G. Chen, and C. Xu. Cycloid: A scalable constant-degree p2p overlay network. *Journal of Performance Evaluation's Special Issue on Peer-to-Peer Networks*, (3):195–216, 2006.
92. A. Slominski, Y. Simmhan, A. L. Rossi, M. Farrellee, and D. Gannon. Xevents/xmessages: Application events and messaging framework for grid. Technical report, Indiana University, 2001.
93. C. Snoeren, K. Conley, and D. K. Gifford. Mesh based content routing using XML. In *Proc. of SOSP*, 2001.
94. I. Stoica, R. Morris, D. Liben-Nowell, M. F. Kaashoek, D. Karger, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Trans. on Networking*, August 2002.
95. R. Strom, G. Banavar, T. Ch, M. Kaplan, K. Miller, B. Mukherjee, D. Sturman, and M. Ward. Gryphon: An information flow based approach to message brokering. In *Proc. of the International Symposium on Software Reliability Engineering*, 1998.
96. D. Tam, R. Azimi, and H.-A. Jacobsen. Building content-based publish/subscribe systems with distributed hash tables. In *Proceedings of the international workshop on databases, information systems and peer-to-peer computing*, September 2003.
97. W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. P. Buchmann. A peer-to-peer approach to content-based publish/subscribe. In *Proc. of Workshop on DEBS*, 2005.

98. Tibco software inc. tibco rendezvous faq, 2003. http://www.tibco.com/solutions/products/active enterprise/rv/faq.jsp.
99. D. Tran, K. Hua, and T. Do. Zigzag: An efficient peer-to-peer scheme for media streaming. In *Proc. of IEEE Conference on Computer Communications (INFOCOM'03)*, 2003.
100. P. Triantafillou and I. Aekaterinidis. Content-based publish-subscribe over structured P2P networks. In *Proceedings of the third international workshop on distributed event-based systems (DEBS)*, pages 104–109, May 2004.
101. P. Triantafillou and A. Economides. Subscription summaries for scalability and efficiency in publish/subscribe. In *Proc. of Workshop on Distributed Event-Based Systems*, pages 619–624, 2002.
102. P. Triantafillou and A. Economides. Subscription summarization: a new paradigm for efficient publish/subscribe systems. In *Proceedings of the 24th IEEE ICDCS*, pages 562–571, 2004.
103. Y. Wang, L. Qiu, D. Achlioptas, G. Das, P. Larson, and H. J. Wang. Subscription partitioning and routing in content-based publish/subscribe networks. In *Proceedings 16th International Symposium on DIStributed Computing (DISC)*, October 2002.
104. T. Wong, R. Katz, and S. McCanne. An evaluation of preference clustering in largescale multicast applications. In *Proc. of IEEE INFOCOM*, March 2000.
105. X. Yang and Y. Zhu. A peer-to-peer approach to content-based publish/subscribe. In *Proceedings of the 2nd international workshop on Distributed event-based systems table of contents*, pages 1–8, 2003.
106. X. Yang and Y. Zhu. A DHT-based Infrastructure for Content-based Publish/Subscribe Services. In *Proceedings of P2P*, 2007.
107. X. Yang, Y. Zhu, and Y. Hu. A large-scale and decentralized infrastructure for content-based publish/subscribe services. In *Proceedings of the 36th International Conference on Parallel Processing (ICPP)*, 2007.
108. X. Yang, Y. Zhu, and Y. Hu. Scalable content-based publish/subscribe services over structured peer-to-peer networks. In *Proceedings of the 15th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, 2007.
109. B. Zhang, S. Jamin, and L. Zhang. Host multicast: A framework for delivering multicast to end users. In *Proc. of IEEE Conference on Computer Communications (INFOCOM'02)*, 2002.
110. C. Zhang, A. Krishnamurthy, and O. Y. Wang. Brushwood: Distributed trees in peer-to-peer systems. In *Proceedings of the 4th International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 47–57, 2005.
111. C. Zhang, A. Krishnamurthy, O. Y. Wang, and J. P. Singh. Combining flexibility and scalability in a peer-to-peer publish/subscribe system. In *Proc. of Middleware*, 2005.
112. R. Zhang and Y. C. Hu. HYPER: a hybrid approach to efficient content-based publish/-subscribe. In *Proceedings of international conference on distributed computing systems (ICDCS)*, June 2005.
113. 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 Division, UC Berkeley, April 2001.
114. Y. Zhu and Y. Hu. Ferry: an P2P-based architecture for content-based publish/subscribe services. *IEEE Trans Parallel Distrib Syst*, 18(5):672–685, 2007.
115. Y. Zhu and H. Shen. An efficient and scalable framework for content-based publish/subscribe systems. *Peer-to-Peer Networking and Applications*, 1(1):3–17, March 2008.
116. S. Zhuang, B. Zhao, A. Joseph, R. Kotz, and J. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proc. of the Eleventh Intl. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, 2001.