Reliable Publish/Subscribe Middleware for Time-sensitive Internet-scale Applications

Christian Esposito, Domenico Cotroneo
Dept of Computer and Systems Engineering,
Università di Napoli - "Federico II"
Napoli, 80125 - Italy
{christian.esposito, cotroneo}@unina.it

Aniruddha Gokhale
Institute for Software Integrated Systems
Dept of EECS - Vanderbilt University
Nashville, TN 37235 - USA
a.gokhale@vanderbilt.edu

ABSTRACT

Federating mission critical systems over wide-area networks still represents a challenging issue. For example, it is hard to assure both reliability and timeliness in a hostile environment such as Internet. The publish/subscribe (pub/sub) interaction model is a promising solution for scalable data dissemination over wide-area networks. Nevertheless, currently available pub/sub systems lack efficient support to achieve both reliability and timeliness in unreliable scenarios. This paper describes an innovative approach to fill this gap making three contributions. First, a cluster-based peer-to-peer organization is introduced to handle a large number of publishers and subscribers. Second, the cluster coordinator is replicated to mask process crashes and to preserve cluster connectivity toward the outside world. Third, multiple-tree redundancy is applied to tolerate link crashes thereby minimizing unpredictability in the delivery time. We present a simulation-based evaluation to assess the effectiveness of our approach in an unreliable setting. This study indicates that our approach enforces the reliability of event delivery without affecting its timeliness.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Reliability, availability, and serviceability

Keywords

Fault-tolerance, Timeliness, Peer-to-Peer Overlay, Replication, Multiple-Tree Redundancy

1. INTRODUCTION

In the last few years, more and more industrial projects aim to develop the so-called *Large scale Complex Critical Infrastructures* (LCCIs) [1], *i.e.*, Internet-scale federation of several autonomous and heterogeneous systems that work collaboratively and synergistically to provide critical facilities. This represents a novel prospective on how mission critical systems are architected: a shift from small, monolithic and vertical architectures, which characterized traditional systems, toward large highly modular and integrated systems. As a practical example, we can consider the road map outlined by EuroControl for the European Air Traffic Control (ATC) evolution, object of the European Research Project called *Single European Sky* (SESAR) [1] . The current European ATC framework is segmented among several systems, namely Area Control Centers, each one responsible for a well-defined portion of the air space. In order to handle more efficiently the growing aviation traffic, the proposed solution is a framework where the ATC operations are seamlessly and fully integrated. In fact, the novel ATC framework will be based on a data-centric model in which all Area Control Centers cooperate via a data distribution service.

The effectiveness and performance of LCCIs strongly depend on the quality of the adopted interconnection middleware, which has to deal with some serious challenges. First, critical operations exhibit a time-sensitive behavior: information is useful only if delivered "on time", *i.e.*, respecting certain time boundaries. So, *messages have to be exchanged with a predictable latency* (**Timeliness**). Second, one of the key properties that a critical system has to provide is to be dependable and able to properly handle error conditions imposed by network and process faults. Then, *message dissemination has to be guaranteed despite manifestations of several faults* (**Reliability**). Last, the rising of the activity of a single system and the escalation of connected systems increase the number of data exchanges, and consequently the time of computation. Therefore, the adopted *middleware must be able to scale while maintaining suitable performance* (**Scalability**).

Middleware solutions that adopt a publish/subscribe (pub/sub) interaction model, *i.e.*, pub/sub services, are very appealing to efficiently interconnect several systems since the inherent decoupling properties promote scalability [2]. The focus of the pub/sub community has seldom been on reliable event dissemination for two main reasons. On one hand, guaranteeing message delivery despite network failures has been always thought of as being inherited by the pub/sub system from the protocol used to implement the notification service. On the other hand, there were more challenging issues to be addressed first, such as scalability and expressiveness. However, the research interest of the community is recently shifting toward novel approaches to satisfy reliability requirements including new techniques developed specifically for these middleware to cope with several kind of faults [3, 4, 5]. However, these efforts lack an adequate support to assure reliability along with timeliness.

[1]This is one of the key objectives of the research project, namely IniziativaSoftware (www.iniziativasoftware.it), supported by the University of Naples and by FINMECCANICA, one of the largest Italian company which develops mission critical and complex system infrastructures in the context of military and civil scenarios.

DEBS '09 July 6-9, 2009, Nashville, TN, USA

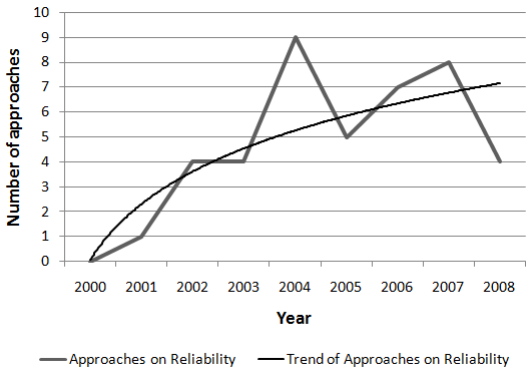

Figure 1: Efforts on reliability issues in publish/subscribe middleware

In order to fill this gap, this paper presents a novel approach based on a peer-to-peer infrastructure in the context of a pub/sub system. Our solution comprises three novel ideas as follows:

- Clustering publishers and subscribers that reside on the same routing domain, and using a hierarchical peer-to-peer organization to handle large numbers of participants without affecting the message latency;
- Implementing a replication scheme for the coordinator in each cluster to treat process crashes without leading to disconnections among the clusters or causing considerable fluctuations in the message delivery time;
- Adopting a multi-tree approach to cope with link crashes and to preserve connectivity within the overlay of coordinators without worsening the timeliness of the data dissemination.

The reminder of the paper is organized as follows. Section 2 provides 1) a definition of the properties that a reliable pub/sub service has to provide in order to be used to federate time-sensitive applications, 2) discussion on the faults that can occur in a pub/sub middleware, and 3) a taxonomy of the available solutions to implement a reliable pub/sub system. In Section 3, we describe in details our approach, while in Section 4 we report experimental results of our simulation-based study to assess the quality of our approach. Section 5 discusses related work, and we conclude in Section 6 with some remarks on future work.

2. BACKGROUND AND OPEN ISSUES

Since its inception, the publish/subscribe community has been more focused on scalable architectures, efficient delivery, and expressive subscriptions rather than reliable event dissemination. As a proof of this lack of attention on fault-tolerance issues, standardized and mature commercial pub/sub services do not address these issues at all, such as in the *Java Message Service* (JMS), or provide very basic mechanisms, such as in the recent OMG standard called *Data Distribution Service* (DDS) [6]. However, this status quo is changing as more and more pub/sub services have started to be used in application domains that expose stringent reliability requirements, *e.g.*, EuroControl decided that the technology to be used in the project SESAR, which aims to device the novel European ATC framework by interconnecting critical systems, is the DDS specification.

We have performed a survey of the available literature on reliable pub/sub services in order to measure the efforts spent on these issues. We have collected all the papers published at international conferences and journals, counting [2] how many approaches focus on reliable pub/sub services in each year starting from 2000. As shown by the black line in Figure 1, our study of the literature demonstrates that in the last decade we have witnessed a growing interest of the community in studying novel approaches to guarantee a reliable event dissemination upon networks and nodes that expose a faulty behavior.

This section presents a study of reliability aspects in pub/sub services by means of 1) defining the fundamental properties that a reliable pub/sub service has to provide; 2) studying which failures are of interest to be handled in order to guarantee a reliable event dissemination; and 3) analyzing the current approaches to devise a reliable pub/sub service to find the most suitable one to provide both reliability and timeliness.

2.1 Fundamental Properties of Reliable and Timely Publish/Subscribe Middleware

A pub/sub service is made of several processes that exchange messages through a so-called notification service. These processes play the roles of *publishers*, which produce messages, and/or *subscribers*, which consume the messages in whom they are interested. In fact, a process p_j receives only those messages that satisfy the k-th, namely $C_{j,k}$, of its active subscription predicates, contained in the set called Σ_j: if the message m_i matches the subscription $C_{j,k}$, then $C_{j,k}(m_i) \equiv \top$ and m_i is delivered to p_j, otherwise $C_{j,k}(m_i) \equiv \bot$ and m_i is not delivered to p_j. Given a message m_i, it is possible to define as its view, namely $viewof(m_i)$, all the processes that have to receive the message m_i:

$$viewof(m_i) = \begin{cases} V & \exists j,k : C_{j,k}(m_i) = \top \\ \bot & otherwise \end{cases}$$

where $V = \{p_j | \exists C_{j,k} \in \Sigma_j : C_{j,k}(m_i) = \top\}$.

Thus, a process p_j performs a series of the following operations:

1. *Publish*: At time t, a message m_i is published if p_j sends it to the notification service, namely NS, and the view of the message is not empty:
$$t = pub(m_i, p_j) \Rightarrow t = send(m_i, NS),$$
under the constraint that $viewof(m_i) \neq \bot$;
2. *Notify*: At time t, p_j is notified of a published message when a message is received by the notification service, namely NS, and one of the subscriptions contained in Σ_j is verified:
$$t = notify(m_i, p_j) \Rightarrow (t = recv(m_i, NS) \wedge C_{j,k}(m_i) = \top);$$
3. *Subscribe*: A new subscription $C_{j,k}$ is created:
$$sub(C_{j,k}, \Sigma_j) \Rightarrow \Sigma_j = \Sigma_j + C_{j,k};$$
4. *Unsubscribe*: An existent subscription $C_{j,k}$ is erased:
$$unsub(C_{j,k}, \Sigma_j) \Rightarrow \Sigma_j = \Sigma_j - C_{j,k}.$$

A pub/sub service defined in terms of the previous four operations has to satisfy two main properties:

- ☞ *Safety*: a process p_j is notified of a message m_i because another process p_i has previously published it:
$$\exists p_j : t_n = notify(m_i, p_j) \Rightarrow$$
$$\exists p_i : (t_p = pub(m_i, p_i) \wedge t_p < t_n);$$
- ☞ *Liveness*: if a process p_j has a subscription $C_{j,k}$, it receives one of the published messages that satisfy it:
$$\exists p_j, \exists C_{j,k} \in \Sigma_j \Rightarrow \exists m_i : notify(m_i, p_j).$$

A pub/sub service is defined *reliable* if the message delivery is guaranteed despite that processes may fail and/or the network may be affected by several anomalies. Hence, considering a view of the

[2] An approach may have been published in several papers in different years. *E.g.*, the Self-Stabilizing approach [4] has been published in 3 papers in 2005, 3 papers in 2006, 1 in the 2007 and 1 in 2008, we have counted it as a single paper in in the median year of the distribution, *i.e.*, 2006.

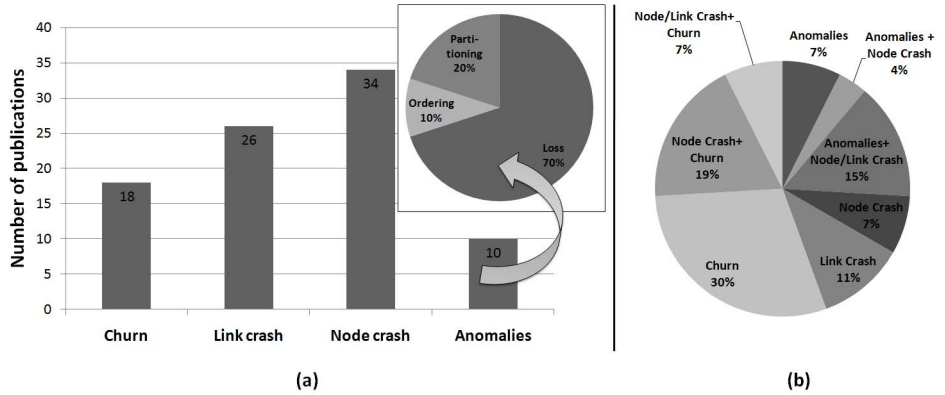

Figure 2: (a) Distribution of current solutions for each fault; (b) Taxonomy of faults addressed by current solutions

message m_i, namely V, the following properties have to be guaranteed by a reliable pub/sub service:

- ⋆ *Agreement*: if a non-faulty process p_j is notified, then all the other non-faulty processes are eventually notified:
$$\exists p_j \in V : notify(m_i, p_j) \Rightarrow \forall p \in V : notify(m_i, p);$$
- ⋆ *Validity*: if a non-faulty process p_j publishes the message m_i, then at least one of all non-faulty processes is notified:
$$\exists p_j \in V : pub(m_i, p_j) \Rightarrow \exists! p_k \in V : notify(m_i, p_k);$$
- ⋆ *Integrity*: every non-faulty process p_j performs the notify of the message m_i at most once:
$$\exists p \in V : t_{n1} notify(m_i, p) \Rightarrow \nexists t_{n2} = notify(m_i, p) : n1 \neq n2.$$

Moreover, since critical systems, such as LCCIs, exhibit real-time constraints jointly to fault-tolerance, *i.e.*, a message delivered out of temporal deadlines is considered as lost. Therefore, a pub/sub service to be used in such systems has to verifies an additional property:

- ⋆ *Timeliness*: given a deadline Δ, all non-faulty processes are notified of a published message before Δ is expired:
$$\exists \Delta, \exists p_j \in V : pub(m_i, p_j) \Rightarrow \nexists p_k \in V : not(m_i, p_k) > \Delta.$$

2.2 Taxonomy of Faults in Publish/Subscribe Middleware

Several kinds of faults may affect a pub/sub service, and they need to be carefully treated in order to achieve reliability. Briefly, such faults can be classified as follows:

- **Network anomalies**: temporary misbehaviors of a link:
  - *Loss*: links behave as a fairy-loss channel, *i.e.*, messages in transit though the link may be lost;
  - *Ordering*: messages are not received in the same order as they have been published;
  - *Corruption*: messages are received corrupted;
  - *Delay*: a message is delivered later than expected;
  - *Congestion*: a link/router is overloaded, suddenly causing several anomalies;
  - *Partitioning*: network may get fragmented into several disconnected parts.
- **Link crash**: links experience loss of connectivity, *i.e.*, packets are always lost for a certain time, which is not necessarily permanent but may dynamically appear and disappear;
- **Node crash**: nodes crash due to hardware/software failures;
- **Churn**: nodes unexpectedly join/leave the system.

With respect to this taxonomy, we have analyzed which faults are handled by the reliable pub/sub services that implement the approaches surveyed at the beginning of this section. Only 10% of these systems, as shown in Figure 2(a), handle faults due to network anomalies, and the majority of such systems, 70% of them, focus on recovering from message losses. The reason behind this may be found on the consideration that pub/sub services are designed on top of an *event dissemination protocol*. Studying how to cope with network anomalies has hitherto not been the focus of the community since it was deemed to be easily addressed using a reliable event dissemination protocol, such as one of the reliable multicast protocols developed during the last twenty years [7, 8].

On the other hand, many efforts have been spent to address faults that can lead to inconsistent topologies within the broker overlay. The most studied fault is the node crash, 34% of the systems in particular, focus on the case in which the crashed node hosted a process that acted as a broker, which is critical in the system since it glues together publishers and subscribers. While, only a smaller number of systems, 18%, consider also churn. This may be due to two reasons: 1) often these systems are built on peer-to-peer infrastructures, which gracefully manage these dynamics [3] ; 2) this fault is treated for free, without a particular solution. A close scrutiny of Figure 2(b) reveals that systems dealing with churn also handle node crashes. In fact, churn can be seen as a special case of node crash, so all the techniques adopted to cope with node crashes can also be used in this other case. On the other hand, the second most studied class of faults is the link crash, specifically in 26% of the systems. In conclusion, as shown in Figure 2(b), most of the reliable pub/sub systems deal jointly with node and link crashes, 64% in total. So, putting everything together, a failure model suitable for pub/sub services comprises only node and link crashes.

2.3 Classification of Approaches to Reliable Publish/Subscribe Middleware

This section aims to provide a classification of the current approaches adopted in literature to implement reliable pub/sub services. There are two ways to implement the notification service: one uses IP Multicast [6], while the other one adopts an application-level multicast for scalability reasons [9]. The first two approaches that we have included in this classification are employed in the first

[3] In fact, the ones that do not care about churn adopt a static broker overlay rather than a peer-to-peer infrastructure.

perspective, while the other approaches in the latter one.

Retransmissions (ARQ). A way to achieve reliable event dissemination is to have publishers store messages so that subscribers can request retransmissions when losses are somehow detected [6]. Despite the ease of implementation, this approach is not optimal due to several drawbacks. First, it can deal with only message losses. Second, the time to recover dropped messages is unpredictable. In fact, the number of retransmissions needed to successfully receive a message depends on the network behavior, which is never known *a priori*. Moreover, there are no guarantees to achieve agreement, due to a link crash that disconnects partially/completely publishers from a part of the interested subscribers or the possibility that a publisher may crash before all the subscribers have recovered lost messages. Lastly, messages can be received twice due to false-negative detection of dropped messages.

Forward Error Correction (FEC). A different way to recover lost messages is to use spatial redundancy rather than temporal redundancy: instead of using retransmissions, the sender forwards redundant data. This way the receiver can reconstruct the original message even if some packets have been dropped during the delivery [10]. This solution enforces timeliness since the worst case latency is predictable, however, it presents several drawbacks. The main one is that agreement is reached depending on the redundancy used at the sender side. In fact, a receiver can obtain the message only if the sender has delivered more additional data than the packets lost by the network. Otherwise the dropped packets are more than the ones the receiver can reconstruct and the message is considered lost. Choosing the right redundancy is a very hard task in Internet since losses are highly variable over time.

Epidemic Algorithm (Ep-ARQ). ARQ suffers from a serious scalability limitation due to the centralization of the recovery duty at the publisher side. To overcome it, a different approach is to use an epidemic approach [3] that distributes the recovery responsibility among the leaf nodes of the forwarding tree. In fact, each process exchanges at a random time its history of the received messages with a randomly-chosen process among the ones constituting the system. Subsequently, inconsistencies, *i.e.*, message drops, are detected by comparison and corrected through retransmissions. It provides more guarantees than using ARQ since it can also recover messages lost due to crashed nodes and/or links. Moreover, since the detection of lost messages is performed by comparison of histories rather than timeout expiration, it avoids having to receive the same message twice. However, there is no guarantee that the published message reaches all the subscribers, but agreement is obtained within a certain probability due to the random nature of the algorithm. Moreover, there is no predictable upper bound on the time to reach the agreement since retransmissions are adopted.

Reconfigurations (Reconf). Adopting topological reconfiguration is another possible solution to recover connectivity in the forwarding tree after a node/link has crashed. An example is the self-stabilization publish/subscribe [4], *i.e.*, routing entries within the brokers have a limited "time to live", and they have to be periodically renewed otherwise they are deleted. This approach aims to guarantee a consistent connectivity for the system, *i.e.*, all the process are connected to each other. But, it does not cope with message drops, so not all the subscribers will receive the messages of interest if network omissions may occur.

Broker Replication (BroRep). Another method to achieve fault-tolerance is applying redundancy into the system. This is realized by replicating the brokers in the forwarding tree [5]. The state of a broker is replicated to its neighbors so in case of a broker failure it can be easily substituted without losing subscription consistency into the system. This solution is tailored only for node failures, and link crashes may involve that some subscribers are not reachable and so there may be no agreement in the system.

Table 1: Properties guaranteed by the current approaches

System	Agreement	Validity	Integrity	Timeliness
ARQ [6]		✔		
FEC [10]	✔*	✔	✔	✔
Ep-ARQ [3]	✔**	✔	✔	
Reconf [4]		✔	✔	
BroRep [5]		✔	✔	
PatRed [11, 12]	✔***	✔	✔	✔

* only with the appropriate redundancy
** only within a known probability
*** only if path diversity is guaranteed

Path Redundancy (PatRed). Systems, such as Bayeux [11] and XNET [12], use a different form of redundancy, *i.e.*, establishing redundant paths among the nodes of the system. A message is sent through multiple paths, and only the first-received replica of the message is delivered to the application. This solution appears to be tailored to link failures but can also cope with node crashes thereby circumventing the failed node. This allows all the properties of the reliable pub/sub service plus timeliness under one condition: all the paths to a destination exhibit path diversity, *i.e.*, if a path fails, the others are not affected by the same failure [13].

As shown in table 1, path redundancy is an appealing solution to architect reliable publish/subscribe middleware with timeliness constraints, however, providing path diversity is still a challenging issue. In fact, different overlay links may be built on top of the same network links, so choosing different overlay links does not guarantee diversity. To mitigate this issue, the adopted broker overlay needs to be topology-aware of the underlying network, but current solutions do not provide this feature. This paper aims to define a multi-tree overlay that is aware of the underlay topology.

3. DESIGN OF RELIABLE AND TIMELY EVENT DISSEMINATION

In this section we describe the design of our novel approach for a topic-based publish/subscribe middleware to federate mission critical systems (*e.g.*, the so-called *Large scale Complex Critical System* (LCCI) [1]) over wide-area networks. Due to the requirements that LCCIs impose on the adopted middleware, this novel approach has been designed keeping the following objective in mind: all the subscribers are guaranteed to receive on-time all the messages despite several link and node crashes that may occur (*reliable and timely event delivery*).

Considering that our application scenario is related to LCCIs, we envision that a pub/sub service is composed of tens or hundreds of processes hosted on nodes scattered over Internet. Moreover, these processes do not expose churn: all the nodes provide a long-running service and will abruptly leave the system only due to a failure. Drawing from the conclusions of Section 2.2, we also assume that the pub/sub service can be affected only by node and link crashes among the plethora of possible faults. On one hand, processes may fail due to crashes, which cause nodes to halt and to lose their internal volatile state. Without loss of generality, we assume that processes do not recover after a crash (*fail-stop failure model*). On the other hand, links can crash, however, they recover

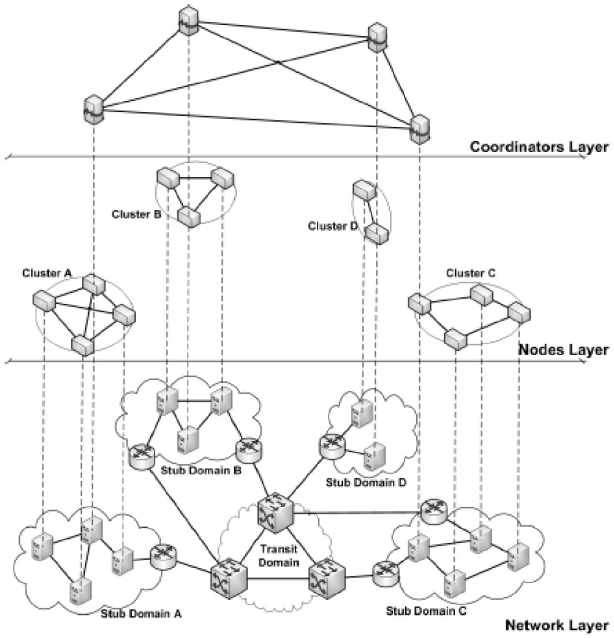

Figure 3: Different layers of abstraction in a pub/sub service

after a certain period of time has elapsed since the crash happened (*fault-recovery failure model*). Thus the same link may experience several crashes during the operational phase of the pub/sub service. Moreover, we assume that the network is not partitionable due to link crashes: given two correct processes p and q, there will always be a correct path, *i.e.*, constituted by the operational links that allow p to reach q. Even if such a situation is likely to happen in real use cases that adopt Internet as the interconnection infrastructure, we decided to not address it in this paper [4] and leave it for future work.

3.1 Grouping Nodes in Clusters

The topology of the current Internet is composed of interconnected *Routing Domains*, each one sharing common administration control and routing protocol [14]. Domains exhibit a hierarchical topological organization according to two abstraction levels, as illustrated in the lower part of Figure 3. On one hand, there are the so-called *Stub Domains* within which the path joining two of its nodes resides. These domains may consist of Local Area Networks (LANs) or Autonomous Systems (AS), and are managed by a central organization. So policies to assure quality-of-service (QoS) constraints in the data dissemination may be applied.

On the other hand, *Transit Domains* are in charge of efficiently interconnecting several stub domains and to form the network backbone. Due to a lack of a central management reference and traffic orchestrator, transit domains are affected by several failures that may compromise the effectiveness and resiliency of the message forwarding. Although important technical progress [15] has been made to address this, more work needs to be done to achieve trustworthy QoS guarantees in Internet.

A topic-based pub/sub service at the Internet scale exhibits processes, which 1) are scattered across different stub domains, 2) communicate each other through several transit domains and 3) behave according to the model introduced in subsection 2.1. As shown in the upper part of Figure 3, we propose a hierarchical approach to organize a pub/sub service, which reflects the previous considerations on the Internet topology: 1) for simplicity a node runs only a single process of the pub/sub service [5] , 2) nodes in the same domain are clustered together, and 3) each cluster holds a *coordinator* that allows interactions with the other clusters. Nodes in the same cluster communicate using *intra-cluster routing*, and can send messages outside the cluster only through their coordinator. In fact, a coordinator allows communications to the outside world exchanging messages with other coordinators using an *inter-cluster routing*. Since Internet comprises multiple domains with dissimilar QoS, intra-cluster and inter-cluster routing can be designed separately.

3.2 Overlay Routing

Since clusters are built grouping nodes that reside on the same stub domain, it is suitable to use IP Multicast as a means to implement the intra-cluster routing. In fact, it is feasible that IP Multicast is provided [6] by the network infrastructures in such domains. Using IP Multicast, we can achieve efficiency both in term of judicious use of network bandwidth and scalable support to a large number of nodes. Moreover, such domains usually exhibit low, or even negligible, probability that network faults will occur. Thus it is viable to assume that timely and reliable data delivery within clusters is guaranteed.

Each cluster has to be connected to other clusters through its coordinator, and the various coordinators need to cooperate by exchanging messages. In the literature there are three possible architectural solutions mentioned to implement this cooperation:

- *Network-level multicast*, which is based on IP Multicast [7]. This has been demonstrated to be unsuitable for communications over Internet due to two main problems: 1) the lack of IP Multicast facilities widely deployed over Internet [17], and 2) issues imposed by the so-called *Reliability versus Scalability problem* [18]. In particular, network-level Multicast is unable to guarantee highly-reliable delivery to a large number of subscribers due to an ACK implosion and the centralization of the recovery duty in the multicaster, which represents a single-point-of-failure in the system.
- *Proxy-based overlay network*, which shifts multicast support from the routers to end-systems that are organized in a static overlay structure [19]. The main drawback of this architectural paradigm is the lack of self-* capabilities (*i.e.*, self-organizing, self-configuring and self-healing) [20], *e.g.*, it needs a strong human intervention for its deployment [21]. Self-* capabilities are needed when publish/subscribe middleware are adopted in systems, such as LCCIs, that impose significant challenges. First, the geographic scale of LCCI causes complex deployment and configuration of publish/subscribe applications, which are extremely error-prone and time-consuming if realized by a human operator. Second, massive data dissemination generates possible overloading conditions, which have to be resolved taking decisions at run-time and self-organization is the only feasible solution. Lastly, publish/subscribe are affected by several possible faults, and which can be addressed faster only if pub-

[4] In fact none of the approaches introduced in Section 2.3 deal with partitionable networks.

[5] Therefore, 'process' and 'node' are sometimes used as synonyms.

[6] For simplicity, we have assumed that Stub Domains offer the possibility to use IP Multicast, however, there may be cases where this is not necessarily true. In these cases, one of the several *Group Communication Toolkits* (GCT) available in literature [16] can be used instead of IP Multicast.

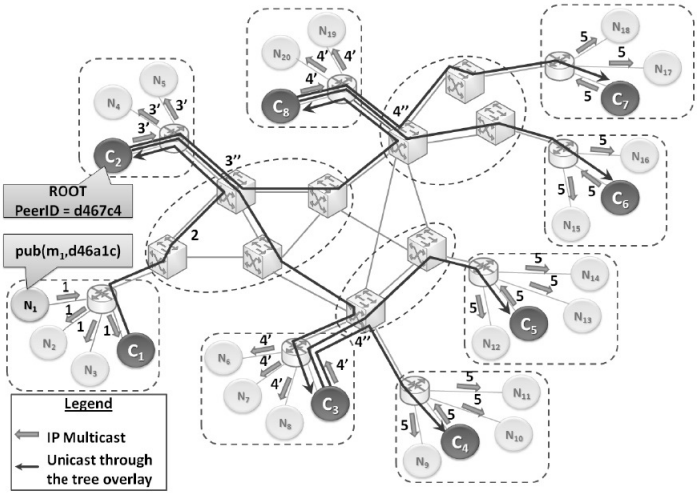

Figure 4: **Event dissemination through the coordinator tree and the clusters**

lish/subscribe applications provide self-healing capability.

- *Peer-to-Peer application-level multicast*, which implements multicast services on top of a peer-to-peer communication infrastructure. This solution provides the same scalability properties of a proxy-based overlay network, and in addiction, it enhances the system with self-* capabilities. Due to these properties, peer-to-peer application-level multicast is the most promising architecture to be used for the inter-cluster routing. However, they suffer from inefficiencies due to high load on the network. But, using proximity criteria in the peer-to-peer overlay allows reducing network load. Thus, this drawback does not limit the applicability of peer-to-peer application-level multicast in wide-area scenarios.

There is a considerable amount of literature focused on peer-to-peer application-level multicast. The ongoing debate in this field is to determine the right approach, in terms of performance and reliability, to structure the participant in a multicast session [22]. On one hand, there are *tree-based* approaches that organize the nodes into a tree, where each node can implicitly define its parent from which it receives the incoming messages. On the other hand, there are *mesh-based* approaches that expose a less structured organization letting each node to employ a swarming delivery mechanism to a certain subset of nodes.

Our choice has been to adopt a tree-based solution, and to propose mechanisms to overcome its intrinsic weaknesses. The reasons of such decision are the following: tree-based approaches provide direct control over the path followed by messages and exhibit lower communication overhead [23]. In the proposed approach, the tree-base application-level multicast is built on top of a structured or Distributed Hash Table (DHT) overlay since it simplifies the tree construction. Among the available DHT solutions, we have preferred the ones based on the *Plaxton Mesh* data structure since it enforces a fast search operation (its complexity is approximately $O(log(N))$ hops, where N is the number of peers in the overlay). Therefore, we have chosen to build our system on top of Pastry [24], however, other similar DHT overlays can be used since most of Plaxton-based DHTs do not present strong differences among themselves. Moreover, we have drawn on the experience of Scribe [25] for the tree construction. Each peer in the system is identified by a unique peerID and each multicast communication has a unique groupID. The coordinator whose peerID is numerically close to the groupID is selected as the root of the related multicast tree. A coordinator that wants to join an existint multicast tree, because one of the peers in its cluster or itself is interested to receive messages from it, sends a join message to the root. Each coordinator along the way checks if it is already in the tree: in the positive case, it registers the sender as a child and informs it to be its parent, otherwise, it forwards the message to the next coordinator on the way to the root. If a coordinator wants to send a message, because one of the peers in its cluster or itself has published a message, it forwards the message to the root, and the root would deliver it though the layers of the tree.

To sum up our approach, let us consider an example of a node publishing a message of a given topic, as shown in Figure 4:

1. Node N_1 publishes the message m_1 with groupID equal to $d46a1c$ so a multicast message is sent to all the interested nodes in the cluster and to the coordinator;
2. The coordinator C_1 of the cluster receives the message m_1 and passes it to the root of the overlay tree;
3. The root C_2 exchanges the received message with all the interested nodes in his cluster (arrows $(3')$ in the figure) and then forwards m_1 to its children (arrows $(3'')$ in the figure);
4. All the coordinators reached by the message of the root C_2 perform an IP Multicast in their cluster (arrows $(4')$ in the figure) and then forward m_1 to their children (arrows $(4'')$;
5. The message is propagated through the layers of the tree until there are coordinators, such as C_4-C_7, with no children. Subsequently, these coordinators multicast the received message into their cluster, and the delivery is concluded.

3.3 Fault-tolerance at the Cluster Level

The main weakness of our approach is that the entire system is vulnerable when a coordinator fails. In fact, the failure of a coordinator leads to both the isolation of the affected cluster from the rest of the system and the disconnection of some coordinators from the rest of the overlay tree. Moreover, since the entire traffic toward the outside world passes through the coordinator, the consequent strong workload exacerbates its time to fail.

A trivial solution to this problem is to replicate the coordinator, however, the choice of which replication flavor to use is critical since it affects the quality of the system perceived by the end-users. On one hand, one option is to use a passive replication scheme [26],

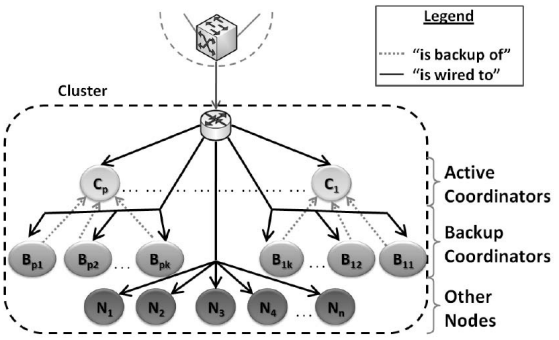

Figure 5: Replication schema inside a cluster

in which the failed coordinator is replaced by one of its backups. The case that both the coordinator and all of its replicas fail at the same time is extremely rare, so this solution improves the availability. However, timeliness is compromised since the cluster would be isolated for a certain time window, *i.e.*, the time to detect the failure of the coordinator and election of a new coordinator among the backups, so the overlay tree would be disconnected in some of its parts. On the other hand, an other option is to use an active scheme [27], in which a cluster exposes a virtual coordinator made up of a set of partners with equal responsibilities. Since there are several coordinators available at the same time, a failed coordinator is instantaneously replaced without isolating the cluster or affecting the overlay tree. So timeliness is achieved, however, availability may suffer due to the possibility of failure of all the coordinators due to common mode errors.

We propose a hybrid scheme where the coordinator is actively p-redundant, *i.e.*, there are p coordinators active at the same time, moreover, there are k backups for each active coordinator. The system designer is free to choose the robustness of the cluster varying <p,k>. Such an organization is illustrated in Figure 5 and constructed through a distributed bully election algorithm [28]. This solution has a positive impact on the overlay tree management. In fact, since the participants of the tree would result in high availability, there is no need to cope with node crashes, avoiding traditional routing approaches that circumvent the failed element and compromise the timeliness of the message delivery.

3.4 Fault-tolerance at the Overlay Level

The proposed replication-based approach does not cope with link crashes, in fact, a cluster does not receive a message when the link connecting it to its parent in the multicast tree has crashed. Considering that several prior studies have demonstrated that current Internet exhibits redundant connections at AS-level [29], link crashes can be handled by exploiting path redundancy. There are three alternative approaches to implement such a solution [30]: 1) *cross-link*, *i.e.*, connecting random peers via extra cross-cutting links; 2) *in-tree*, *i.e.*, establishing alternative links among different layers of the tree; and 3) *multiple-tree*, *i.e.*, creating several overlapping trees. While all of them have been demonstrated to improve the resiliency of the multicast service, we have chosen to adopt the latter approach, *i.e.*, the multiple-tree, since it is able to reduce delivery ratio and copes better with stringent real-time deadlines [31].

As discussed in Section 2.3, the multiple-tree approach enforces reliability and timeliness under the condition to guarantee diversity among the paths composing the multiple trees. When a message has to reach a node from another one, it travels through a path consisting of a succession of network devices, *e.g.*, routers and/or switches. Given two paths P_1 and P_2, we can define a measure of their reciprocal diversity, namely $Q(P_1, P_2)$, as the number of overlapping network devices. The two paths are deemed to be diverse if $Q(P_1, P_2)$ is zero. The overlapping devices can be identified through measurements at path- and AS- level as described in [32].

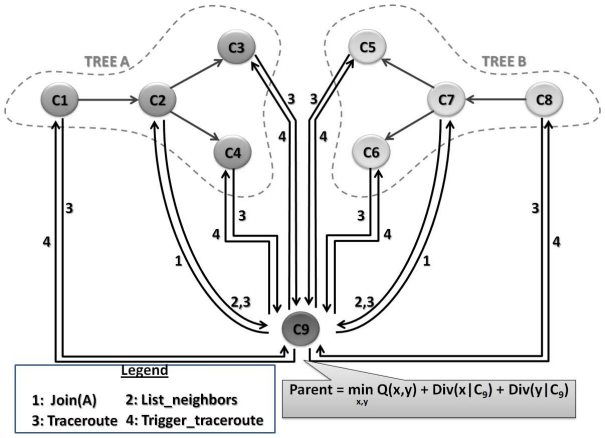

Figure 6: How a new node join two distinct trees

Given such a measure, it is possible to have two different formulations of path diversity. Given a forest of n trees, namely T_i with i = 1,...,n, such a forest verifies the path diversity constraint if and only if it is not possible to find in any of the n trees two paths that exhibit a positive value as measure of their reciprocal diversity (*Global Diversity*):

$$F = \bigcup_{i=1,\ldots,n} (T_i) : F \; is \; diverse \Leftrightarrow$$
$$\nexists P_i, P_j \in F : (Q(P_i, P_j) > 0) \wedge (i \neq j).$$

In large scale networks, it is impossible to verify if trees satisfy this condition since it requires global knowledge of all the connections among the coordinators in the systems and their reciprocal diversity. Hence, to use a multi-tree approach in a distibuted manner, we have to provide a different formulation of tree diversity: given a node N_A and n trees, namely T_i with i = 1,...,n, the forest of n trees verifies the path diversity constraint if and only if all the paths from N_A to its parents and children in the i-th tree, namely $P_{i|N_A}$ do not exhibit a positive value as measure of their reciprocal diversity (*Local Diversity*):

$$F = \bigcup_{i=1,\ldots,n} (T_i) : F \; is \; diverse \Leftrightarrow$$
$$\forall N_A \; \nexists P_{i|N_A}, P_{j|N_A} : (Q(P_{i|N_A}, P_{j|N_A}) > 0) \wedge (i \neq j).$$

where $P_{x|y}$ is a path from node y to node x. The local diversity does not imply the global diversity, however, it makes possible to implement a distributed algorithm that is able to construct locally-diverse multiple trees. Therefore, we have modified the joining procedure of Scribe to construct multiple path-disjoint trees that satisfy the local diversity constraint.

As illustrated in Figure 6, let consider that a new node C_9 wants to join two different trees, namely A and B. At the beginning, C_9 sends two join messages through Scribe and two nodes of the systems, indicated in Figure as C_2 and C_7 are contacted. C_2 and C_7 have respectively joined tree A and B. Each one of these nodes replies C_9 with a message containing 1) a list of the neighbors in the tree (*e.g.*, its parent and children) and details on their path, and 2) content of a traceroute on the path to C_9, *e.g.*, the list of the traversed network devices. Then, C_9 contacts all the nodes in each

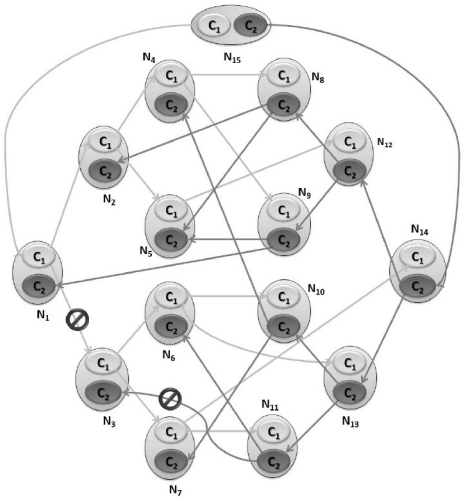

Figure 7: Event delivery among coordinators

on of the received lists, and receives traceroute messages about the path to them, too. After collecting such information, C_9 can make the decision on which will be its parent in each tree, according to these two rules:

1. the paths from C_9 to its parents have to expose the lowest measure of diversity;
2. given a parent of C_9, the paths to its children have to maintain a measure of diversity closer to the value they had before the inclusion of C_9 as a child.

So the node C_9 decides on its parents by performing the following optimization to achieve local diversity:

$$\widehat{x} = \min_{\bar{x}, y=C_9} \left[\check{Q}(\bar{x}, y) + \sum_{x_i \in \bar{x}} Div(x_i|y) \right],$$

where $\bar{x} = \{x_1, x_2, ..., x_n\}$ is a list of the possible parents for C_9, while $\widehat{x}$ is the list of the chosen parents for C_9. Moreover, the first addend of the sum to be minimized formalizes the first rule, where $\check{Q}(\bar{x}, y)$ measures the diversity of the paths from node y to the parents contained in vector $\bar{x}$, whose length is equal to n:

$$\check{Q}(\bar{x}, y) = \sum_{\substack{x_i, x_j \in \bar{x} \\ i \neq j}} Q(P(x_i, y), P(x_j, y)),$$

where the path from a node x to a node y is indicated as $P(x, y)$. While, the second addend formalizes the second rule and evaluates the variation of the diversity of the neighbors of a node x if x is promoted as child of x:

$$Div(x|y) = \check{Q}(\{V_x \cup y\}, x) - \check{Q}(V_x, x),$$

where V_x is the list of the neighbors of node x before putting y in the children list. In the optimal case, due to the locality of Pastry, we will always find nodes that exhibit a diversity measure equal to zero, however, in the real case this is not always possible. Since, the achievable diversity is always lower than the intrinsic diversity of the topology at the network level, we acknowledge that there are cases where the minimum of the previous optimization is not zero.

Since clusters host multiple active coordinators, there is a problem on how performing the joining procedure. In order to avoid the case of coordinators of a same cluster exhibiting different dependencies in the same overlay tree, only one coordinator at a time performs the joining procedure. Recalling from the previous example, at the end of the join procedure, C_9 1) sends to its parents the IP addresses of its partners and backups, 2) receives from the parents the IP addresses of their partners and backups, and 3) forwards the received IP addresses to its partners and backups. So, each active coordinator will be in charge of sending messages only through one tree. Thus the parameter p also defines the number of multiple trees that are established in the system.

Even when using a multiple-tree approach, there may be cases in which some nodes may experience message losses due to link crashes. As shown in Figure 7, since two links have crashed, node N_3 will never receive messages published by N_{15} even if it is not isolated. This is possible when all the inbound connections to the link have crashed. However, the outbound connections are still correct [7] , and they can be used to recover from this situation.

Messages exchanged though different trees do not reach a node at the same time. Let us consider node N_7 in Figure 7. It would receive a message before from N_3 and then from N_{10}. So, when N_7 receives a message from N_{10}, but nothing from N_3 before a timer has expired, it can assume that the message has not reached N_3 and notifies it that it has received a message. So, N_3 knows that it has missed a message and asks N_7 for its transmission. Since it has lost a message, N_3 assumes that all his inbound connections are incorrect and executes the joining procedure to restore its connections to the trees.

4. SIMULATION STUDY

In this section we present the results of a simulation study to assess the quality of our approach. Instead of using a real wide-area network, such as PlanetLab [8] , that exhibits uncontrollable loss patterns and makes tests non reproducible, we have conducted the study of our approach in a simulation environment called OMNET++ [9] . Our choice of this environment is motivated by its ease of use, modular architecture, parametric approach and open-source code base. Moreover, OMNET++ is rapidly becoming a popular simulation platform in the scientific community as well as in industrial settings. In fact, it has an advantage over other existing simulators since owing to its generic and flexible architecture it easily allows for the simulation of virtually any modular, event-driven system, and not just communication-network oriented systems.

OMNET++ follows a hierarchical architecture. At the lowest level there are simple modules which encapsulate behavior of a given protocol or application. These simple modules are represented by C++ classes. A compound module may be composed of simple as well as other compound modules. Modules communicate with each other via message-passing. An event is said to have occurred whenever a module sends/receives a message. OMNET++ enforces code reuse since its community provides several third-parties models that can be easily included.

An example is the INET framework, which models mostly all the protocols of wired, wireless and ad-hoc networks and several networking components, such as routers, switches and access points. These models have parameters whose values are specified externally in an initialization file, and can be varied in different simulation runs. In the context of data dissemination protocols, these parameters can be used to simulate and analyze the effect of different network conditions on the performance of event delivery. Additional information about OMNET++ can be found in its User Manual. [10]

[7] We do not treat the case in which all the connections to a node, inbound and outbound, are crashed, because in this case the node would be completely isolated. Since we have assumed that the network is not partitionable, this case can never happen.

[8] www.planet-lab.org

[9] www.omnetpp.org.

[10] http://www.omnetpp.org/doc/manual/usman.html

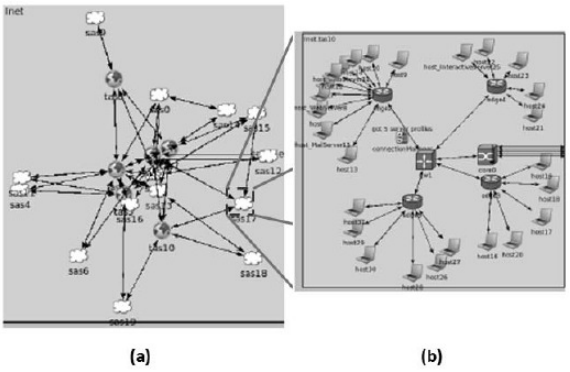

Figure 8: OMNET Model of the Internet topology used in our study: a) topology at AS-level, b) topology at the router-level, and b) implementation of a node

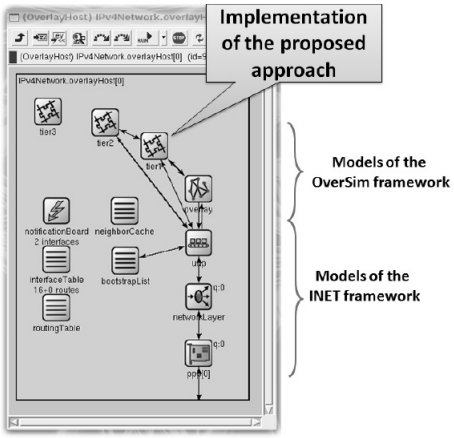

Figure 9: OMNET Model of the implementation of a node

4.1 Simulation Setup

In our tests we have used a two-level power-law topology with 300 nodes, 8 AS and 500 edges generated by a topology generator for OMNET++ called Rease [11] with default end-to-end delays. Specifically, the AS-level topology, *i.e.*, how different ASs are interconnected to each other, is based on the *Positive Feedback Preference* (PFP) model [33], shown in Figure 8(a). The router-level topology, *i.e.*, how different routers and nodes in a given AS are interconnected each other, is based on the *Heuristic Optimal Topology* (HOT) model [34], illustrated in Figure 8(b).

At each test, we have randomly selected a fixed number of nodes from the 300 IP-layer nodes as overlay nodes, among which only one at time is the publisher and all the others are subscribers. Moreover, we have 1) used the Scribe implementation provided by an overlay network simulation framework for the OMNeT++ called OverSim [12] , modifying it according to our approach, as shown in Figure 9, 2) implemented an OMNET module to perform the path measurements as described in [32], and 3) made a patch to the INET framework in order to reproduce link crashes that do not partition the network.

Node and link failures are uniformly distributed over the simulation time, and parameters have been derived from [35, 36]: while the duration of a link crash has a random distribution with a minimum duration of 10 seconds and a maximum duration of 1,200 seconds, the time between link failures is reduced to a distribution with a mean equal to 5,000 seconds. Lastly, the publishing rate is one message per second and the simulated duration of a single test is two hours.

4.2 Evaluation Results

Figure 10 illustrates the results of the tests we have performed to assess the scalability of our approach, as indicated in the figure as "Mod. Scribe", compared to the unmodified version of Scribe. As shown in Figure 10(a), clustering AS-related nodes improves the scalability of the approach, and the latency is affected only by the number of groups. However, if we vary the number of nodes and leave the same number of groups, the trend of the latency is almost constant. We have studied the timeliness quality of our approach in terms of the standard deviation of the delivery time, illustrated in Figure 10(b).

With few nodes, the original Scribe exhibits better values for timeliness, but increasing the number of nodes involves a rise in the standard deviation indicating a jitter. On the contrary, the timeliness achieved by our approach exhibits a trend with lower increasing rate. The greater standard deviation measured when using our approach than when using Scribe is caused by the use, in our approach, of alternative paths to deliver a message when a path crashes. In fact, as we previously stated, we do not experience the same delivery time of a message though different trees. Thus, the faster path may crash, hence the node receives messages over the slower path. This leads to the variation in the delivery time that motivates the worsening observed in Figure 10(b).

We have evaluated the improvement in terms of reliability, measured as the mean success rate, *i.e.*, the ratio of the received messages and the total published messages, shown in Figure 10(c). Scribe provides no assurances on reliable event dissemination, so its success rate is low and related to the number of nodes in the tree: increasing the number of nodes causes a drop in the success rate. Our approach on the other hand exhibits better reliability degree, which is not dependent on the total number of nodes. However, the achieved reliability is not equal to 1. In fact we have registered during our tests that about 5% of the published messages do not reach a subset of all the subscribers. In fact, as we previously stated, it is possible that the minimization performed when a coordinator joins the multicast tree does not return 0 as the diversity measure. The reason for this result is that the achievable diversity is lower bounded by the redundancy degree in the Internet topology, *i.e.*, greater is the path redundancy among the ASes, the closer we can get to the global diversity in the tree forest. In fact, an AS may be connected to only one transit domain, and this limits the achievable diversity, causing subscribers on these ASes to lose messages. In fact, we have made tests with topologies where we forced the ASes to have connections to more than one transit domain, and in these cases reliability is equal to 1.

Lastly, the previously-presented tests have been performed applying only link crashes, and we have seen how they influence delivery time and number of lost messages. To study the effects of process crashes, we have performed the same tests applying only process crashes, then we have compared the obtained results to the results of previous tests. In these tests with process crashes our approach presents comparable delivery latencies to the previous tests,

[11] projekte.tm.uka.de/trac/ReaSE
[12] www.oversim.org/

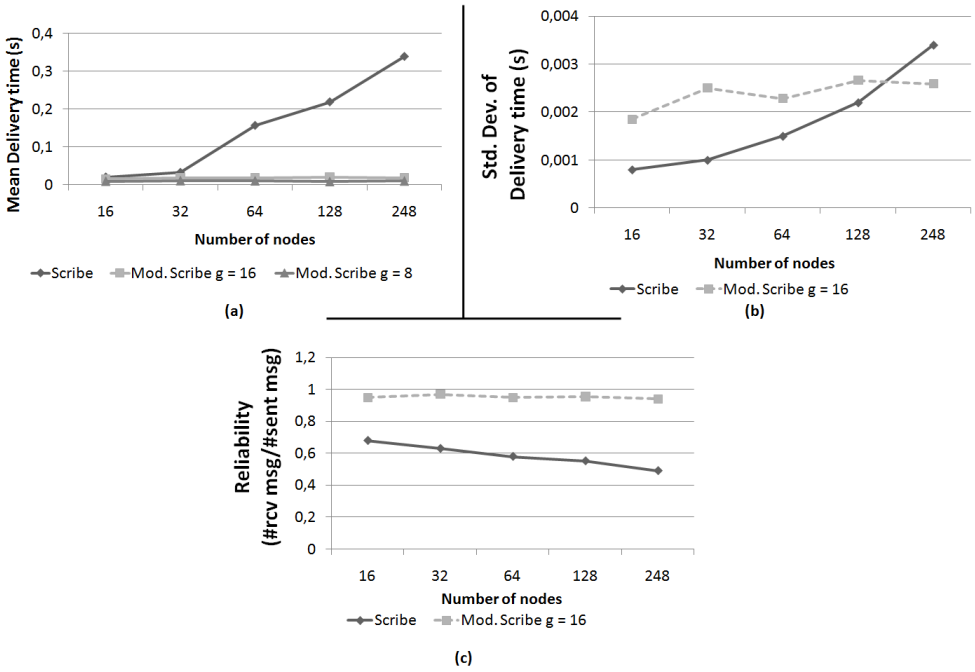

Figure 10: Results of the simulation study: a) Scalability, b) Timeliness, and c) Reliability

as shown in Figure 11(a). Each coordinator receives a copy of the message from a different path characterized by a certain latency. When a coordinator with the best path, *i.e.*, the first to receive the message, crashes, the nodes of the cluster receive the message from the coordinator of the worst quality. This is similar to the case when the path of best quality crashes and the message is received from the path of worst quality. For this reason the performance of our approach when process crashes are applied are similar to the performance when only link crashes are applied. On the other hand, even if process crashes happened, we observed that all the published messages are delivered to all the subscribers, as illustrate in Figure 11(b). This means that the replication of the coordinator allows masking any possible process crashes without leading to considerable performance drops.

5. RELATED WORK

As stated in section 2, a quite large number of research proposals have addressed the issue of reliable publish/subscribe middleware. This section aims to compare our approach to the similar ones available in literature. Moreover, we have used three well-known techniques but in an innovative manner:

- a clustered architecture to publish/subscribe middleware;
- multiple trees to guarantee message delivery;
- replication of coordinators in clustered systems or of brokers in distributed notification systems.

Since these techniques have been around for a while, there exist several approaches that have used them. Therefore, we discuss the difference between these approaches and ours in order to point out its novelty.

Our approach comes close to the one implemented in [37] since it also adopts broker replication and path redundancy. However, there are some primary differences: 1) since the connection among the brokers is ring-based in [37], it is less scalable than our approach, which is tree-based, 2) even if paths are replicated in the overlay, the topology-awareness is not addressed at all, and finally 3) system initialization needs global knowledge into the system, which is impossible to have in Internet-scale pub/sub service, while we do not need such information.

Clustering has been adopted in several publish/subscribe solutions [38] by grouping together applications that share similar subscriptions. Such techniques are basically used for two reasons. A hierarchical clustered organization allows publish/subscribe systems [39] to efficiently manage a large number of applications, as we demonstrated in 10(a). On the other hand, clustering reduces the maintenance costs when a pub/sub service has to handle a large number of topics [40]. On the contrary, we use clustering focusing more on reliability than scalability or efficiency. In fact, we group together applications that communicate by sharing the same quality of service rather than based on their subscriptions.

Multiple trees have been extensively used for resilient overlay multicast in the context of multimedia applications [30], but not in publish/subscribe systems. The main difference of our approach with such multimedia approaches consists of the broker replication. This replication completely masks process crashes at the coordinator layer so it is not possible to have disconnections in the tree and we do not have to use time-consuming reconfiguration techniques.

Also **Broker replication** was primarily adopted to improve scalability and to reduce dissemination latency [41], sharing among several brokers the load of distributing several different events. Subsequently, it has also been used to improve reliability through a redundant routing tree [42]: a logical broker is made by several physical brokers, so a logical link between two logical brokers is composed of several physical links among the physical brokers that compose the logical brokers. Such a solution is similar to ours, but the redundant paths that are chosen are not diverse in related work. Thus, even if there are several physical links, two logical brokers can become disconnected by a link crash.

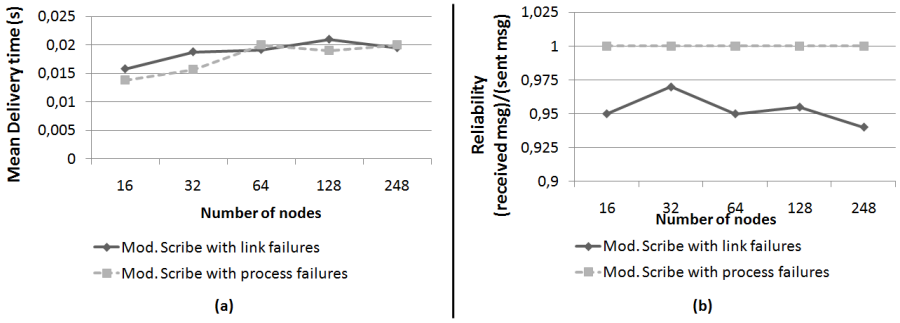

Figure 11: Comparison of performance (a) and reliability (b) varying the applied fault model

6. CONCLUSIONS AND FUTURE WORK

In this paper we have analyzed the available approaches to implement reliable pub/sub services and their effectiveness to the application scenario of time-sensitive application over Internet. Path redundancy has been shown to be an appealing approach, however, current solutions that apply this approach do not provide path diversity, which has to be satisfied to guarantee that all subscribers would receive all the messages. We have proposed a novel hybrid peer-to-peer approach that combines the replication of the coordinator in a cluster with multiple trees. We have performed a simulation study to assess the quality of the proposed solution in terms of scalability, timeliness and reliability.

We plan to make a more comprehensive simulation study to analyze in details the properties of the proposed solution under different network conditions. Moreover, multiple trees expose the drawback to generate considerable additional traffic, so we aim to apply Network Coding in order to optimize the generated traffic and to achieve efficient use of the network resources.

7. ACKNOWLEDGEMENTS

This work has been partially supported 1) by Regione Campania, in the framework of the REMOAM project (REti di sensori per il MOnitoraggio dei rischi AMbientali), misura 3.17 POR Campania 2000/06, 2) by the University of Naples and by FINMECCANICA group in the framework of the project "Iniziativa-Software", and 3) by the Consorzio Interuniversitario Nazionale per l'Informatica (CINI) and by the Italian Ministry for Education, University, and Research (MIUR) in the framework of the FIRB Project "TOCAI.IT: Tecnologie Orientate alla Conoscenza per Aggregazioni di Imprese in InterneT".

8. REFERENCES

[1] S. Bologna, C. Balducelli, G. Dipoppa, and G. Vicoli. Dependability and Survivability of Large Complex Critical Infrastructures. *Computer Safety, Reliability, and Security, Lecture Notes in Computer Science*, 2788:342–353, September 2003.

[2] P.Th. Eugster, P. A. Felber, R. Guerraoui, and A. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys (CSUR)*, 35(2):114–131, January 2003.

[3] P. Costa, M. Migliavacca, G.P. Picco, and G. Cugola. Introducing Reliability in Content-Based Publish-Subscribe through Epidemic Algorithms. *Proceedings of the 2nd International Workshop on Distributed Event-Based Systems (DEBS 03)*, pages 1–8, 2003.

[4] G. Muhl, M.A. Jaeger, K. Herrmann, T. weis, A. Ulbrich, and L. Fiege. Self-stabilizing Publish/Subscribe Systems: Algorithms and Evaluation. *Proceeding of the 11th International Euro-Par Conference*, pages 664–674, 2005.

[5] M.R. Selim, Y. Goto, and J. Cheng. A Replication Oriented Approach to Event Based Middleware over Structured Peer to Peer Networks. *Proceedings of the 5th International Workshop on Middleware for Pervasive and Ad-hoc Computing: held at the ACM/IFIP/USENIX 8th International Middleware Conference*, pages 61–66, 2007.

[6] Object Management Group. Data Distribution Service (DDS) for Real-Time Systems, v1.2. *OMG Document*, 2007.

[7] K. Obraczka. Multicast Transport Mechanisms: A Survey and Taxonomy. *IEEE Communications Magazine*, 36, January 1998.

[8] C.L. Abad, W. Yurcik, and R.H. Campbell. A Survey and Comparison of End-System Overlay Multicast Solutions Suitable for Network-Centric Warfare. *Proceedings of SPIE conferences*, 5441:215–226, 2004.

[9] A. Carzaniga, D.S. Rosenblum, and A.L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems (TOCS)*, 19(3):332–383, August 2001.

[10] J. Hoffert, D.C. Schmidt, M. Balakrishnan, and K. Birman. Supporting Large-scale Continuous Stream Datacenters via Pub/Sub. *Proceedings of the Large-Scale Distributed Systems and Middleware Workshop (LADIS 08)*, 2008.

[11] S.Q. Zhuang, B.Y. Zhao, A.D. Joseph, R.H. Katz, and J.D. Kubiatowicz. Bayeux: An Architecture for Scalable and Fault-tolerant Wide-area Data Dissemination. *Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 11–20, 2001.

[12] R.Chand. XNet: A Reliable Content-Based Publish/Subscribe System. *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems (SRDS 03)*, pages 264– 273, 2003.

[13] J. Han and F. Jahanian. Impact of Path Diversity on Multi-homed and Overlay Networks. *Proceedings of the 34th International Conference on Dependable Systems and Networks (DSN 04)*, pages 29–39, 2004.

[14] E.W. Zegura, K.L. Calvert, and S.Bhattacharjee. How to Model an Internetwork. *Proceedings of the 15th Annual Joint Conference of the IEEE Computer Societies on Networking the Next Generation (INFOCOM '96)*, 2:594–602, 1996.

[15] W. Zhao, D. Olshefski, and Henning Schulzrinne. Internet Quality of Service: An Overview. *Columbia University Research Report CUCS-003-00*, 2000.

[16] G.V. Chockler, I. Keidar, and R. Vitenberg. Group Communication Specifications: a Comprehensive Study. *ACM Computing Surveys (CSUR)*, 33(4):427–469, December 2001.

[17] C. Diot, B.N. Levine, B. Lyles, H. Kassan, and D. Balendiefen. Deployment issues for the IP Multicast services and architecture. *IEEE Network - Special Issue Multicasting*, 14:78–88, 2000.

[18] J.F. Rezende and S. Fdida. Scalability Issues on Reliable Multicast Protocol. *Proceedings of COST 237 Workshop*, 1999.

[19] L. Lao, J.H. Cui, and M. Gerla. TOMA: A Viable Solution for Large-scale Multicast Service Support. *Proceedings of the International IFIP-TC6 Networking Conference No4*, pages 906–917, 2005.

[20] E. Anceaume, A.K. Datta, M. Gradinariu, G. Simon, and A. Virgillito. A Semantic Overlay for Self-* Peer-to-Peer Publish/Subscribe. *Proceedings of the 26th IEEE International Conference on Distributed Computing Syste*, pages 22–31, 2006.

[21] P.R. Pietzuch and J. Bacon. Peer-to-Peer Overlay Broker Networks in an Event-Based Middleware. *Proceedings of the 2nd International Workshop on Distributed Event-Based Systems (DEBS 03)*, pages 1–8, 2003.

[22] J. Seibert, D. Zage, S. Fahmy, and C. Nita-Rotaru. Experimental Comparison of Peer-to-Peer Streaming Overlays: An Application Perspective. *Proceedings of the the 33rd IEEE Conference on Local Computer Networks*, pages 20–27, 2008.

[23] M. Hosseini, D.T. Ahmed, S. Shirmohammadi, and N.D. Georganas. A Survey of Application-Layer Multicast Protocols. *IEEE Communications Surveys & Tutorials*, 9(3):58–74, 2007.

[24] A. Rowstrom and P. Drushel. Pastry: Scalable, Decentralized Object Localization and Routing for Large-scale Peer-to-Peer Systems. *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001), Lecture Notes in Computer Science*, 2218:329–351, 2001.

[25] M. Castro, P. Drushel, A.M. Kermarec, and A. Rowstrom. Scribe: A Large-scale and Decentralized Application-level Multicast Infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC)*, 20(8):1489–1499, January 2004.

[26] J. Li and S. Vuong. An Efficient Clustered Architecture for P2P Networks. *Proceedings of the 18th International Conference on Advanced Information Networking and Applications (AINA 04)*, pages 278–284, 2004.

[27] B. Beverly Yang and H. Garcia-Molina. Designing a Super-Peer Network. *Proceedings of the 19th International Conference on Data Engineering (ICDE 03)*, pages 49–60, 2003.

[28] H. Garcia-Molina. Elections in a Distributed Computing System. *IEEE Transactions on Computers (TC)*, C-31(1):48–50, January 1982.

[29] R. Teixeira, K. Marzullo, S. Savage, and G.M. Voelker. In Search for Path Diversity in ISP Networks. *Proceedings of the 3rd ACM SIGCOMM Internet Measurement Conference (IMC 03)*, pages 313–318, 2003.

[30] S. Birrer and F.E. Bustamante. A Comparison of Resilient Overlay Multicast Approaches. *IEEE Journal on Selected Areas in Communications (JSAC)*, 25(9):1695–1705, December 2007.

[31] J. Seibert, D. Zage, S. Fahmy, and C. Nita-Rotaru. Experimental Comparison of Peer-to-Peer Streaming Overlays: An Application Perspective. *Proceedings of the 33rd IEEE Conference on Local Computer Networks (LCN 2008)*, pages 20–27, 2008.

[32] Z.M. Mao, J. Rexford, J. Wang, and R.H. Katz. Toward an accurate AS-level Traceroute Tool. *Proceedings of the Conference on Applications, Technologies, Architectures and Protocols for Computer Communications*, pages 365–378, 2003.

[33] H. Haddadi, D. Fay, S. Uhlig, A. Moore, R. Mortier, A. Jamakovic, and M. Rio. Tuning topology generators using spectral distributions. *Proceedings of the International Performance Evaluation Workshop (SPEC 08), In Lecture Notes in Computer Science*, 5119, 2008.

[34] L. Li, D. Alderson, W. Willinger, and J. Doyle. A First-principles Approach to Understanding the Internet's Router-level Topology. *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 3–14, 2004.

[35] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.N. Chuah, and C. Diot. Characterization of Failures in an IP Backbone. *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 04)*, 99(7):7–11, January 2004.

[36] B. Chandra, M. Dahlin, L. Gao, and Amol Nayate. End-to-end Wan Service Availability. *IEEE/ACM Transactions on Networking (TON)*, 11(2):300–313, 2003.

[37] H. Jafarpour, S. Mehrotra, and N. Venkatasubramanian. A Fast and Robust Content-based Publish/Subscribe Architecture. *Proceedings of the 7th IEEE International Symposium on Network Computing and Applications (NCA 08)*, pages 52–59, 2008.

[38] L. Querzoni. Interest clustering techniques for efficient event routing in large-scale settings. *Proceedings of the 2nd ACM International Conference on Distributed Event-Based Systems (DEBS 08)*, pages 13–22, 2008.

[39] R. Baldoni, R. Beraldi, V. Quema, L. Querzoni, and S. Tucci-Piergiovanni. Tera: Topic-based event routing for peer-to-peer architectures. *Proceedings of the Inaugural ACM International Conference on Distributed Event-Based Systems (DEBS 07)*, pages 2–13, 2007.

[40] T. Milo, T. Zur, and E. Verbin. Boosting topic-based publish-subscribe systems with dynamic clustering. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 749–760, 2007.

[41] N. Carvalho, F. Araujo, and L. Rodrigues. Scalable qos-based event routing in publish-subscribe systems. *Proceedings of the Fourth IEEE International Symposium on Network Computing and Applications (NCA 05)*, pages 101–108, 2005.

[42] Y. Zhao, D. Sturman, and S. Bhola. Subscription propagation in highly-available publish/subscribe middleware. *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware (MIDDLEWARE 04)*, pages 274–293, 2004.