Epidemic Algorithms for Reliable Content-Based Publish-Subscribe

Paolo Costa, Matteo Migliavacca, Gian Pietro Picco, Gianpaolo Cugola
Dip. di Elettronica e Informazione, Politecnico di Milano, Italy
{costa,migliava,picco,cugola}@elet.polimi.it

Publish-subscribe middleware has recently become popular because of its asynchronous, implicit, multipoint, and peer-to-peer style of communication. A number of publish-subscribe systems have been proposed to date. Here, we focus on those that seek increased scalability and flexibility by exploiting a distributed architecture for event dispatching, and use a content-based scheme to match events with subscriptions. Moreover, we assume the common routing strategy based on a subscription forwarding scheme where subscriptions delivered to every dispatcher along a single unrooted tree connecting all the dispatchers are used to establish the routes followed by events. Unfortunately, this kind of middleware usually does not provide reliability guarantees and solutions developed in other contexts are not immediately applicable. In our research contribution, we developed solutions for reliable publish-subscribe. Our results are not tied to a specific source of event loss (e.g. lossy links or topology changes) and hence enjoy general applicability.

Our approach achieves reliability by relying on *epidemic algorithms*, a breed of distributed algorithms that find inspiration in the theory of epidemics. Epidemic (or *gossip*) algorithms constitute a lightweight, scalable, and robust means of reliably disseminating information to a group of recipients, by providing guarantees only in probabilistic terms. Given their characteristics, epidemic algorithms are amenable to the unreliable and highly dynamic scenarios we target.

Epidemic algorithms differ about the mode of communication, which can exploit a push or pull style. In a *push* style, each process gossips periodically, to disseminate a digest of its cached events to other processes. Instead, in a *pull* style a process solicits the transmission of events from other processes to compensate for losses.

These algorithms typically rely on some notion of *group* (or subject) that is exploited in at least two ways. On one hand, the group defines the set of nodes (the group members) that collectively define the scope of a gossip interaction. Hence, it provides a way to determine how to route gossip messages within the system. On the other hand, the group is used to tag the messages exchanged within it, providing a clue for the recovery process when the message gets lost. Unfortunately, content-based publish-subscribe systems do not provide an *explicit* notion of group. This observation is at the core of the challenge of applying epidemic algorithms to content-based publish-subscribe systems, which can be summarized in two main issues: detecting event loss and routing gossip messages.

The first algorithm we developed uses proactive gossip push. Despite the absence of a group notion, we leverage off of the fact that every dispatcher knows, according to the subscription forwarding scheme, all the patterns in the system, and consequently can determine which patterns match a cached event e. Hence, each dispatcher is able to construct a gossip message which includes a digest of all the cached events matching a given pattern p. This gossip message can then be labelled with p and routed similarly to events matching p.

The other two algorithms adopt instead a reactive pull scheme. The technique we employ to detect event loss is to tag events with identifiers carrying enough information to enable loss detection. Besides the publisher, these identifiers contain information about the patterns matched by the event, each associated with a sequence number incremented at the source each time an event is published for that pattern. This scheme, which is a generalization of the one employed in subject-based systems, enables the detection of event loss. Whenever a dispatcher receives an event matching a pattern p, but for which the sequence number associated to p in the event identifier is greater than the one expected for that pattern and source, it can detect the loss of an event and trigger the appropriate actions. The two solutions are complementary, and differ in the way they attempt to retrieve the missing event. The first one steers gossip messages towards the event receivers (the subscribers), while the other steers them towards the publisher.

Simulation results show that our solutions indeed improve significantly event delivery, are scalable, and introduce only limited overhead.