Generic Announcement Protocol for Event Messaging

Andrea Soppera, Trevor Burbridge, Bob Briscoe, Mike Rizzo
BTexact, Adastral Park, Martlesham, Ipswich. IP5 3RE
{andrea.2.soppera,trevor.burbridge@bt.com, bob.briscoe@bt.com, mike.rizzo@bt.com}

August 28, 2003

Abstract

We describe the Generic Announcement Protocol (GAP), a two-tier generic multicast transport designed for scalable event notification. GAP requires no extra central infrastructure or administration beyond a multicast enabled network or an equivalent overlay. GAP's scalability arises from the use of announcement indexes, which exploit the underlying openness and flexibility of the raw GAP protocol. Event notifications can be massively multiplexed onto a multicast group channel, with interested receivers only joining the group for the brief duration of the announcement, coordinated by an acyclic graph of indexes, which are themselves announcements on other channels. This indexing technique, combined with the GAP protocol is particularly efficient when waiting for events to occur, since the network resources (for addressing and filtering) are kept to a minimum.

1 Introduction

In this paper we discuss the requirements for large scale event messaging and introduce a Generic Announcement Protocol that allows the scalable distribution of event messages in the publish/subscribe paradigm.

We can predict a world [1] in which distributed systems are found everywhere, continually exchanging information across their networks. This can provide huge benefits for large-scale collaborative environments, from social communities to business supply chains. The Internet has promoted the growth of distributed information processing by removing the physical boundaries that have restricted inter-enterprise communication. We foresee a future where the massive amount of information shared between different companies provides great benefits for trading partnerships and the automation of business relationships. The ability to monitor events in a scalable and efficient manner will be a key requirement in the future of distributed applications.

Events are characterized by the fact that a listener does not have any control over when an event is generated. However, a listener must be notified nearly immediately when such an event occurs. An event messaging system can be considered as a system that transmits status changes of variables or objects in the form of events. The Generic Announcement Protocol has been designed to be scalable such that applications writers can programme listeners for any number of distributed events as they would for events on the local bus.

This article is organized as follows. Section 2 reviews the requirements for large-scale global events notification systems and introduces the concept of index-based event messaging. Section 3 describes our Generic Announcement Protocol. Section 4 presents some implementation and performance results.

2 Event Messaging Systems

Event messaging systems are typically based on the 'publish/subscribe' paradigm [2]. Subscribers submit their interest on a set of events to the event messaging system, which provides the routing of events. Such a subscription can take the form of a previously agreed address or channel (on which the user expects events of interest to be delivered), or a dynamic filter that expresses what they are, or are not, interested in. Often both are used in combination. The dissemination is often provided through application layer multicast using filtering on top of delivery channels to decide whether to forward an event to the listeners. The use of this method requires a strong naming and addressing scheme. The events available on a channel must be well understood, and filtering takes place on meta-information that describes the nature of the event.

Even though this class of solution is widely exploited in current distributed systems, it is not suitable as a general solution. A single distribution channel for each possible topic will soon exhaust the resources

available in the network and stretch the ability to administer such topics. This will be particularly true if we foresee a world of pervasive devices that generate trillions of events associated with thousands of different applications. Each network will store an amount of routing states linearly proportional with the number of forwarding topics. The problem is critical in the core of the network [4] but it still presents a problem where routing state is spread across peers[3], because the number of topic is unbounded.

Providing filtering capabilities over less specific distribution channels is not a general solution, as this simply trades processing into the network against inefficiency (as unwanted events are delivered). Filtering capabilities can be overloaded due to the high numbers of rules that they need to implement. The numbers of rules to be processed depends on both the expressiveness of the filters and on the number of subscribers.

The scheme that we describe below provides benefits to all of the problems discussed above. Events may be massively multiplexed on a few multicast channels, but interested receivers are co-ordinated to only join the appropriate channel in order to receive event messages that they are interested in. This allows fewer distribution channels (and hence network state), and removes the requirement to administer what information is delivered over a single channel. The co-ordination of listeners to join the channel only for the period of the event delivery provides a filtering capability at the edge, which is propagated into the network by the underlying multicast join and leave operations. The join action creates a dissemination tree to the listener, which can then torn down after the receipt of the event, leaving the modified tree to be used for another unrelated event to different listeners.

The index-based publish/subscribe approach is very simple in concept. Meta-information about event messages is carried over separate channels from the event messages themselves. This meta-information is called an index, and provides enough information for the listener to be able to detect that an event message is pending on another channel. Each index is very lightweight, and contains information about more than one other event message. In this manner a listener may monitor one index to be directed to many different event messages on a number of additional channels.

To implement index-based publish subscribe we therefore require a receiver-driven channel delivery capability (such as IP multicast), and an announcement protocol that allows announcements on channels to be referenced by index announcements on separate channels. This second capability is provided by the Generic Announcement Protocol and is described below.

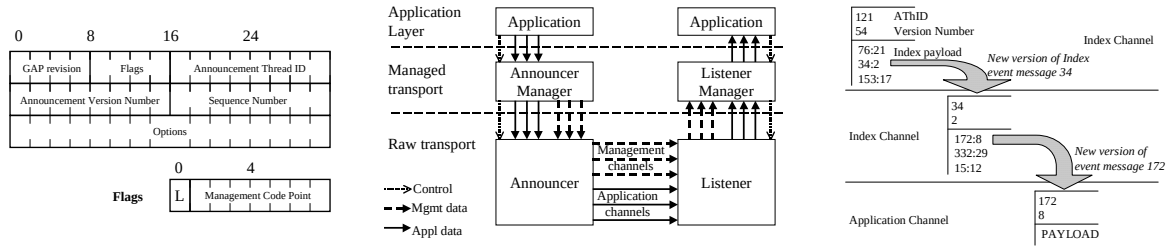
3 GAP: Generic Announcement Protocol

Our goal is to design a basic protocol that is minimal, while providing higher management functionalities that can be evolved, and which provide a platform for the development of additional capabilities and applications. The first purpose is to support event announcements. Given that different applications listen to events that have different properties, the protocol has to provide general capabilities. For example an application interested in all the information delivered over a specific topic channel may persistently listen to that channel using basic GAP functionalities, while if the interest is in only specific announcements, the application may use the index functionality provided by the managed GAP service.

This section describes GAP's design choices and some properties of its specification. We present the GAP announcer structure, its raw service properties, and we describe the managed service with a simple index scheme. The functions of configuration, discovery, index management, security authentication & encryption, archival and recovery are not discussed in this paper.

GAP is implemented as a two tier system (Fig.1b), providing a raw general announcement capability, and optional managed services. To decouple senders from receivers, a globally unique identifier, termed an Announcement Thread Identifier (AThID) is allocated to an event. This AThID generally maps to the changing state of a single object event. Each new announcement within a thread is uniquely identified by an incrementing Announcement Version combined with its AThID. AThIDs can be allocated autonomously, but still with guaranteed uniqueness.

Announcement Structure. A GAP announcement consists of a header containing its AThID and Version (see earlier), and a generic payload that is constructed by the application sending the announcement. The start of the header (Fig.1a) contains a GAP revision number to allow for future protocol changes. The size of the AThID has been chosen such that, in combination with the Announcement Version Number, there is a high probability that the announcement can be uniquely identified within the period in which the Listener is joined to the multicast channel. The *Option* field is available for use by the management functions.



(a) (b) (c)

Figure 1: *(A) GAP message Header. The generic header comes at the beginning of every GAP message, further information can be included in the option field (B) The two-tier structure of the GAP protocol. (C)Managed GAP: indexes coordinate access to announcement information.*

The header also contains a flag field. The first bit of the field is used to signal that the packet is the last (or only) section of a fragmented announcement. The remaining bits are used as a code-point field by the managed GAP service. In our implementation of the managed service, the first bit of the code field is set to indicate an index announcement, while the second bit indicates a configuration announcement. The other flag combinations are available for use by additional management functions for GAP (for example authentication). Finally a sequence number is used to reconstruct fragmented announcements.

3.1 Raw Service

The GAP Announcer receives announcements from applications (or the managed GAP layer) and sends them out on their respective multicast channels in turn. When an announcement is too large to fit into a single IP packet (or MTU of another type of network), it is decomposed into several smaller packets with appropriate sequence numbers, in a similar fashion to TCP. The raw service uses default techniques for reliability and congestion control, both of which are chosen to avoid the need for feedback. But modular implementation allows replacement with alternatives, either under the control of the application, or automatically on detection of lower level capabilities. The default reliability technique is simple repetition. If the announcement is larger than a single packet, erasure codes are interleaved between repetitions. Tornado erasure codes or proprietary codes (e.g. Digital Fountain LT codes [5]) could be used where their efficiency is warranted for larger announced objects. Repetitions also serve the function of loosening the synchronisation requirements between indexes and application announcements.

3.2 Managed Service

The raw service only joins a channel when it is told. Making this decision is the role of the index function of the managed service. Index announcements are identified by the index code-point in their header. They are constructed by the GAP Announcer, and consumed by the GAP Listener.

Index Announcements. An index announcement is associated with a number of other announcements. Its payload consists of a table with an entry for each associated announcement, listing each AThID and its latest Version. As with all announcements, the header of each index announcement also has its own AThID and Version. When the payload of an index changes, so must its Version Number. As indexes are themselves announcements, they may be associated with other indexes in a hierarchical manner. An announcement thread may be associated with multiple indexes, forming a graph, which must be acyclic.

Application announcements with a common Listener population may be, but do not need to be grouped onto the same multicast channel. This is because listeners join and leave application channel(s) repeatedly, regardless of the spread of announcements across the channels. Index announcements are typically announced onto a separate multicast channel from any of the announcements they index. The multicast channel associated with each announcement does not need to be included in index announcements, as each such association would have been locally stored when the announcement thread was originally discovered.

A Listener may choose to listen for a number of index announcements (Fig.1c) delivered over a management channel instead of listening to busy application channels. Since an index payload includes a number of associated announcement threads, the number of bandwidth amount of index announcements monitored will be less than the bandwidth number of application announcements. This has the effect of listening to less multicast channels, which are also less bandwidth hungry (since typically an index payload will be less than most application payloads).

Index announcements may be constructed by the application sending the application announcements. It is also possible for indexes to be automatically generated based upon sampled feedback from the Listener population, and the clustering of their interests. This makes it much easier to implement applications that use GAP. It is also particularly useful when a large number of applications are all sending a small number of different announcements to a common Listener population. This is because automated index construction can occur over announcements sent by different applications, with no knowledge of the meaning of the announcements.

4 Implementation and Performance

The index-based publish/subscribe approach has been implemented in an event notification prototype in BT Exact. The data and index channels are delivered over PIM-SM multicast [6] enabled FreeBSD routers. The announcements are sent, indexed and received by a multi-tier middleware framework, providing basic announcement capabilities, along with managed services to the application layer. The middleware can provide automated indexing based upon elicited feedback. A clustering algorithm is used to construct indexes that reference data announcements destined for a similar listener population. This reduces the number of indexes required by an individual listener, while maintaining the overall number of indexes at a manageable level.

Our analysis (results not shown in this article) through simulation and prototype implementation have shown that a global event notification system implemented with GAP can increase scalability properties compared to current solution. If events are generated with low frequency the resources maintained in the network are optimized, since receiver do not permanently subscribe to a specific information channel. Furthermore this protocol provides an improvement on current solutions since multicast channels can carry a broad variety of topics.

5 Conclusions

We have presented the Generic Announcement Protocol as a solution to scalable event messaging utilising an index-based approach. GAP provides an efficient basic channel announcement service, along with extensible managed service modules. A managed index service can provide and extremely efficient method of monitoring event messages by co-ordinating the access of groups of listeners to the channels as the events that they are interested in are delivered. This approach reduces the state maintained in the network, while providing a filtering capability to the edge receivers. Since the channels, and the event IDs are meaningless, there is no requirement for centralised agreement, and the semantic mapping of events to application or business meaning can occur locally.

References

[1] N.H.Cohen, A.Purakayastha, J.Turek, L.Wong,D.Yeh "Challenges in Flexible Aggregation of Pervasive Data", IBM research report, RC21942, January 2001.

[2] http://www.tibco.com, The Infomation Bus- An architecture for extensible distributed system, SIGOPS 1993 M. Adler, Z. Ge, J. F.

[3] M. Castro, P. Druschel, A-M. Kermarrec and A. Rowstron, "SCRIBE: A large-scale and decentralised application-level multicast infrastructure", IEEE Journal on Selected Areas in Communications (JSAC) 2002.

[4] Aiguo Fei, Jun-Hong Cui, Mario Gerla, Michalis Faloutsos, Aggregated Multicast with Inter-Group Tree Sharing. In Proceedings of NGC2001, UCL, London, UK, November 7-9, 2001

[5] J. Byers, M. Luby, M: Mitzenmacher and A. Rege, "A Digital Fountain Approach to Reliable Distribution of Bulk Data", Proceedings of ACM Sigcomm '98, Vancouver, Canada, September 1998.

[6] Deering, Estrin, Jacobson et al, "Protocol Independent Multicast-Sparse Mode (PIM-SM): Motivation and Architecture" draft-ietf-idmr-pim-arch-01.ps