Scalable Group Communication System for Scalable Trust

Krzysztof Ostrowski
Cornell University
Ithaca, NY 14850, USA
krzys@cs.cornell.edu

Kenneth P. Birman
Cornell University
Ithaca, NY 14850, USA
ken@cs.cornell.edu

ABSTRACT

Programmers of large-scale trusted systems need tools to simplify tasks such as replicating services or data. Group communication systems achieve this via various flavors of reliable multicast, but the existing solutions do not scale in all major dimensions. Typically, they scale poorly in the number of groups; yet we believe that using groups casually could lead to new, easier ways of programming. We propose QSM [1], a new multicast substrate that scales in several dimensions at once. Our approach relies on a novel way of exploiting the overlap between groups.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Reliability

Keywords

Group communication

1. MOTIVATION

Programming distributed systems is known to be difficult; the need to deal with node failures or unresponsiveness, packet losses and delays, forces a developer to anticipate and handle a variety of scenarios, which often obscures the problem being solved. Developers need tools that simplify this task, by implementing and encapsulating the most common design and programming patterns, and thus allowing a developer to focus on higher-level problems. Trusted systems must be built from components that behave in reliable and predictable ways; in practice this often means that components and functionality are distributed and replicated, to allow the system to tolerate failures. A developer of a trusted system, therefore, is typically faced with the challenge of coordinating a number of components distributed over a wide area, ensuring consistency of the replicated data, consistent actions taken by the system components, which may require the components to reach various forms of agreement on which actions are taken, by whom, in what order, against which version of the data, and how the entire system reconfigures upon failures.

Traditionally, these issues have been addressed by *group communication systems* (GCS); problems related to replicating data and services are often either equivalent to, or greatly facilitated by various forms of reliable multicast. However, few of these systems have been designed for or evaluated in environments that might include thousands of nodes. Furthermore, most work on scalability of GCSs has been focused on the number of nodes in the system; less so on the total number of groups. Yet if group communication and reliable multicast were to be used casually, as a basic design pattern and a fundamental programming tool for building trusted, reliable systems, we expect that the number of groups could be very large. Separate groups could be defined for individual products or categories of products, types of services, events or user requests. At a stock exchange, a separate group might be defined for each stock, including all servers that process transactions and calculate prices for the particular stock, etc. We believe this approach might lead to new, easier ways of programming.

We are not aware, however, of any existing GCSs that could scale to thousands of nodes and thousands of groups simultaneously. Existing solutions are also often hard to use and unintuitive, not integrated well with existing standards, limited to a single platform. Although a variety of publish-subscribe and content delivery systems have been designed for very large scale deployments, some of them well integrated and interoperable, even standardized (e.g. WS-Eventing, WS-Notification), such systems lack the strong reliability properties of the GCSs. As a result, developers of trusted large-scale systems still lack tools that, like GCSs in small systems, could significantly simplify their programming tasks and hide complexities.

Is a programming model that assumes casual use of groups in very large systems at all feasible? We believe it is, and our research is focused on building such platform. In this paper, we report on QSM [1], a new protocol that offers throughputs comparable to the raw network bandwidth at the scales of up to 110 nodes and 8192 groups at which we evaluated it, and that tolerates many classes of perturbances, such as bursts of losses, churn, crashes or unresponsiveness, such as triggered by garbage collection etc. QSM is a scalable transport layer with a simple, but still useful reliability property based on periodic reporting of ACKs and NAKs. We generalized QSM to support virtual synchrony, and we are working on supporting consensus and transactional semantics. However, in this paper we focus on QSM, as the most complete[1] and the best evaluated ([1], [9]) component of our architecture.

2. OUR APPROACH

We recognize that in systems with large numbers of groups, the individual groups may heavily overlap (Figure 1, left). The overlap

[1] QSM is currently available for download [7].

may not always be regular. The numbers of groups the individual nodes are members of could vary, and so could the numbers of nodes with "similar" overlap. Yet in many scenarios, such as when members include nodes of a cluster, or service replicas, regular overlap patterns might arise. Overlap hints to the possibility of sharing workload, yet most existing protocols do not benefit from it. In QSM we achieve scalability in the number of groups by exploiting the overlap to reduce the per-group overheads in both data dissemination and loss recovery.

How can overlap be used to improve dissemination performance? *Lightweight group systems*, such as Spread [3] or the Isis Toolkit, map the application (*lightweight*) groups to a smaller set of *heavyweight groups*, multicast in the latter and filter on reception.

Overlap permits batching. Messages destined to a large number of lightweight groups mapped to the same heavyweight group can be transmitted in a single packet. However, filtering incurs overhead. Also, such systems often rely on infrastructure nodes (*agents*) to relay messages, which increases latency, and in certain scenarios may lead to bottlenecks. Can we benefit from across-group batching without the need for filtering or infrastructure nodes?

In QSM, we achieve this by mapping groups to regions of overlap. A *region* is a set of nodes that are members of the same groups; formally, nodes **x** and **y** are in the same region iff **G**(**x**) = **G**(**y**), where **G**(**x**) is the set of all groups of which **x** is a member. Each node is normally[2] in a single region (Figure 1, right). QSM employs a *Global Membership Service* (GMS) to process all group subscriptions, and to manage the group membership. The GMS determines the region boundaries and provides all members with consistent group and region membership notifications. To track membership, QSM uses a 2-level structure, in which both groups and regions are versioned (Figure 2).

Each region is then assigned a separate IP multi-cast address. Nodes in the region subscribe to that address, and the sender that wants to transmit data to a group, transmits the message to each of the regions spanning over it. Messages that are addressed at different groups that overlap on the given region can be batched together, as it was the case with the lightweight groups. In QSM, however, groups do not cut across the region boundaries (each group includes either all nodes in a given region or none), hence filtering is not necessary.

The efficiency of this dissemination scheme heavily depends on the pattern of overlap. If a group consists of many small regions, the resulting communication pattern resembles application multicast. For such scenarios, QSM offers an alternative mode of multicasting to a per-group IP multicast address, the mechanism used in most GCSs, where support for multiple groups is realized by running separate, independent protocol instances side-by-side. The resulting protocol is a hybrid one, with dissemination on a per-group and recovery on a per-region basis (for details, see [1]).

[2] More precisely, in a single *region view*, and only during the periods of stability. When membership is changing, a node may temporarily be a member of multiple different *region views*, but only very briefly, until protocols in the old views quiescence.

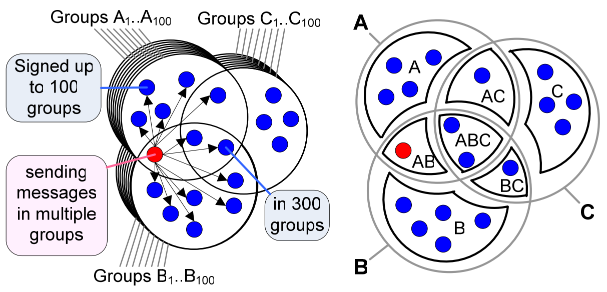

Figure 1. Left: A very large number of heavily overlapping multicast groups. Right: Their "regions of overlap".

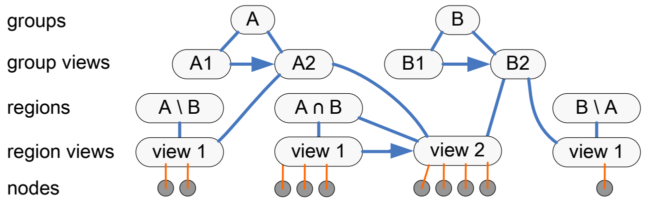

Figure 2. A two-level membership scheme used in QSM.

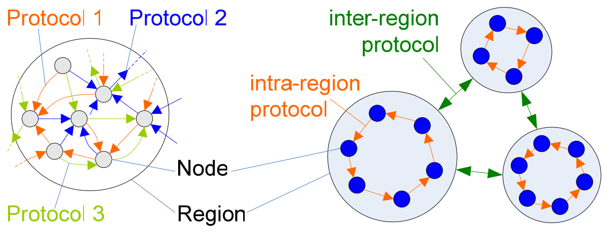

Figure 3. Left: Separate protocols running in different groups are overlapping on a set of subscribers. Right: The multi-level protocol architecture used in QSM.

There is another factor besides the opportunity for across-group batching that favors the per-region IP multicast scheme to the per-group IP multicast in systems with very large numbers of groups. Our experiments show that if a node subscribes to a large number of IP multicast groups, it is disturbed even by network traffic destined to groups that it is not a member of: packet filtering starts to involve the OS.

A similar approach to multicasting, permitting more flexibility in the way regional mappings are defined, has independently been explored in [6], but in QSM, we are also concerned with reliability. QSM regions have important characteristics that we heavily rely upon in our loss recovery and rate control protocols, and which we shall refer to as interest sharing and fate sharing. The former refers to the fact that nodes in a region are, by definition, members of the same groups, and hence receive the same data and participate in recovery protocols in the same groups. Fate sharing refers to the fact that nodes in a region experience similar workloads and churn rates resulting from other nodes joining or leaving groups, and may experience correlated bursts of packets or losses.

How can interest or fate sharing benefit recovery? To understand this, let us first look at how overlap affects the majority of GCSs: those that run separate protocol instances for each group, side-by-side. Such systems typically organize their nodes into trees and other peer-to-peer structures and distribute the task of processing acknowledgements and loss recovery among the receivers to offload the sender. The key is to ensure that each node interacts with a small number of neighbors and hence is not burdened by a high volume of incoming packets. However, if creation of trees and other structures is not coordinated among groups, a node may still have a large number of neighbors, and the benefits may be lost (Figure 3, left).

Conversely, if work performed by different protocols could be shared, such as by creating peer-to-peer structures in a way coordinated among groups or packing control messages related to the different groups in a smaller number of packets, overheads could be reduced. The idea of sharing workload in this way was explored in [5], to achieve scalability in the number of senders. In QSM, we use it to reduce the per-group overhead.

In QSM, loss recovery is hierarchical, somewhat inspired by RMTP [4]. First, nodes in every region run a local intra-regional protocol (a "bottom" protocol) to perform local recovery and to calculate aggregate ACK/NAK information for the entire region. Then, another ("upper") protocol is run across all the regions the given group spans over (Figure 3, right).

In QSM, regions simply communicate their aggregate ACK or NAK information to the sender, to permit cleanup or request retransmission for the entire region, but our approach can be extended to allow the regions themselves to perform inter-regional recovery. In our other paper [2], we propose a flexible architecture for hierarchical stacking of loss recovery protocols, inspired by our work on QSM, in which we describe how the idea of running peer-to-peer recovery protocols across distributed entities such as QSM regions can be realized efficiently.

The two-level recovery scheme just described allows for workload sharing. Since all nodes in a region are members of the same groups, each packet transmitted to the region, irrespectively of the group it was destined to, or its source, must be delivered to all nodes. In QSM, a single recovery protocol is used in a region for packets in all groups, for all senders.

Senders in QSM number packets they transmit to regions on a per-region, rather on a per-group, basis. Region members can view all messages multicast to their region by the given sender in multiple different groups as a single, contiguous sequence. The group information is still present in a message to deliver it to the application in the appropriate context, but it is completely ignored during recovery. This way, QSM offers performance that is, by design, virtually independent of the number of groups.

Likewise, each packet in the local recovery protocol carries multiple recovery records, for each of the senders that are currently actively multicasting into the region. This way, control overhead can be amortized also across senders, as in [5].

Note that neither of the above would be possible had region members received different sets of packets, for in that case some nodes would be asked to participate in repairs for packets that they were never meant to see, an inefficiency analogous to filtering.

Fate sharing can be used for purposes such as rate control. Because all nodes in a region experience the same load incurred by the incoming traffic, rate control can be implemented by letting each node deter-mine the maximum rate it can sustain (e.g. based on the rates of successfully received multicasts, its own CPU load, the rate of its own packet losses etc.), calculating an average or minimum of such rates across the entire region, and finally splitting the resulting value across all senders, using a fair-sharing or other policy. Senders then use the received values to tune their per-region rate controllers. Although still work in progress and not included in the current release of QSM, this scheme was implemented and tested, the approach is feasible and very efficient. The remaining challenge is to correctly estimate the admissible rates for the individual nodes in certain scenarios. Additional details of our rate control and recovery schemes can be found in [1] and [9].

The efficiency of the scheme we proposed strongly depends on the regularity of overlap. QSM works best when groups overlap fairly regularly, thus forming large regions. This would be the case, for example, in a data center, if groups of cluster nodes with a similar hardware and software configuration are designated to perform similar tasks, and consequently often process the same data and join the same groups. We believe that many systems can be architected in a way that promotes regular overlap. We view QSM as a tool that allows a system architect to exploit any such regularity to reduce overhead and gain performance. This might, of course, require extra effort on behalf of the system architect. However, in general no protocol can achieve sub-linear scalability of overhead in the number of groups "for free". Contrary to most existing protocols, QSM makes such sublinear scaling at least possible, even though achieving it requires the system to exhibit some degree of regularity, and in different scenarios, in different parts of the network, the benefits may vary.

In the extreme, if overlap is totally irregular, QSM regions could be as small as even single nodes. In such scenarios, QSM would degenerate to just the "upper" protocol (the "bottom" protocol would still "run", but in a degenerate form that does not involve communication, and that incurs negligible overhead). It would be essentially equivalent to application multicast. Although the upper protocol is very simplistic, and the performance in such scenarios would be low, our approach can be generalized. In [2] we outline a hierarchical architecture for reliable multicasting, inspired by QSM, in which any of a very large class of protocols expressible in a framework presented there can be stacked into hierarchies.

3. REALIZATION

While implementing QSM, we identified various phenomena and principles that we believe are essential to achieving scalability. The full account is given in [1]. Here we present a brief summary.

The first issue, which we refer to as *relative asynchrony*, is related to the fact that nodes run at different speeds; are unevenly disturbed by events such as garbage collection or by the scheduler; receive packets at different times, or in a different sequence. As explained in [1], this phenomenon has several negative consequences that result in large buffering over-heads and suggest the need for locality. We respond to this by splitting every region into multiple **k**-node partitions responsible for buffering disjoint subsets of the arriving packets. Recovery in every region is then performed in a hierarchical manner: first, a local recovery protocol (a token ring) runs among all nodes in every partition, and then, another protocol (also a token ring) runs across all partitions within a region, much like on Figure 3 (right). This leads to a 3-level protocol hierarchy, with protocols running in groups, regions, and partitions.

Another issue is related to the scheduling and priority in handling different sorts of events. We found that when data is exchanged at high throughputs, the system resembles a crowded highway, where highest speeds are easier to sustain when distances between the vehicles are larger. If the number of vehicles increases, they have to follow each other more closely, and maintaining safe distances becomes more difficult. Slow obstructions can slow the individual cars down; such oscillations may ripple through the traffic and cause a traffic jam or an accident. If this occurs, it is essential that emergency vehicles can get through the traffic faster and correct the problem. In QSM, disruptions such as those caused by a garbage collector can similarly cause packets to pile up. This leads to convoy effects and various sorts of priority inversions unobservable at lower speeds, and hard to diagnose and debug. In [1], we explain how we dealt with these issues by replacing the unpredictable preemptive scheduling with our own scheme that permits control traffic to be processed faster and architected our protocol stack to ensure that control packets sent over the network are always fresh, i.e. they are generated just in time for transmission and hence they reflect the most recent state. Prioritizing of control traffic and ensuring "freshness" turned out to be critical for ensuring the stability of our protocol.

As shown on Figure 4, in an experiment in which all groups completely overlap QSM scales extremely well with the number of groups; the decrease in performance results mostly from the increasing memory consumption caused by the packets queued in thousands of per-region sending buffers on the sender.

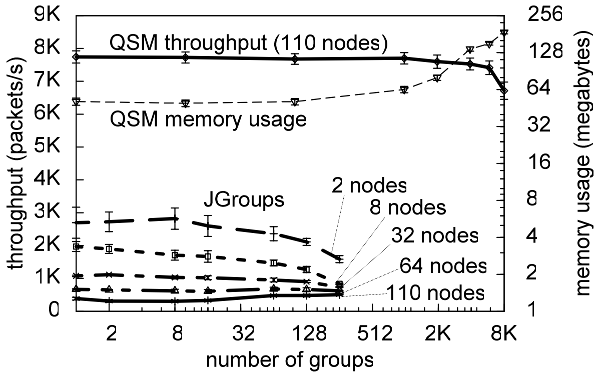

Figure 4. Scaling in the number of groups. All groups have identical membership and overlap on a single region spanning over the entire system. With a different overlap, performance may vary (see [1]). For comparison, we present performance results[3] for JGroups, a popular group communication system.

4. CONCLUSION

Our work on QSM suggests that scaling to thousands of groups is practically feasible. Although currently QSM supports only basic reliability and further work is needed to optimize it for various irregular overlap patterns, we believe that QSM represents the first successful step towards a platform that enables programming based on a casual use of groups.

QSM also shows that a hierarchical stacking of protocols that run at different levels, such as among nodes, partitions or regions, and sometimes on behalf of multiple groups at once, although fairly complex and requiring dedicated infrastructure, can be engineered to run very efficiently: in our experiments ([1]), QSM nodes send or receive thousands of multicasts per second while utilizing only a fraction of their slow 1.3GHz CPUs. We believe that this object-oriented way of thinking about protocols, as running among nodes or groups of nodes, and providing semantics at different levels, can be a powerful tool. In [2] we propose a multicast architecture that extends QSM, and that allows different "lower" protocols to run in different parts of the network, and yet form a single dissemination structure thanks to the "upper" protocols running at higher levels in the hierarchy, with reliability properties spanning over the entire system. The ability to locally fine-tune protocols in parts of the network allows for additional optimizations and promotes interoperability. In [8] we go even further, and propose a different approach to scalable group communication, based on our "object-oriented" approach to modeling protocols, where protocols can be expressed using sets of rules and automatically translated to form hierarchical structures resembling QSM.

5. ACKNOWLEDGMENTS

Our effort is supported by AFRL and Cornell Information Assurance Institute. We are grateful to Mahesh Balakrishnan, Danny Dolev, Maya Haridasan, Tudor Marian, Stefan Pleisch, Robbert van Renesse, and Einar Vollset, who offered a great many comments and suggestions over the course of the project.

6. REFERENCES

[1] K. Ostrowski, K. Birman, and A. Phanishayee. QuickSilver Scalable Multicast. April 2006. http://www.cs.cornell.edu/~krzys/QSM-2006.pdf

[2] K. Ostrowski and K. Birman. Extensible Web Services Architecture for Notification in Large-Scale Systems. In *Proceedings of the IEEE International Conference on Web Services (ICWS '06)*.

[3] Y. Amir, C. Danilov, M. Miskin-Amir, J. Schultz, and J. Stanton. The Spread Toolkit: Architecture and Performance, 2004.

[4] J. C. Lin and S. Paul. RMTP: A Reliable Multicast Transport Protocol. *INFOCOM*, 1996.

[5] B. Levine, D. Lavo and J. J. Garcia-Luna-Aceves. The Case for Reliable Concurrent Multicasting Using Shared Ack Trees. *ACM Multimedia*, 1996.

[6] Y. Tock, N. Naaman, A. Harpaz, and G. Gershinsky. Hierarchical Clustering of Message Flows in a Multi-cast Data Dissemination System. *PDCS*, 2005.

[7] QuickSilver Scalable Multicast project website at Cornell: http://www.cs.cornell.edu/projects/quicksilver/QSM/

[8] Krzysztof Ostrowski, Ken Birman, and Danny Dolev. Properties Framework and Typed Endpoints for Scalable Group Communication. July 2006. http://www.cs.cornell.edu/~krzys/PropertiesFx.pdf

[9] Krzysztof Ostrowski, Ken Birman, and Amar Phanishayee. The Power of Indirection: Achieving Multicast Scalability by Mapping Groups to Regional Underlays. November 2005. http://www.cs.cornell.edu/~krzys/QSM-2005.pdf

[3] Results for JGroups were collected by Amar Phanishayee.