

# Efficient Hierarchic Management For Reconfiguration of Networked Information Systems

**J.C. Rowanhill, P.E Varner, J.C. Knight**  
Department of Computer Science  
University of Virginia  
151 Engineer's Way, P.O. Box 400740  
Charlottesville, VA 22904-4740  
(434) 982.2216 - Voice (434) 982.2214 - FAX  
{jch8f, pev5b, knight}@cs.virginia.edu

## Abstract

The management of modern distributed systems is complicated by scale and dynamics. The continued effectiveness of an authoritative ('top-down') approach depends on its ability to handle complexity. Additionally, distributed authoritative management must support interaction with distributed cooperative approaches, such as found in peer-driven algorithms.

Scalable, decoupled communication establishes flexible, loosely coupled component relationships. These relationships would help meet the present demands on authoritative management. However, traditional decoupled addressing mechanisms focus on only one of the parties involved in communication. While in general, a communication relationship involves a sender, communicated content, and receivers. The state of all three are simultaneously relevant to correctness of a management relationship and its communications.

We introduce Selective Notification, a scalable, decoupled event dissemination architecture supporting simultaneous and combined addressing of senders, receivers, and events. We demonstrate its application to programming dynamic, scalable management relationships. We then discuss its implementation, and present measurements of its effective capabilities.

**Submission Category:** Regular paper.

**Total Word Count:** 5,300

**Contact author:** J. C. Knight.

**Cleared:** Paper has been cleared for publication

**Keywords:** scalable decoupled communications, event dissemination architecture, loosely coupled system, loosely coupled management

# Efficient Hierarchic Management For Reconfiguration of Networked Information Systems

## 1. Introduction

Very large networked information systems—with millions of components—have become crucial to many organizations, both military and civilian. Yet they are inescapably exposed to a wide variety of traumas including extreme environmental conditions, failures of operating software, and losses of available resources because of malicious or accidental damage. In order to provide dependable service such networks have to respond to these changes with explicit management because, without response, dependability would be limited by entropy. The required responses may be large and complex, necessitating a sophisticated management service architecture.

In this paper we introduce a communication mechanism for facilitating the distributed management of networked information systems, *Selective Notification*, that provides symmetrically addressed, decoupled event dissemination. It permits reconfiguration to be commanded quickly, efficiently and in a highly scalable way. Coupling this with a sensing and analysis capability yields an architecture for hierarchical management in support of general fault tolerance mechanisms.

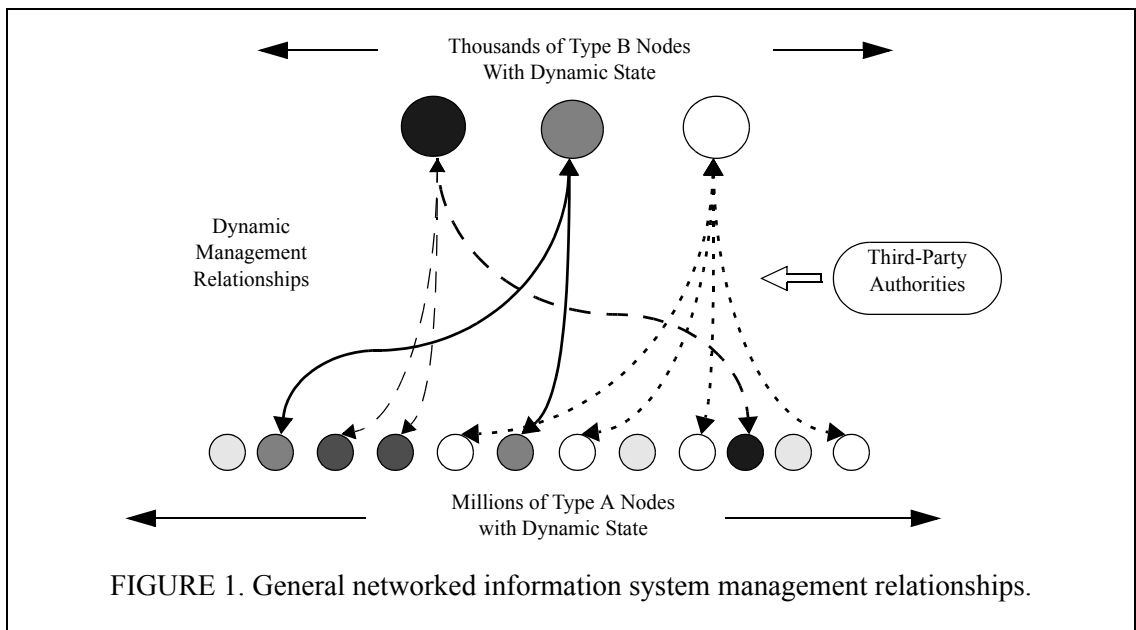
Researchers are pursuing more dynamic and less hierarchical management structures. Yet there is still merit to the hierarchical approach because of its ability to rapidly convey non-local attention. The combination of hierarchical and widely distributed management has significant potential in complex systems, and more generally, management hierarchy must apply to increasingly large and dynamic information systems [10].

The elements of an approach to the hierarchical management with increased dynamics may be found in the form of a *loosely coupled system* [5]. Our proposed mechanism is a general architecture, illustrated by the intentionally general example of Figure 1. A very large collection (millions) of nodes of type A (e.g., those requiring management) operate in coordination with a smaller collection of nodes of type B (e.g., those determining management actions). The hierarchical relation-

ships between them are established *dynamically* and *fluidly*, being based on the current state of participants and third parties. In this example, nodes interact with those of the same shade, where shade indicates some aspect of their states. As their modelled state, or shades, change, their management relationships are automatically updated to reflect appropriate connections. Thereby, these relationships remain current and appropriate over highly dynamic system state. This occurs transparently, without requiring any nodes to know the state of any other.

Consider the nodes of Figure 1 as managers over a large distributed system. In general, the appropriateness of their intercommunication might involve any combination of the state of (1) communication senders (2) communicated content, and (3) receivers. This requires a communication service addressing all three elements. It is in their simultaneous combination that loosely coupled management relationships may be achieved. By contrast, existing scalable services such as publish/subscribe support asymmetric addressing. The questions we address are whether a more symmetrically addressed mechanism can scale with reasonable performance and whether it can be used effectively for expressing management relationships.

Selective Notification is a symmetrically addressed, decoupled event service. We claim that it deals with both questions listed above. Our argument begins with a synthesis of Selective Notifica-



tion’s core concept— symmetric indirect addressing. We then demonstrate its utility through application in a hypothetical management scenario. This is followed by exposition of its implementation as data transforms as well as modifications and extensions to Siena [2], a scalable publish/subscribe architecture.

Our argument for feasibility is based on experiments with a full implementation. The results of these experiments allow us to model its performance for systems far larger than we can directly implement. We conclude that symmetrically addressed decoupled communication scales for hierarchical event dissemination in loosely coupled management.

## 2. Selective Notification

Clients of decoupled communication interact without having knowledge of one another. More specifically, *spatially decoupled communication* [5] allows clients to interact despite continually not knowing each other’s location, quantity, distribution, or state. Property-based communication is a particularly useful form. It allows otherwise decoupled components to communicate by describing—rather than explicitly naming—relevant objects in communication relationships. Communication in this form involves two key elements:

- *Property addressable objects*: Some objects advertise a model of their properties to the communications service. An object’s properties constitute its address.
- *Descriptive, indirect addresses*: Clients communicate by describing, (often through constraints) properties of addressable objects. A description forms an indirect address of the relevant targets for communication.

Scheme	Addressed Object	Addressing Object
Content-based Publish Subscribe	Events	Receivers
Intentionally Addressed One-to-many Messaging	Receivers	Senders
Sender Qualified Messaging	Senders	Receivers

**Table 1:** Summary of traditional, decoupled, property based addressing schemes.

Several forms are in use today. They differ in which objects are addressable and which objects perform addressing. Three common forms are summarized in Table 1. They are content-based publish/subscribe [3,4], intentionally addressed one-to-many messaging [1], and sender qualified messaging.

## **2.1 Indirect Addressing for Specifying Management Relationships**

Now consider application of the decoupled addressing mechanisms of Table 1 to a loosely coupled hierarchy such as that illustrated in Figure 1. A node undertaking reconfiguration might apply intentional addressing to indirectly target an event to managed nodes, by describing the states of their internal security alarms for example. Likewise, managed nodes might apply sender qualification to describe necessary properties of high-level managers from which they will receive commands, requiring proper authority for example. However, the architectures of Table 1 do not supports delivery of communication events based upon simultaneous consideration of all three addressing mechanisms.

## **2.2 The Concept of Selective Notification**

Selective Notification combines content, sender, and receiver addressing in a unified, simultaneously applied addressing mechanism, and permits a scalable implementation. Our implementation has two limitations: not all clients are supported as simultaneous senders if efficiency is to be maintained, and it necessitates more traffic in the overlay network than is required for content-based forwarding.

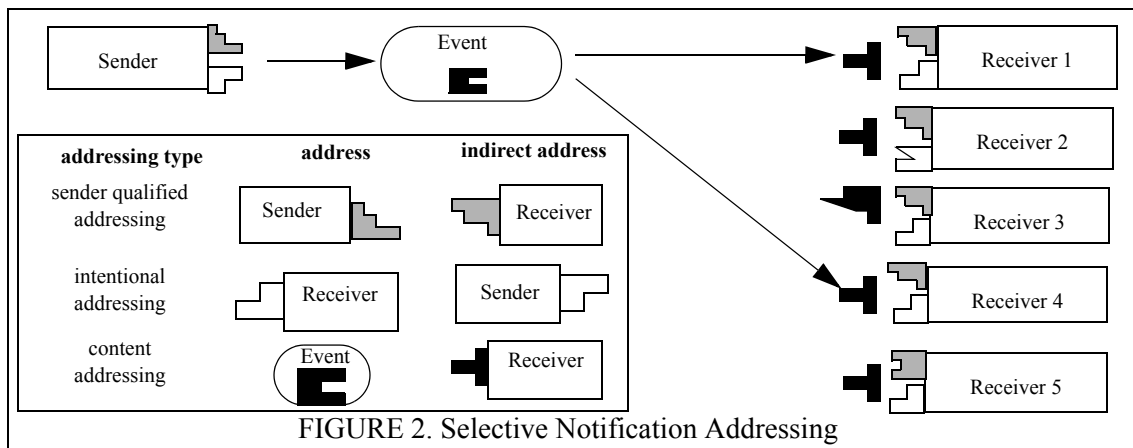
Event delivery in symmetric decoupled communications is illustrated in Figure 2. A message sending client is shown on the left and an array of potential receivers clients are shown on the right. Both senders and receivers *advertise* their local state to form their respective addresses. In the figure, each client's advertised state is represented by an attached "puzzle-piece". Sender addresses are lightly shaded, while receiver addresses are white. Receivers subscribe to messages

with content of interest. The characteristics that define content of interest are depicted by a black puzzle-piece at each receiver. Receivers may also require sender qualifications. These are presented in the figure as lightly shaded puzzle pieces at receivers.

Senders push events to the communications system. Figure 2 shows a sender emitting an event. It's content is a u-shaped black puzzle-piece. In the notation of this figure, an address matches an indirect address when their respective puzzle-pieces 'fit together'. Selective Notification delivers an event to a receiver if and only if intentional addressing, sender qualification, and content addressing match. In this example, the sender's event will be delivered to receivers 1 and 4. The remaining receivers mismatch in one or more element of addressing.

### 2.3 Addressing Language Specification

Selective Notification's addressing is specified using languages defined by the grammars shown in Figure 3. In general, addresses are typed attributes and indirect addresses are constraints over their values. The goal symbols of the grammars are "Receiver Policy"s and "Selective Notification"s. Clients subscribe Receiver Policies and send Selective Notification events. Receiver Policies include a receiver's address, as well as content and sender indirect addressing. Selective Notifications include event content (address) and indirect addressing of receivers. The most restricted element of the grammar is the Selection Function—applied to achieve intentional addressing. Selection Functions must be known to both senders and receivers.



### 3. Related Work

Several research groups have applied decoupled communication for the purpose of management, such as Software Dock [7] and Astrolab [13]. To our knowledge, the importance of *symmetry* in establishing loosely coupled management relationships has not been discussed.

Skarmeas et al. [11] design a symmetrically addressed decoupled communication mechanism in the form of an agent blackboard. It was not designed as a scalable architecture.

Overlap in potential between intentional addressing and publish/subscribe has been applied in many applications, including sensor networks [9] and control of robots [6]. Designers of scalable communication mechanisms have also noted this potential. For example, Siena has been modified to support intentional addresses [8], while the authors of the Astrolab system have proposed using Bloom filters to achieve publish/subscribe [12].

<b>Attribute Data Types:</b> integer, floating point, Boolean, String, finite-domained set	
<b>Syntax</b>	<b>Semantic Constraints</b>
op → <   ≤   >   ≥   ≠   =   prefix   postfix   ⊆   ⊇	
compare → <i>attributeName</i> op <i>value</i>	type compatibility
conj → compare   conj ∧ conj	
exp → conj   exp ∨ exp	
senderQualifier → exp	<i>attributeNames</i> from sender clientModels
contentConstraint → exp	<i>attributeNames</i> are from anticipated events
comp2 → <i>parameter</i> op <i>attribute</i>	type compatibility
conj2 → comp2   conj2 ∧ conj2	
exp2 → conj2   exp2 ∨ exp2	
selectionFunction → exp2	<i>attributeNames</i> are from a single clientModel
t1 → contentConstraint   <i>true</i>	
t2 → senderQualification   <i>true</i>	
t3 → selectionFunction   <i>true</i>	
triplet → t1 ∧ t2 ∧ t3	
<b>ReceiverPolicy</b> → triplet   ReceiverPolicy ∨ ReceiverPolicy	single receiver-side clientModel
attribute → <i>attributeName</i> = <i>value</i>	type compatibility
attributes → <i>null</i>   attribute   attributes , attribute	
event → attributes	
Selector → <i>SelectionFunctionName</i> (attributes)	attributes are parameters of the Selection Function
<b>SelectiveNotification</b> → event   Selector , event	

FIGURE 3. Language of Selective Notification addressing.

Our work focuses on demonstrating the importance and feasibility of symmetric addressing in decoupled communications. This requires simultaneous and scalable support for combined use of three indirect, property-based addressing mechanisms.

#### 4. An Example Application

We now describe a hypothetical application for Selective Notification. Returning to the illustration of Figure 1, assume a million low-level managers are embedded within a global distribution of Internet Web servers. Assume a few thousand high-level managers are run by service providers. Consider the Web servers to be owned by a fictitious corporation, ‘Macrocorp’. Macrocorp has contracted a fictitious security provider, ‘Intellimmune’, and cooperates with government emergency response systems.

Our hypothesized scenario assumes the spread of a flash-worm. Like most worms, this one exploits a bug in network code to self-replicate (in this case, in Web servers). Left to its task, the worm might undermine Macrocorp’s global application. Fortunately, Intellimmune and government systems monitor for attacks against common software applications. Detecting the attack, they manage systems such as Macrocorp’s through symmetrically addressed decoupled communication policies. Our implementation uses an XML syntax but, in this example, a more human-readable syntax is used.

##### *Modelling System State*

Our model of Web servers, i.e., the state that Web servers advertise as their address, is:

```
<Model WebService>=      {String application;           String application_version;
                          String serviceIPAddress;           int servicePort;
                          DomainedSet{docs, cgi, xml} services; float load}
```

Models are named elements consisting of typed, named attributes. Every Web service in our example application presents an *instance* of this model to Selective Notification.

An instance of a model is an assignment of a value to each model attribute, so a Web server somewhere might supply:

```

<WebService>      =      {application="IIS";           application_version=2.4.0;
                        serviceIPAddress="128.142.55.55";  servicePort=8080;
                        services={docs, cgi};               load=0.39}

```

This Web server is free to change its attribute values at any time. For example, this Web server will periodically update its load attribute with its latest calculation. Changes can also include servers joining and leaving the system. All such changes are handled automatically by the decoupled communications aspect of Selective Notification.

Now suppose that Macrocorp agrees to respond to government regional fault-response systems. Such high-level systems are not allowed to define their own region of command or trustworthiness. Instead, this state is assigned to fault-response systems by authorized third parties, such as regulated trust manager. For example, a Northwest regional controller might be assigned the following sender qualifications by authorized third parties:

```

<Model FRSystem> =      {String _command_region="northwest";   int _trust_level=4}

```

Restricted sender qualification within Selective Notification allows a tiered-authority model of sender state enforcement, so that increasingly critical state can be managed by increasingly trustworthy elements.

### *Connectivity Policies*

Web servers describe the messages they will receive, and the clients from which they will receive messages, through subscription of Receiver Policies. For example, the Web servers in our hypothetical application register the following policy:

```

<Event>:      {alert==ANY
                AND threat_level>=4}
                AND <Sender=="Controller">:{_command_region=="northwest" AND _trust_level>=2}
OR
<Event>:      {command==ANY}
                AND <Sender=="Controller">:{entity=="Intellimmune" AND _command_region=="national"}

```

which translates to “Observes command events from any national Intellimmune control system and alerts greater than or equal to threat-level 4 from Northwest controllers with trust rating greater than or equal to 2”. Once in place, received events from senders are those for which evalu-

ation of the connectivity policy expression is *true*. Thus, Web servers only receive understandable commands and alerts from qualified commanders.

### *Command and Alert Events*

Assume that a worm has begun propagating through Macrocorp's networks. A fault-response system in the Northwest is the first to detect the worm infection. It determines that the sensor events are all coming from Web servers running version 2.4 of "IIS." First, it reports this to national fault-response systems. Then, it issues a worm alert as an event:

```
Event: {alert="worm"; threat_level=4; target="IIS"; version="2.4.*"}
```

Any receiver of this event will be able to obtain the information associated with it by examining the attributes.

Given that Web servers are enforcing the policy defined above, all Web servers in the Northwest region will receive this alert. From the alert they can determine whether they are vulnerable to attack. Meanwhile, the worm continues to infect the distributed system. Intellimmune attempts to halt the attack with commands to Web servers. Its national fault-response system has determined that the worm is spreading through a vulnerability exposed in CGI-scripts running on version 2.4 of IIS Web servers. Therefore, it issues the following command event:

```
<Model==WebServer>:{application=="IIS" AND application_version<"2.4" AND services supersetOf{cgi}}  
Event: {command="disable_cgi"}
```

This event contains a preamble that is a selector (intentional address). It selects Web servers that are running "IIS" version 2.4 and support CGI scripts. The event itself is a command for those Web servers to disable CGI elements. The goal of this command is to limit the infection by disabling the worm's attack vector.

In an attempt to limit the worm's aggression, Intellimmune emits another command. It has determined that IIS version 2.4 servers showing sustained load are potentially infected. These servers are ordered to shut-down with the following command event:

<Model==WebServer>:{application="IIS" AND application\_version<"2.4" AND load>0.9}

Event: {command="shutdown\_now"}

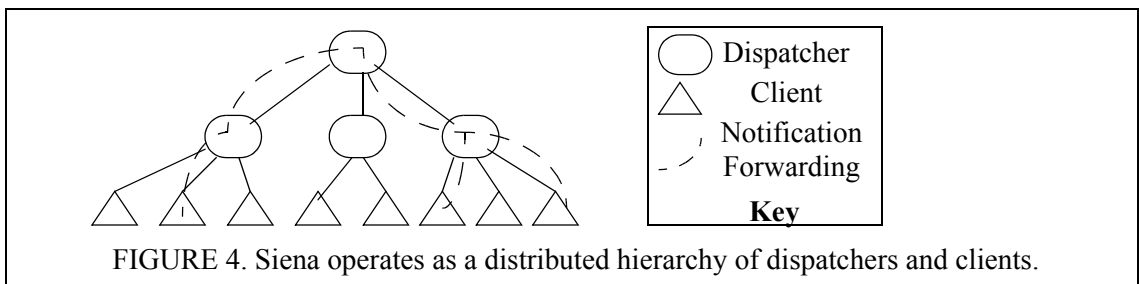
This example has demonstrated the delivery of an alert event and two command events to application components of an Internet-scale system. The connectivity policies between managers address properties of senders, receivers and content. They define a total connectivity policy targeting management at run-time based on the current state of participants.

## 5. Implementation

Our implementation was developed, in part, by modifying the core data structures and algorithms of Siena [2] (Scalable Internet Event Notification Architecture)—a content-based, publish/subscribe infrastructure.

Siena's core data structures are Filters and Notifications. A Notification is a communicated event consisting of a set of typed attribute/value pairs. Filters are Boolean conjunctive expressions over notification attributes. They are used to define content subscriptions issued by potential receivers. Siena operates as a distributed tree of dispatch servers, an example of which is shown in Figure 4. Dispatchers perform two key algorithms. These are:

- *Filter Propagation*: A subscribed filter,  $f$ , propagates up the dispatcher tree until it arrives at the root of the tree, or at a dispatcher with a filter logically covering it. Dispatchers store received filters. Siena scales well when most subscribed filters are covered by others.
- *Notification Forwarding*: A published notification,  $p$ , is propagated up to the root of the dispatcher tree. It is also sent down any sub-tree from which a matching filter was received. As a result, notifications are only forwarded to sub-trees with receivers.



## 5.1 Data Transformations

Selective Notification transforms Receiver Policies and Selective Notifications into Filters and Notifications, respectively. Figure 5 sketches the transformation of data constructs from Selective Notification (part a) to Siena publish/subscribe (part b). Shapes represent data objects. Arrows represent the products of transformations. ‘Plus Signs’ indicate the combination of two data objects in a transform.

Siena already supports content-based addressing. The transformation of sender qualification is straightforward, storing attributes and constraints in notifications and filters, respectively. The transformation of intentional addressing is more complex and best illustrated by example. Consider a client with an attribute called “load” with value “0.3”. If the client advertises the selection function “ $X < \text{load} < Y$ ,” then this is translated to a Siena Filter of the form “ $X \leq 0.3$  and  $Y \geq 0.3$ .” When a sender selects a set of receivers by load, it does so by sending a notification defining values for X and Y.

## 5.2 Modifications to Publish/Subscribe Infrastructure

The second part of the implementation requires modification of Siena dispatcher algorithms and data structures. If it were applied without change to Siena, our transformed input would (1) elimi-

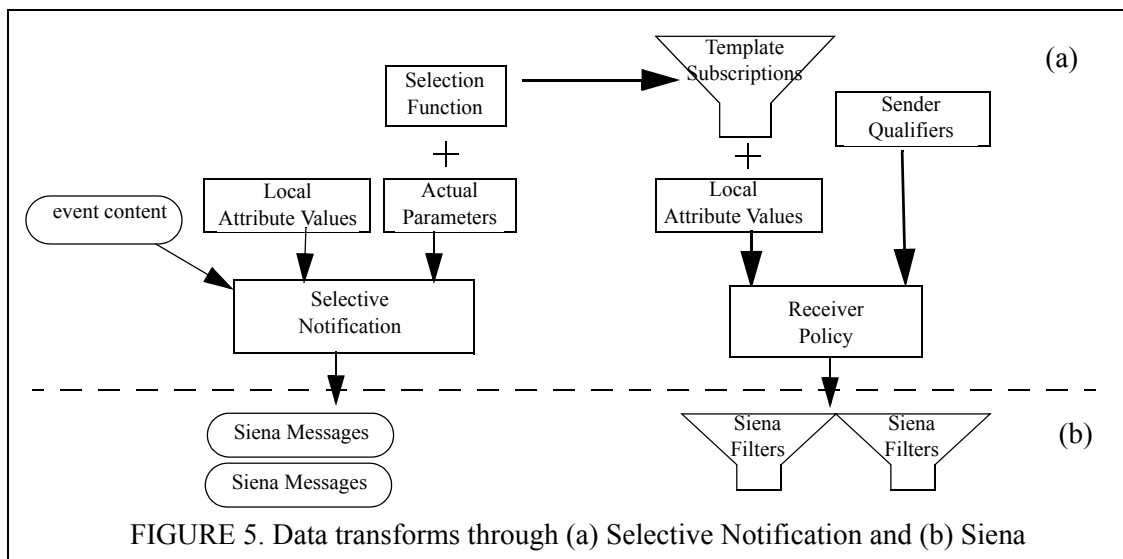


FIGURE 5. Data transforms through (a) Selective Notification and (b) Siena

nate the scalability of Siena filters, (2) fail to deliver most relevant notifications, and (3) allow clients to lie arbitrarily about their attribute qualifications. Only applying data transforms, Selective Notification would be a nearly-pathological application of publish/subscribe. Therefore, significant alterations to algorithms have been made while preserving the two core operations of notification forwarding and filter propagation. Briefly, these alterations and modifications are:

- *Notification Persistence*: Notifications remain at dispatchers for a specified lifetime where they forward to later subscription filters. In this way, the consistent and rapid changing of filters for intentional and sender qualification does not impede reliable delivery of notifications.
- *Filter Coagulation*: Intentional addressing does not generate efficient filter covering relationships. Hence, we apply lossy data aggregation over filters to maintain system scalability. While reducing message forwarding efficiency, our aggregation maintains notification delivery reliability—all receivers obtain all and only relevant notifications.
- *Attribute Authorization and Capability*: Clients of Selective Notification must register for notification and subscription capabilities by ‘login’ with password. This restricts clients to stating attributes in models and notifications for which they are authorized to make claims.
- *Third Party Qualifiers*: We enable third parties to contribute state to client addressing, for example for trust management. Third parties must be given permission, by session key, from the client that they are to augment. Importantly, a third party may have different authorized capabilities than the client it augments. This supports tiered authority models in the use of sender qualification and intentional addressing.
- *Channeling and Event Ordering*: Rather than computing the forwarding path for all events, we allow some events to record their forwarding paths, and others to follow these paths. This allows streams of events to travel to the same set of receivers, even as their state changes.

## 6. Measurements of Performance and a Model of Scale

The essential practical challenge with Selective Notification is scale both in terms of network size and rate of change of addresses. It is important to determine the overall performance of the technique and how that performance relates to the scale of the target system. The issue of performance is complex because performance metrics need to be defined and measured along the spectrum of dimensions that will affect performance in real systems. From the perspective of general utility, we consider the following to be the critical metrics:

- *Sustainable event delivery time*: Time from event issue to event delivery to all relevant clients.
- *Sustainable event throughput*: The sustainable rate at which events can be issued into the service without overloading the service.

With those metrics in mind, the key dimensions that affect performance are:

- The size of the application system as measured by the total number of independent nodes.
- The addressing policies that describe senders, receivers, and content simultaneously.
- The rate of change of the state being presented to the Selective Notification mechanism.

In this section, we present the results of experiments to determine the metrics above for these parameters. From the results, we build implementation performance-driven models of scale.

### 6.1 Experimental Method

We have directly tested overlay networks on a test-bed of 128 physical computers, each of which is a dual 400 MHz CPU i86 machine running Redhat Linux 6.4 with kernel version 2.2. All software was implemented in Java for runtime 1.4.1. Network level communication was performed with TCP sockets over a 100 MBit/sec fully switched Ethernet. Some computers were dedicated to the execution of Selective Notification dispatchers and the remainder were used to execute a hypothetical distributed application. Several application nodes were located on each physical machine and

this permitted a total of several thousand client nodes to be constructed. The actual number of clients varied with the details of the experiments being conducted but was typically 3,400.

Clearly, this target system is not of Internet scale, and so we cannot test Selective Notification directly over applications with hundreds of thousands or millions of clients. Instead, we have built implementation-driven models of performance and measured the maximum, worst-case performance of a dispatcher. Assuming that all dispatchers have the same resources, we have modeled the worst-case performance of a dispatcher overlay network using the worst case performance of all dispatchers in the tree (assuming optimal network-level performance.) From this data, we have modeled the resources required to achieve necessary service properties.

## **6.2 Measurement of Dispatcher Performance**

We obtain dispatcher performance capabilities by operating and measuring a dispatcher in a controlled environment; one in which all the input factors affecting performance are controlled. These factors are:

- *Available Resources*: This includes computing hardware, network resource, and the run-time system environment.
- *Event Forwarding Set Size*: The size of the set of sub-dispatchers and clients to which events are forwarded.
- *Event Persistence Lifetime*: The persistence lifetime of event messages.
- *Subscription Change Rate*: The rate at which clients modify their state and the rate at which filter coagulation is performed.
- *Connection Policy Complexity*: The number of attribute constraints registered by clients.

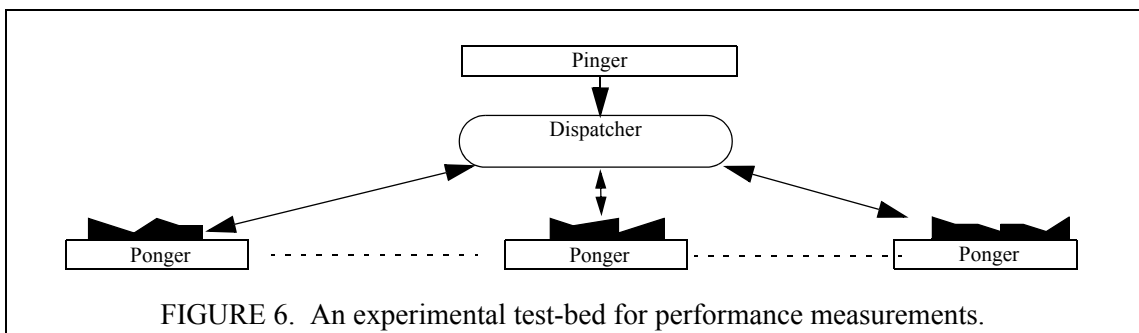
Our experimental apparatus, illustrated in Figure 6, is a “ping-pong” throughput experiment. A single “Pinger” application sends “ping” messages to “Ponger” applications in sequence. “Ponger” applications respond to the “Pinger” with a “pong” message. Additionally, “Ponger” applications

generate broadcast-like background messages sent to all other “Pongers.” During the entire experiment, “Ponger” applications randomly change the values of their client models (essentially their addresses) at a specified rate, representing diverse dynamic state change throughout the distributed system. These last two conditions are indicative of a worst-case system behavior with consistently changing client state and the pervasion of “broadcast” commands and alerts.

By varying the number of “Ponger” elements, the rate at which they “chatter”, the event-persistence lifetime of “chatter”, the size of the “Ponger” client model, and the rate at which “Ponger”s change their client model instances, we observe performance capability with respect to the input parameters of interest. For purposes of experimentation, each application—including each “Ponger”, the “Pinger”, and the Selective Notification dispatcher—were run on separate computers.

#### *Event Throughput and Output Performance*

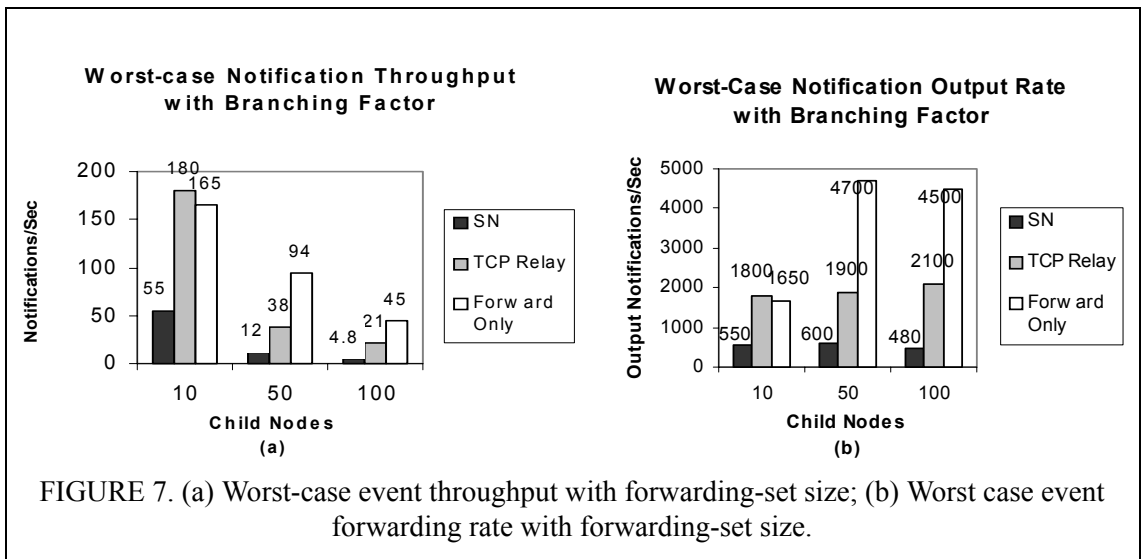
Figure 7(a) shows the maximum throughput rate computed using the “ping-pong” experimental configuration. Figure 7(b) shows the maximum event forwarding rate computed for the same experiments. The X-axes are the number of “Ponger” clients attached to a Selective Notification dispatcher. Maximum rates were calculated by varying “Ponger” broadcast-chatter rates and determining the point of throughput saturation, i.e., the point where the dispatcher would begin falling behind permanently. This experiment was performed with 10 second client-model attribute updates and coagulation updates, and 60 second persistence lifetimes for notifications. Ping notifications were sent every two seconds.



Three versions of the Selective Notification dispatcher were run in separate trials. The first was complete, the second blindly forwarded all notifications, and the third simulated infinitely-fast TCP communications. The results show that the throughput rate of Selective Notification under worst-case forwarding conditions (“broadcast”-like) is dominated by the cost of TCP communications, followed closely by the cost of forwarding computation. Figure 7(b) shows that these costs are governed by the number of output events that must be produced by the dispatcher. Note that the rate is constant for a given feature set regardless of branching factor. Thus, the primary determination of throughput is the size of the set of potential receivers. The cost is not constant at ten “Pong” nodes for simulated TCP because of domination by the cost of Java garbage collection.

*Effect of Client Model Size on Performance*

The size of client models, i.e., the size of the exposed address, and the resulting size of targeting notifications has a significant impact on system performance. A client-model’s size is measured by the number of attributes in its address. This is proportional to half the size of the default filters generated for intentional addressing. Figure 8(a) shows the output notification rate of Selective Notification with the size of subscription filters. This experiment was performed with 60 second message persistence, 10 second filter coagulation, and 50 “Ponger” applications. The second data point (5 model attributes) corresponds roughly to the size of the attribute models in the exper-



iment from which data was collected in Figure 7. Selective Notification is five times less efficient with 25-attribute client models than with 5-attribute client models.

### *Round-trip Message Time*

Our experiments recorded round-trip message time for “ping” and “pong” message pairs. Figure 9 shows “ping-pong” time over the course of an experiment in linear and logarithmic scale. The data is from an experiment with 50 “Pongers”, with attribute changes every ten seconds and notification persistence of 60 seconds. Ten messages were input to the dispatcher per second, so that the system was not processor-saturated. The experiment was run for 200 seconds. Average round-trip time was 220 milliseconds, with a standard deviation of 430 milliseconds. Deviation occurs from persistent notification time-outs, filter coagulation, clusters client model changes, and Java garbage collection. Under the worst case example of these conditions, round-trip time may be as high as three or four seconds.

### **6.3 A Model of Scale**

From these experiments, it is clear that the throughput of Selective Notification depends heavily on the number of clients and dispatchers connected to a dispatcher, and on the size of client models. Less important but still significant are the rates of client attribute (model) changes. Using the measurements from the previous section for a dispatcher operating in controlled conditions, we

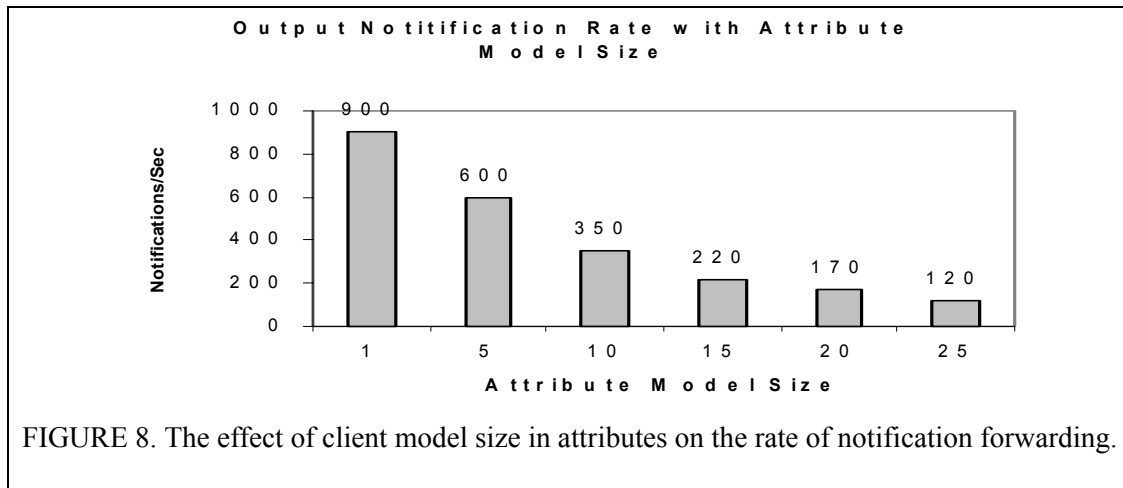


FIGURE 8. The effect of client model size in attributes on the rate of notification forwarding.

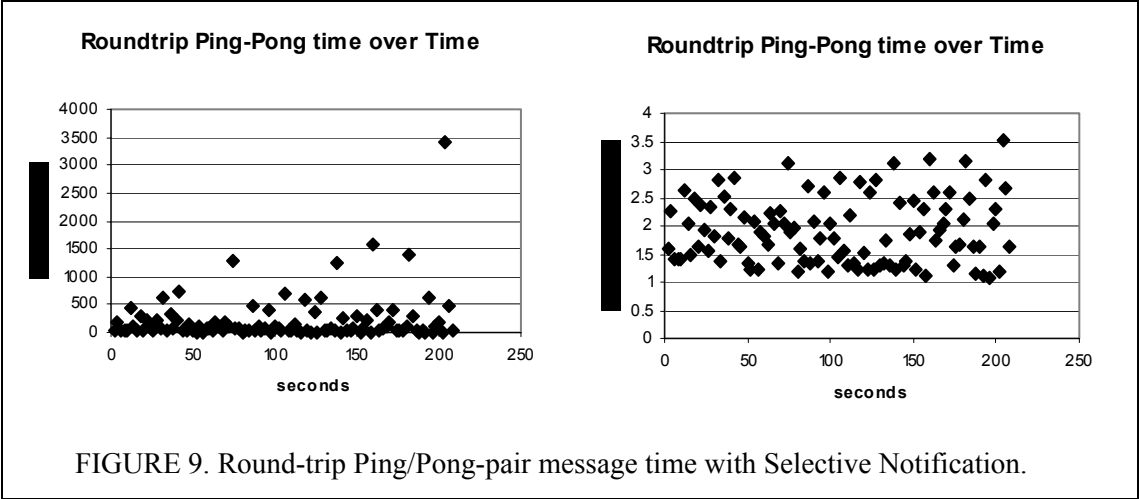


FIGURE 9. Round-trip Ping/Pong-pair message time with Selective Notification.

can estimate the maximum throughput potential of Selective Notification in large distributed applications.

A hierarchy of Selective Notification dispatchers is needed to reach large numbers of clients. Consider a system with a million clients. Figure 10 shows notification worst-case throughput and worst-case delivery time for variations in dispatcher-tree branching factor and resource dedication. We consider dedicated dispatcher networks (labelled with solid shapes) in which the dispatchers use all computational resources, and peer dispatcher networks (labelled with shape outlines) in which the dispatchers use one tenth of the resources while clients use the remainder.

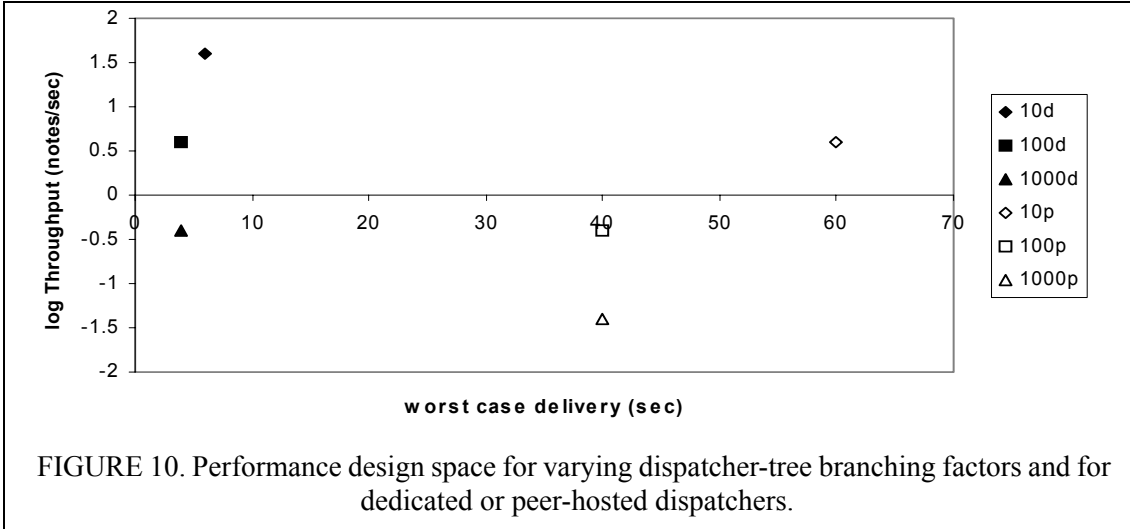


FIGURE 10. Performance design space for varying dispatcher-tree branching factors and for dedicated or peer-hosted dispatchers.

Using our current implementation, a dedicated dispatcher tree with a branching factor of one-thousand can support a notification every three seconds and deliver events in four seconds to a million elements. With a branching factor of ten, a peer dispatcher tree can support four notifications per second delivered in 60 seconds. Dedicated dispatchers at the higher levels of the tree and peer dispatchers at lower levels can provide intermediate results for both notification rate and throughput. 10,100 dedicated dispatchers at the base of the tree with a branching factor of 100 followed by peer dispatchers with a branching factor of ten to the clients would result in four notification per second throughput with 24 second delivery time.

## **7. Conclusions**

We have introduced a comprehensive and symmetric approach to communication between managing entities and managed entities that has immediate utility in dealing with security attacks and other traumas that require rapid reconfiguration of large networked information systems. Our experimental assessment of our implementation suggests that the approach scales and provides acceptable performance. It might be possible to cope with worm attacks in very large networks through rapid, targeted event dissemination as illustrated in Section 4.

While additional properties in management relationships are necessary to implement loosely coupled management, symmetric decoupled communication can serve as a backbone for potential architectures. These systems will allow flexible, dynamic, and run-time management relationships to reflect and change with system and manager state. As a result, traditional management structures, such as hierarchy, may be applicable to very large and dynamic systems. This includes application to interface with more widely distributed and cooperative forms of management.

## **8. Acknowledgements**

It is a pleasure to acknowledge many helpful discussions about this work and about the Siena software system with Antonio Carzaniga, Alex Wolf, and Dennis Heimbigner. This work was supported in part by the Defense Advanced Research Projects Agency under grant N66001-00-8945

(SPAWAR) and the Air Force Research Laboratory under grant F30602-01-1-0503. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, the Air Force, or the U.S. Government.

## References

- [1] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. "The design and implementation of an intentional naming system." *Operating Systems Review*, Vol. 34 No. 5, pp 186-2001. Dec. 1999.
- [2] A. Carzaniga, D. Rosenblum, A. Wolf. "Achieving scalability and expressiveness in an Internet-scale event notification service." *Symposium on Principles of Distributed Computing*, 2000. pp 219-227.
- [3] A. Carzaniga, A. Wolf. "Content-based Networking: A New Communication Infrastructure." *NSF Workshop on an Infrastructure for Mobile and Wireless Systems*. In conjunction with the *International Conference on Computer Communications and Networks ICCCN*. October, 2001.
- [4] G. Cugola, E. Di Nitto, A. Fuggetta. "The JEDI event-based infrastructure and its application to the development of the OPSS WFMS." *IEEE Transactions on Software Engineering*, Volume: 27 Issue: 9, pp 827 -850. Sept. 2001.
- [5] P. Eugster, P. Felber, R. Guerraoui, A. Kermarrec. "The Many Faces of Publish/Subscribe." *Microsoft Research Technical Report EPFL*, DSC ID:2000104. January 2001.
- [6] B. Gerkey, M. Mataric. "Murdoch." *Proceedings of the Fourth International Conference on Automated Agents*. June 2000
- [7] R.S. Hall, D. Heimburger, A.L. Wolf. "A Cooperative Approach to Support Software Deployment Using the Software Dock." *International Conference on Software Engineering*. 1999.
- [8] D. Heimburger. "Adapting publish/subscribe middleware to achieve Gnutella-like functionality." *Selected Areas in Cryptography*, 2001. pp 176-181.
- [9] C. Intanagonwiwat, R. Govindan, D. Estrin. "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks." *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking (MobiCOM '00)*. August 2000
- [10] J. Martin-Flatin, S. Znaty, J. Hubaux. "A Survey of Distributed Network and Systems Management Paradigms" *Journal of Network and Systems Management*. 7(1). pp 9-22. 1999
- [11] N. Skarmeeas, K.L. Clark. "Content based routing as the basis for intra-agent communication." *Proceedings of the 5th International Workshop on Intelligent Agents(V): Agent Theories, Languages, and Architectures*, 1998.
- [12] Vogels, W. Re, C. van Renesse, R. Birman, K. "A Collaborative Infrastructure for Scalable and Robust News Delivery." *Proceedings of the IEEE Workshop on Resource Sharing in Massively Distributed Systems (RESH'02)*. July 2002.
- [13] Vogels, W. Re, C. van Renesse, R. Birman, K. "Scalable Management and Data Mining Using Astrolab." *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS)*. Cambridge, Massachusetts. March 2002.