Authorization Using the Publish-Subscribe Model

Qiang Wei
University of British Columbia

Abstract

Traditional authorization mechanisms based on request-response paradigm lead to point-to-point (PTP) communication between applications and the authorization server. As distributed applications increase in size and complexity, the PTP-based authorization architecture is becoming fragile and inefficient. This paper introduces the use of a publish-subscribe (pub-sub) paradigm for delivering authorization requests and responses between the application and authorization server. We study the design both qualitatively and quantitatively using an existing pub-sub system. Our evaluation results show that the use of pub-sub can improve the availability of authorization systems.

1 Introduction

Modern access control solutions are based on the request-response paradigm [19, 13, 23, 28, 24, 10], as illustrated in Figure 1. In this paradigm, a policy enforcement point (PEP) intercepts application requests, obtains access control decisions (a.k.a. authorizations) from a policy decision point (PDP), and enforces those decisions.

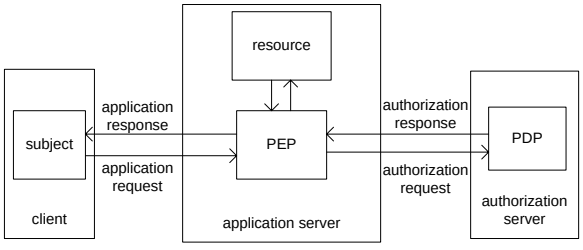


Figure 1. Request-response paradigm.

The separation into PEP and PDP in the request-response paradigm enables reusing PDPs in the form of authorization servers, thereby reducing administrative overhead, as there are fewer PDPs to administer than PEPs, as well as enforcing consistent policies across multiple PEPs.

On the other hand, the request-response paradigm commonly leads to point-to-point (PTP) request-responses, with PEPs obtaining decisions from PDPs through synchronous RPC. The PTP approach breaks down at massive-scale or is expensive to compute authorizations, resulting in fragile and inefficient solutions. Figure 2 illustrates the high coupling of solutions based on PTP architectures, which, as a result, become prone to high fragility.

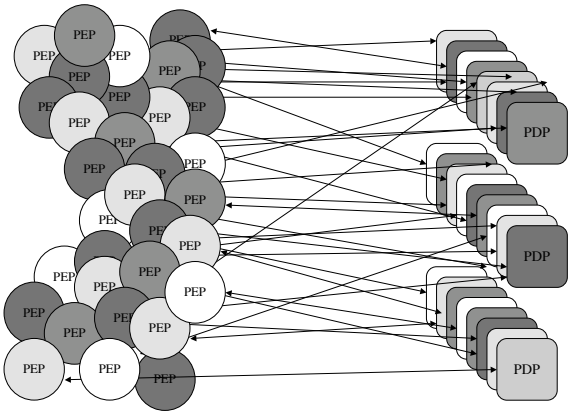


Figure 2. The high degree of coupling between PEPs and PDPs leads to fragile architectures

The frangibility leads to reduced availability: the authorization server may not be reachable due to a failure (transient, intermittent, or permanent) of the network, of the software located in the critical path (e.g., OS), of the hardware, or even from a misconfiguration of the supporting infrastructure. A conventional approach to improving the availability of a distributed infrastructure is failure masking through redundancy of either information, or time, or through physical redundancy [16]. However, redundancy and other general purpose fault-tolerance techniques for distributed systems scale poorly, and become technically and economically infeasible when the number of entities in the

system reaches thousands [17, 31].

First proposed in [3], this paper presents the use of a publish-subscribe (pub-sub) architecture to replace the existing PTP architecture. A pub-sub system is an asynchronous messaging paradigm. Different from PTP paradigms where senders of messages are programmed to send their messages to specific receivers, messages are sent without knowledge of who will receive them. Similarly, the subscribes show interests on some messages without knowing who publish these messages. By using pub-sub in the authorization systems, the dependency of PEP on a particular PDP is fully removed, thus improves the system availability and makes system administration efficient.

This paper makes the following two contributions. First, we present the design of using publish-subscribe model for authorization architecture. Second, we developed a prototype and evaluated it experimentally. Our preliminary evaluation results show that our approach can improve the availability of authorization infrastructures.

The rest of this paper is organized as follows. Section 2 presents background, including the access control architecture and publish-subscribe model. Section 3 describes the system design. Section 4 evaluates a prototype implementation. Section 5 discusses related work. We conclude our work in Section 6.

2 Background

This section provides background on request-response authorization paradigm and publish-subscribe system that are necessary for understanding the rest of the paper.

2.1 Request-Response Authorization Paradigm

In the request-response paradigm, the PEP intercepts application requests from the subject and enforces the decision from the PDP. A PEP can be a security interceptor (such as in CORBA Security [24], ASP.NET [21], and most Web servers), or can be a part of the component container (as in COM+ [12] and EJB [10]). A PEP can also be a part of the corresponding application resource, e.g., implemented via static or dynamic "weaving" using aspect oriented software development techniques [20]. The PDP is usually implemented in the form of authorization servers. It can be designed specifically the application or by the third party components. The PDP has the information of policy which is usually specified by the security administrator and stored in a policy store (which is not shown in Figure 1).

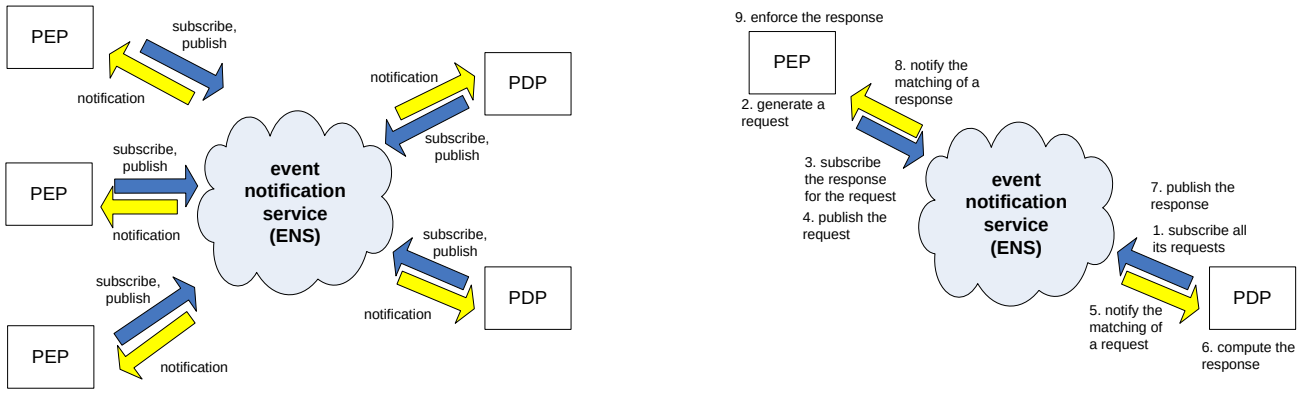
We distinguish between the *application request*, which is generated by the subject and is dependent on the application logic, and the *authorization request*, which is generated by the PEP and is independent from the application logic. This decoupling, for instance, is performed by the *context handler* in the XACML-compliant PEP [33]. The context handler generates an XACML *request context*, which is sent to the PDP for processing. We define the authorization request as a tuple (s, o, a), where s is the subject, o is the object, a is the access right and the authorization response as a tuple (r, d), where r is the request and d is the decision. Defining request and response gives us the context of discussion on subscription of requests and responses.

2.2 Publish-Subscribe Model

A publish-subscribe model is a common model for performing asynchronous communication in large-scale enterprise applications, providing the loosely coupled form of interaction between entities whose location and behaviors may vary throughout the lifetime of the system [14]. Generally, entities that wish to send messages "publish" them as events, while entities that wish to receive certain types of messages (or events) "subscribe" (or register) to those events. Often, an entity may become both a publisher and subscriber, sending and receiving messages within the system.

An event notification service (ENS) provides a logically centralized service that mediates the communication between publishers and consumers of information in distributed system. An ENS is responsible for receiving: (1) subscriptions, coming from event consumers, and (2) events, coming from their producers. With these two sets of information, it performs the matching of subscriptions with their corresponding subset of events, routing the resulting events, as notifications, to the interested parties. The ENS can be implemented by a single server, but also by a set of federated servers to reduce the single point of failure.

Event-based systems can be divided into two groups: subject-based systems and content-based systems. In *subject-based* systems, a message belongs to one of a fixed set of topics (a.k.a. groups and channels). Subscription targets a topic, and the user receives all events that are associated with that topic. *Content-based* systems, on the other hand, are not constrained to the notion that a message must belong to a particular topic. Instead, the decision of to whom a message is directed is made on a message-by-message basis based on a query or predicate issued by a subscriber. The advantage of a content-based system is its flexibility. It provides the subscriber just the information he/she needs. The subscriber need not have to learn a set of topic names and their content before subscribing. In this paper, we use the content-based scheme.



(a) System architecture.

(b) How it works

Figure 3. Publish-subscribe architecture for requesting and delivering authorizations

3 System Design

We propose to use a pub-sub architecture to replace the PTP communication between PEPs and PDPs in the existing authorization systems. The system architecture is illustrated in Figure 3(a). The system consists of PEPs, PDPs and a logically centralized event notification service (ENS). The PEP intercepts the application requests and generates the authorization requests. The PDP is responsible for making authorizations. All communication between PEPs and PDPs is mediated by the ENS, thus PEPs and PDPs are fully decoupled. The ENS can be implemented as a single server or a federation of servers, which depends on the scale of the system. Both PEPs and PDPs can either be subscribers or publishers. In particular, PEPs can publish requests and subscribe to responses, while PDPs can publish responses and subscribe to requests.

Figure 3(b) shows how the system works. The PDP first subscribes to all the requests it can resolve when it boots up (step 1). When the PDP intercepts an application request, it generates an authorization request (step 2). It first subscribes to the response for the request (step 3) and then immediately publishes the request (step 4). If some PDP can resolve the request, the ENS should find a match and notify the request to the PDP (step 5). After the PDP calculates a response (step 6), it immediately publishes the response (step 7). The ENS then notifies the response to the PEP (step 8) and the PEP will eventually enforce the decision in the response (step 9). Although multiple PDPs may resolve the same request and publish the response, it is not necessary for the PEP to wait for all of them. Once the PEP receives a response for a request, it will immediately unsubscribe that response and the ENS will remove the subscription accordingly to improve its own performance.

3.1 Qualitative Considerations

In this section, we provide a qualitative analysis on system availability, performance, security and administration.

- Availability. Using pub-sub, the PEP can receive authorizations even when its own PDP fails thus the availability is increased. With the PTP architecture, a request is only sent to one PDP. In contrast, with the pub-sub architecture, a request might reach multiple PDPs. This is due to the fact that in existing enterprise systems, same data often resides in multiple locations, on multiple machines and in a variety of systems. Consequently, multiple PDPs may have to be setup to resolve the access request for the same data. Therefore, in pub-sub, even though some of the PDPs could fail, the chances that at least one will provide a response on time will be higher[1]. In other words, our design enables the collection of PDPs to appear as a single large reliable PDP matrix.

- Performance. In pub-sub, the need to use an additional software component like ENS between PEPs and PDPs imposes a run-time overhead, which in turn degrades the application performance. In particular, the ENS needs to spend additional time on finding interested subscribers for each message and routing the message to them. However, in the case that this time is far less than the network latency or the computation cost at the PDP, its impact on the overall performance can be neglected.

[1]Note that we assume that different PDPs in the enterprise system enforce consistent policy, so that the PEP can use the decision made by any PDP.

- Security. Using pub-sub introduces new threats to the system. For example, malicious parties can listen to the channel and publish false information. The system design should address these new requirements.

- Administration. We expect the replacement of the coupling between PEPs and PDPs with the couplings between PDPs and publish-subscribe channels as well as between PDPs and those channels could reduce the human resources required to operate and administer authorization infrastructures. Consider a failed PDP, for instance. It still needs to be brought back up and possibly relocated. However, only the configuration of the publish-subscribe infrastructure has to be re-adjusted and not all the PEPs that depend on the failed PDP.

3.2 Using Siena as the ENS

In this paper, we used Siena [6]—an content-based ENS developed using JAVA in the University of Colorado—as an example to study our proposed architecture. In Siena, the publisher publishes the event by specifying a set of attribute and value pairs. The attribute names are simply character strings, and the value can be a predefined set of primitive types commonly found in programming languages, and for which a fixed set of operators is defined. The following sample code shows how to publish an event in Siena.

```
Notification e = new Notification();
e.putAttribute("name", "Antonio"); // name = "Antonio"
e.putAttribute("age", 30); // age = 30
e.putAttribute("nationality", "Italian"); // nationality = Italian
siena.publish(e);
```

The subscriber subscribes to the event by specifying filters using a subscription language. The filters define constraints, usually in the form of name-value pairs of attributes and basic comparison operators ($=, <, \leq, >, \geq$), which identify valid events. The following sample code shows how to subscribe to an event in Siena.

```
Filter f = new Filter();
f.addConstraint("name", "Antonio"); // name = "Antonio"
f.addConstraint("age", Op.GT, 18);  // age > 18
siena.subscribe(f);
```

Matching in Siena is accomplished at the server with a Binary Decision Diagram. Siena's routing paths for notifications are set at time of subscription. In a distributed Siena deployment, a new subscription is stored and forwarded from the originating server to all servers in the network. This forms a tree that connects subscriber with servers. Notifications are then routed towards the subscriber following the reverse path of the tree.

3.3 Subscription Mechanism

By subscription, an entity shows its interests on some events and receives the notification accordingly. The ENS usually provides a $subscribe()$ operation with an additional argument representing a subscription pattern. There are several ways to represent this kind of pattern, including string, template objects, and executable code. Our subscription is based on the string operation as used in Siena, which is also the most widely implemented in existing pub-sub systems. Since the ENS is a two-way communication channel in our design, below we describe the subscription for the PDPs and PEPs separately.

3.3.1 PDP Subscription

By PDP subscription, the PDP shows its interest on the requests that it can resolve. A naive approach is for the PDP to subscribe to any request published by any PEP, even though it can not resolve all of them. Similar to the request broadcasting, this approach is simple yet inefficient in that it causes unnecessary overheads on each PDP.

Ideally, a PDP needs to precisely subscribe to only those requests that it can resolve. To do it, the most intuitive approach is for the PDP to subscribe to all the request tuples (s,o,a) that it is responsible for. This prevents an PDP from resolving requests that it cannot resolve. However, this approach may lead to a large number of subscriptions registered in the ENS when the resource maintained by each PDP is large and when the number of PDPs is large. Previous researches on matching algorithms (e.g., [1]) have suggested that the large number of subscriptions post a challenge on the matching performance of the ENS. Usually algorithm used to find a matching subscription for an event has linear and sub-linear time complexity with respect to the number of subscriptions.

Therefore, one important task in our design is to reduce the number of subscriptions. In the following we discuss three non-exclusion subscription mechanisms to achieve it. We call these approaches as *approximate subscription*, where the number of subscription is reduced at the cost of the possibly increased overhead at each PDP.

The first approach is object-based subscription. In stead of subscribing each possible request, the PDP only subscribes each possible object for which it is responsible. For example, a subscription message can be "$object = file1$" and another subscription message can be "$object = file2$". This approach leads to reduced subscription by reducing the subscription dimensions from three to one. However, the approach may incur unnecessary overhead if some subjects are not enforced the access within that PDP.

The second approach is based on our further assumption on the object namespace. It is well-known that the object namespace in most distributed systems follows a hierarchical structure. This enables a PDP to show interest to a number of objects by subscribing to their parent directory. We use the IBM Tivoli policy director [18] as an example,

where the protected object namespace is a hierarchical portrayal of resources that belong to a secure domain. Its elements are strings whose syntax and structure are similar to absolute URIs but without the scheme, machine, and query components. The slash character ('/') is used to delimit, from left to right, hierarchical substrings of the object's name. For example, '/cgi-bin/test-cgi.exe' and '/sales/budget/quarter1/New%20York/travel' are both examples of object names. In this case, each PDP can subscribe to only the directory string instead of subscribing each leaf string. We expect that this will further significantly reduce the number of subscriptions.

The third approach is called domain-based or application-based subscription, which is based on the assumption that each PDP is responsible for several applications. Therefore, the PDP may subscribe only those requests that come from the applications for which it is responsible. However, this solution unavoidably depends on the structure of enterprise systems, and the authorization request needs to introduce the extra attributes to indicate the domain or applications.

3.3.2 PEP Subscription

By PEP subscription, a PEP subscribes to the responses published by the PDP. This can be achieved by passing the request as the parameter of Siena's $subscribe()$ operation.

The PEP can also benefit from approximate subscription. For example, the subject of the request can be represented as a set of role attributes, as shown in [32], and the PDP makes decision based on these attributed. If two subjects have the similar attributes, their access requests to the same object will receive the same decision. Based on this observation, the PEP can subscribe to the attributes of a request. The decision for other subject but with similar attributes can now be reused.

3.4 Security

In this section, we discuss the security aspect of the pub-sub architecture. We focus on two attacks to the system. Note that we assume that the ENS is always trusted by both the PEPs and PDPs.

The first attack is the performance attack, which happens when a malicious party can subscribe to or publish any event through the ENS. A malicious publisher can send a large number of junk events to the ENS thus increases its load. Likewise, a malicious subscriber can register a large number of junk subscriptions in the ENS thus increases its difficulty in finding an interested party. To prevent this attack, the ENS needs to provide an authentication mechanism to allow only an authenticated party to subscribe or publish to the ENS. However, most content-based systems including Siena allow anonymous subscribers, implying each message instead of the sender should be authenticated before it is processed by the ENS.

The second attack is the correctness attack, which happens when a malicious party can subscribe to any ongoing request and publish false response to the request. If these false responses are used by the PEP, the system correctness is broken. To prevent this attack, one approach is for the ENS to prevent the the publish of malicious messages in the first place. If this is not available, the PEP needs to verify each returned response to make sure that it comes from a trusted PDP. Public Key Infrastructure (PKI) can be used for this, which requires that each PEP is aware of all the trusted PDPs in the system. In addition, the overhead of authenticating every message is probably high.

3.5 Integration with SAAM

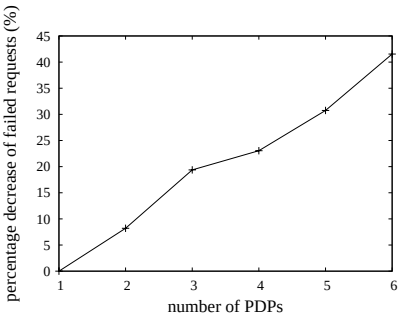
Our analysis shows that the use of ENS impacts the system performance. To reduce this impact, the pub-sub system can be integrated with the Secondary and Approximate Authorization Model (SAAM) [7]. SAAM adds a secondary decision point (SDP) to the request-response paradigm. The SDP is collocated with the PEP and can resolve authorization requests not only by precise recycling but also by computing *approximate authorizations* from cached authorizations. Since SDP can resolve a number of request, less requests will be processed by the ENS, thus reduces the impact of the ENS overhead.

A challenge of using SDP lies in maintaining cache consistency. Using pub-sub provides an efficient way for this: the SDP can register the policy change messages the published by the PDP. After removing the affected cache, the SDP can publish a message indicating whether or not the update is successful.

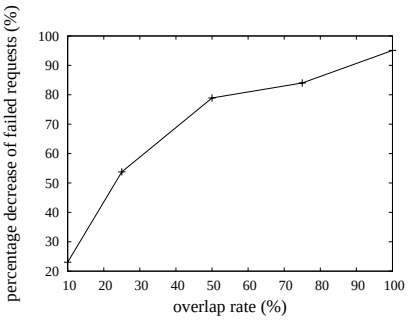
4 Experimental Evaluation

This section presents the experimental evaluation on our design and some preliminary results. Our evaluation focused on the availability gain by using the pub-sub model. We also studied the system performance in terms of response time.

The experimental system consisted of one PEP, one or more PDPs, and a centralized ENS which used the Siena implementation. All of them ran as separate process on the same machine with an AMD Athlon Dual Core processor 3800+ 2.00 GHz and 3GB of RAM, running Windows XP. The evaluation framework ran on Sun's 1.5.0 Java Runtime Environment (JRE).



(a) percentage decrease of failed requests as a function of number of PDPs



(b) percentage decrease of failed requests as a function of overlap rate

Figure 4. Availability results

4.1 Availability

We used the percentage decrease of failed requests as a metric to measure the availability gain. In each experiment, we first measured the failed requests using the PTP model where the PEP only sent its requests to its own PDP, then repeated the experiment with the pub-sub system and measured the failed requests again, and finally calculated the percentage difference. When this number increases, it means that the availability also increases since more requests has been successfully resolved. Our previous analysis shows that using pub-sub might lead to increased availability as a request can reach multiple PDPs. Here we want to understand by how much.

Before the start of each experiment, a RBAC policy was generated for each PDP. We controlled the object space of each PDP to simulate the overlap rate between PDPs. In the beginning of the experiment, each PDP subscribed to the objects for which it is responsible when it booted up. The PEP then started sending 5,000 randomly generated requests every 60ms. For each generated request, the PEP first subscribed to the response for this request and then published this request to Siena. After the PDP received a notification from the ENS, the PDP generated authorization responses in the format of the request and the decision. After receiving the first response for that request, the PEP unsubscribed the response for that request.

In the experiment, each PDP switched between two modes: working mode and failing mode. In the working mode, the PDP published a response after computing a response, while in the failing mode, the PDP would simply ignore the request notification from Siena and did nothing. We used the number of elapsed requests as a measurement of time. We ensured that all the PDPs received all requests so that their time clock was synchronized. We used time-to-failure (TTF) to represent the the time between two consequent failures and time-to-repair (TTR) to represent the time a failure lasted. We used the exponential distribution to simulate both time. The mean TTF (MTTF) is 100 requests and the mean TTR (MTTR) is 10 requests.

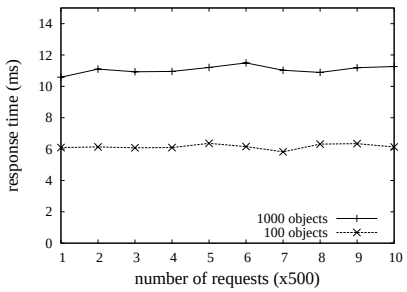
In the first experiment, we varied the number of PDPs while kept the overlap rate between PDPs as 10%. Figure 4(a) shows the results. As we expected, the availability increases with the number of PDPs. In the second experiment, we varied the overlap rate as 10%, 25%, 50%, 75% and 100% while fixed the the number of SDPs as 4. Figure 4(b) shows that the availability increases with the overlap rate. The reason for both results is that the possibility that a request can be resolved by other PDPs is increased.

4.2 Performance

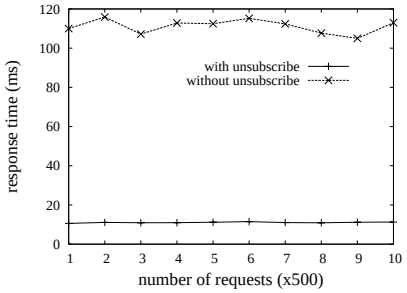
This section presents the performance results in terms of response time. The response time is measured as the period after the PEP generates a requests until it receives a response. The main operations during this period include response subscription and request publishing by the PEP, ENS finding and notifying an interested PDP, response computing and publishing by the PDP, and the ENS finding and notifying an interested PEP.

In the experiments, we did not simulate the failure of PDPs. We studied the impact of following factors on the response time: 1) the number of subscriptions in the ENS, which can be posted by the PDP or PEP; 2) the use of approximate recycling; and 3)system scale, i.e., the number of PDPs in the system.

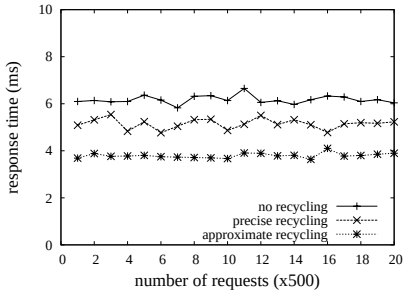
The main overhead of ENS is caused by finding an interested subscriber when receiving a request or response. If the number of subscription increases, the time used to find a match also increases. We ran two experiments to show how the number of subscription affects the response time. In the first experiment, the PDP either subscribed 100 or 1,000



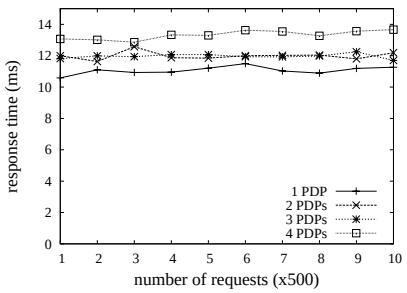
(a) As the subscriptions posted by the PDP vary



(b) As the subscriptions posted by the PEP vary



(c) Integration with SAAM



(d) As the number of PDPs changes

Figure 5. Response time as the function of the number of requests

objects. Figure 5(a) shows that the response time almost doubled with a larger number of subscriptions. In the second experiment, the PEP either unsubscribes those received response or not. Figure 5(b) shows that using unsubscribe method helps to reduce the response time significantly.

Next, we sought to understand how integration with SAAM can impact the response time. In the experiment, the PEP first sent its request to the SDP. If the SDP could not resolve the request, the PEP then published the request to Siena. The PEP also used the returned responses to build the cache. In the experiment, we used 10,000 requests, which is one third of the total requests. Figure 5(c) shows that SAAM helps to reduce the response time as we expected. This is because many requests now can be resolved by the collocated SDP.

Finally, we wanted to understand how the system scale, i.e., the number of PDPs in the system, will affect the response time. Figure 5(d) shows that the response time increases with the number of PDPs. The reason is that adding more PDPs increases the number of subscriptions registered in Siena. Since the PEP always used the first received response, the impact is small.

5 Related Work

Pub-sub systems have been an active research area. The earlier publish/subscribe scheme was based on the notion of topics or subjects, and has been implemented by many industrial strength solutions (e.g., [30]). The content-based publish/subscribe variant improves on topics by introducing a subscription scheme based on the actual content of the considered events (Gryphon [2], Siena [5], Elvin [29], Jedi [8] and Java Message Service [22].)

Pub-sub systems have been used to support different type of applications, such as Internet games [4], mobile agents [25], user and software monitoring [15], mobile systems [9], groupware [11], collaborative software engineering [27], and WWW updates [26], among others. Our contribution of this paper lies in applying pub-sub models to the area of access control.

6 Conclusion

As distributed systems scale up and become increasingly complex, their access control infrastructures face new challenges. Conventional request-response authorization architectures become fragile and scale poorly to massive scale.

In this paper, we have presented a publish-subscribe architecture to improve the availability of access control system. With publish-subscribe, a request for an authorization could be delivered to many PDPs. Even though some of them could fail, the chances that at least one will provide a response on time will be higher. We introduce the system architecture and study the issues including subscription mechanisms, security and integration with SAAM. Our evaluation results demonstrate that our design achieved better availability while the performance is reduced.

For future work, we consider to integrate the security features into our implementation. Second, we want to run more comprehensive experiments. We need to run the experiments on multiple machines and find the potential bottleneck of the system. We might also need to consider an alternative design which would lead to reduced response time. For example, each response can be returned to the PEP directly instead of through the ENS.

References

[1] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *PODC '99: Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*, pages 53–61, 1999.

[2] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. Strom, and D. Sturman. Efficient multicast protocol for content-based publish-subscribe systems. In *Proceedings of the 19th International Conference on Distributed Computing Systems (ICDCS'99).*, pages 262–272, 1999.

[3] K. Beznosov. Flooding and recycling authorizations. In *Proceedings of the New Security Paradigms Workshop (NSPW)*, pages 67–72, Lake Arrowhead, CA, USA, 20-23 September 2005.

[4] A. R. Bharambe, S. Rao, and S. Seshan. Mercury: a scalable publish-subscribe system for internet games. In *NetGames '02: Proceedings of the 1st workshop on Network and system support for games*, pages 3–9, 2002.

[5] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *PODC'00: Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, pages 219–227, 2000.

[6] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Trans. Comput. Syst.*, 19(3):332–383, 2001.

[7] J. Crampton, W. Leung, and K. Beznosov. Secondary and approximate authorizations model and its application to Bell-LaPadula policies. In *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 111–120, Lake Tahoe, CA, USA, June 7–9 2006.

[8] G. Cugola, E. Di Nitto, and A. Fuggetta. The jedi event-based infrastructure and its application to the development of the opss wfms. *Software Engineering, IEEE Transactions on*, 27(9):827–850, Sep 2001.

[9] G. Cugola and H.-A. Jacobsen. Using publish/subscribe middleware for mobile systems. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(4):25–33, 2002.

[10] L. G. DeMichiel, L. Ü. Yalçinalp, and S. Krishnan. *Enterprise JavaBeans Specification, Version 2.0*. Sun Microsystems, 2001.

[11] P. Dourish and S. Bly. Portholes: supporting awareness in a distributed work group. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 541–547, 1992.

[12] G. Eddon. The COM+ security model gets you out of the security programming business. *Microsoft Systems Journal*, 1999(11), 1999.

[13] Entrust. GetAccess design and administration guide. Technical report, Entrust, September 20 1999.

[14] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[15] D. M. Hilbert and D. F. Redmiles. An approach to large-scale collection of application usage data over the internet. In *ICSE '98: Proceedings of the 20th international conference on Software engineering*, pages 136–145, 1998.

[16] B. Johnson. *Fault-tolerant computer system design*, chapter An introduction to the design and analysis of fault-tolerant systems, pages 1–87. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[17] Z. Kalbarczyk, R. K. Lyer, and L. Wang. Application fault tolerance with Armor middleware. *IEEE Internet Computing*, 9(2):28–38, 2005.

[18] G. Karjoth. The authorization service of tivoli policy director. In *Annual Computer Security Applications Conference (ACSAC)*, pages 319–328, New Orleans, Louisiana, 2001. IEEE.

[19] G. Karjoth. Access control with IBM Tivoli Access Manager. *ACM Transactions on Information and Systems Security*, 6(2):232–57, 2003.

[20] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of the 1997 11th European Conference on Object-Oriented Programming, ECOOP, Jun 9-13 1997*, volume 1241 of *Lecture Notes in Computer Science*, page 220, Jyvaskyla, Finl, 1997.

[21] J. Meier, A. Mackman, M. Dunner, and S. Vasireddy. *Building Secure ASP.NET Applications: Authentication, Authorization, and Secure Communication*, 2002.

[22] S. Microsystems. Java Message Service Specification, 2001.

[23] Netegrity. Siteminder concepts guide. Technical report, Netegrity, 2000.

[24] OMG. Common object services specification, security service specification v1.8, 2002.

[25] A. Padovitz, S. Loke, and A. Zaslavsky. Using the publish-subscribe communication genre for mobile agents. In *Proceedings of the First German Conference on Multiagent System Technologies*, Sep 2003.

[26] V. Ramasubramanian, R. Peterson, and E. G. Sirer. Corona: a high performance publish-subscribe system for the world wide web. In *NSDI'06: Proceedings of the 3rd conference on 3rd Symposium on Networked Systems Design & Implementation*, pages 2–2, 2006.

[27] A. Sarma, Z. Noroozi, and A. van der Hoek. Palantír: raising awareness among configuration management workspaces. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 444–454, 2003.

[28] Securant. Unified access management: A model for integrated web security. Technical report, Securant Technologies, June 25 1999.

[29] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps. Content Based Routing with Elvin4. In *Proceedings of AUUG2K*, June 2000.

[30] I. TIBCO. TIB/Rendezvous White Paper. *Palo Alto, California*, 1999.

[31] W. Vogels. How wrong can you be? Getting lost on the road to massive scalability. In *the 5th International Middleware Conference*, Toronto, Canada, October 20 2004. Keynote address.

[32] Q. Wei, W. Leung, J. Crampton, and K. Beznosov. Authorization recycling in bell-la padula systems. working paper, to submit to *ACM Transactions on Information and System Security (TISSEC)*, 2008.

[33] XACML-TC. OASIS eXtensible Access Control Markup Language (XACML) version 2.0. OASIS Standard, 1 February 2005.