Authorization Using the Publish-Subscribe Model

Qiang Wei, Matei Ripeanu, and Konstantin Beznosov
Dept. of Electrical and Computer Engineering, University of British Columbia
Vancouver, BC, Canada
{qiangw, matei, beznosov}@ece.ubc.ca

Abstract

Traditional authorization mechanisms based on the request-response model are generally supported by point-to-point communication between applications and authorization servers. As distributed applications increase in size and complexity, an authorization architecture based on point-to-point communication becomes fragile and difficult to manage. This paper presents the use of the publish-subscribe (pub-sub) model for delivering authorization requests and responses between the applications and the authorization servers. Our analysis suggests that using the pub-sub architecture improves authorization system availability and reduces system administration overhead. We evaluate our design using a prototype implementation, which confirms the improvement in availability. Although the response time is also increased, this impact can be reduced by bypassing the pub-sub channel when returning authorizations or by caching coupled with local inference of authorization decisions based on previously cached authorizations.

1 Introduction

Modern access control solutions are based on the request-response model [15, 11, 18, 21, 19, 10], as illustrated in Figure 1. In this model, 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. The separation into PEP and PDP in the request-response model enables using PDPs in the form of authorization servers, thereby reusing the authorization logic and enforcing consistent policies across multiple PEPs.

The request-response model, however, commonly leads to a point-to-point communication architecture, where PEPs obtain decisions from PDPs through synchronous remote procedure calls (RPC). This point-to-point architecture, in turn, results in tight coupling between PEPs and PDPs. At the same time, large-scale commodity computing is becoming a reality, with eBay having over 12,000 servers and 15,000 application server instances [23], and Google estimated to have "more than 450,000 servers spread in at least 25 locations around the world" [17]. When a distributed system evolves to this scale, a tightly-coupled point-to-point architecture suffers from two drawbacks: fragility and high management overhead.

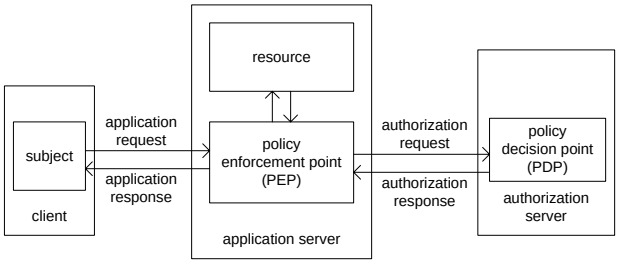


Figure 1. The request-response model.

Fragility occurs in a large-scale, point-to-point architecture when authorization service is unavailable. In that case, all applications depending on that authorization server may not work properly, if at all. The authorization server may be unavailable 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 due to a misconfiguration of the supporting infrastructure. A conventional approach to improving the availability of a distributed infrastructure is failure masking through redundancy (either information, time, or hardware). 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 [14, 25].

Additionally, management of large-scale authorization systems with a point-to-point architecture becomes increasingly costly. Due to the coupling between PEPs and PDPs, even relatively simple changes made on a PDP might be difficult to implement. Implementation may involve several

different application development teams sitting down to coordinate and make sure that the necessary changes do not break anything else. For instance, a failed PDP needs to be brought back up and possibly relocated. Each PEP depending on that PDP needs to update its information to reflect the PDP's relocation.

This paper presents an authorization system that uses a publish-subscribe (pub-sub) architecture to replace the existing point-to-point architecture. Our goals are to reduce system administration overhead, thus improving system manageability, and to increase the system availability, thus improving overall system robustness. Unlike in a point-to-point architecture, where PEPs are configured to send their requests to specific PDPs, a pub-sub architecture enables PEPs to send their requests without knowing which PDP will receive them. Similarly, the PDPs show interest in requests without knowing which PEPs generate them. By using the pub-sub architecture, the coupling between PEPs and PDPs is removed; as a result, system availability is improved and system administration is simplified.

This paper makes the following contributions. First, we present the design of an authorization system based on the pub-sub architecture and examine the requirements for the supporting event notification service (ENS). Second, we develop a prototype using an open-source ENS and evaluate it experimentally. Our evaluation suggests that the pub-sub architecture improves the availability of the authorization infrastructure. Although the response time is increased, this overhead can be reduced by returning authorizations bypassing the ENS, or by caching coupled with local inference of authorization decisions based on cached authorizations [8].

The rest of this paper is organized as follows. Section 2 presents background information on access control architectures and the pub-sub model. Section 3 describes our system design and analysis. Section 4 evaluates our prototype implementation. Section 5 discusses related work. We conclude in Section 6.

2 Background

This section provides background on the request-response authorization model and on the pub-sub model.

2.1 The Request-Response Model

In the request-response model, PEP intercepts application requests from subjects and enforces decisions from the PDP. Subjects are the processes that make application requests on behalf of users. A PEP can be a security interceptor (as in CORBA Security, ASP.NET, and most Web servers), or can be a part of the component container (as in COM+ and EJB). 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 [16]. The PDP is usually implemented in the form of authorization servers. It can be designed specifically for the application or use third party components. The PDP stores policy information which is usually specified by the security administrator.

We distinguish between an *application request*, which is generated by the subject and is dependent on the application logic, and an *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 [28]. 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.

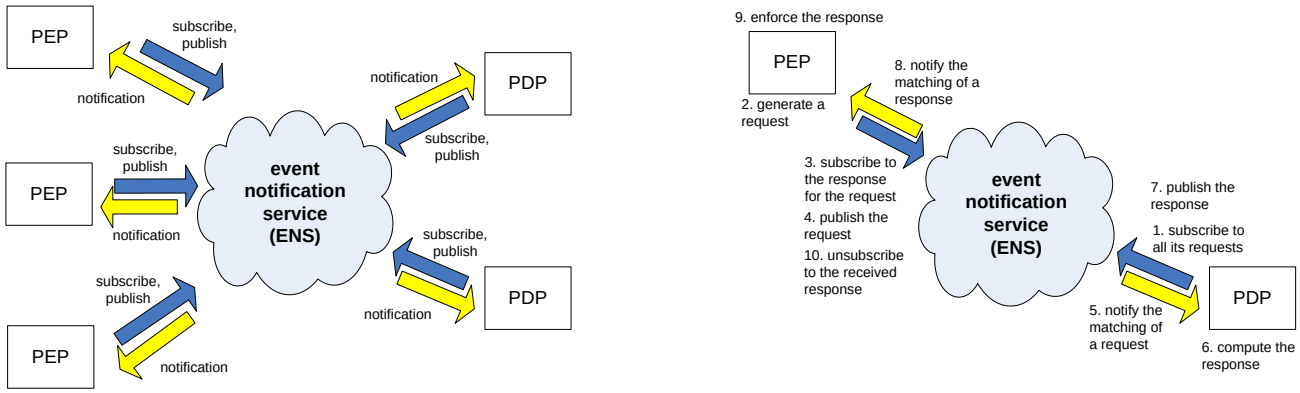
2.2 The Publish-Subscribe Model

A pub-sub model is a common communication model in large-scale enterprise applications, enabling loosely coupled interaction between entities whose location and behaviors may vary throughout the lifetime of the system [12]. Generally, entities that send messages "publish" them as events, while entities that wish to receive certain events "subscribe" to those events. Often, an entity may become both a publisher and subscriber, sending and receiving messages within the system.

Pub-sub systems usually provide one of two types of subscription schemes: topic-based subscription and content-based subscription. In a *topic-based* scheme, a message belongs to one of a fixed set of topics. A subscription targets a topic, and the subscriber receives all events that are associated with that topic. A *content-based* scheme is not constrained to the notion that each message must belong to a particular topic. Instead, the message delivery decisions are based on a query or predicate issued by a subscriber. The advantage of a content-based scheme is its flexibility: it provides the subscriber with the ability to specify just the information he/she needs without having to learn a set of topic names and their content before subscribing.

3 System Design and Analysis

This section presents the design and analysis of an authorization system based on a pub-sub architecture. This replaces the point-to-point communication used between PEPs and PDPs in existing authorization systems.



(a) System architecture.

(b) Typical data flow.

Figure 2. Publish-subscribe architecture for delivering authorization requests and responses.

3.1 System Architecture and Data Flow

The architecture of the pub-sub authorization system is illustrated in Figure 2(a). The system consists of multiple PEPs, PDPs and a logically centralized event notification service (ENS). The ENS mediates the communications between PEPs and PDPs, thereby fully decoupling PEPs and PDPs. The ENS is responsible for: (1) receiving events from publishers, (2) receiving subscriptions from event subscribers, and (3) matching each event to subscriptions and routing the event, in the form of notifications, to the interested subscribers.

PEPs and PDPs can be both subscribers and publishers. A PEP subscribes to the responses published by the PDP. This can be achieved simply by passing the request as the parameter of the $subscribe()$ operation of the ENS for each incoming request. A PDP subscription shows the ability of the PDP to resolve certain requests. A PDP can precisely subscribe to all the request tuples (s, o, a) it can resolve, or subscribe to those requests that contain the objects for which it is responsible, in order to reduce the number of subscriptions it creates.

Figure 2(b) shows a typical data flow. Each PDP first subscribes to all the requests it can resolve when it boots (step 1). After a PEP intercepts an application request, it generates an authorization request (step 2). It first subscribes to the response to the request (step 3) and then immediately publishes the request (step 4). If some PDP can resolve the request, the ENS will find a match and notify that PDP of the request (step 5). After the PDP computes a response (step 6), it immediately publishes the response (step 7). The ENS then notifies the PEP of the response (step 8) and the PEP finally enforces the authorization decision (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. Therefore, once the PEP receives a response to a request, it immediately unsubscribes to the response for that request (step 10).

Note that we assume that policies are consistent across PDPs, so that each PEP can enforce the decision made by any PDP in the system. When they are not consistent (due to policy changes), our solution does not make the system worse than the point-to-point system. To handle the inconsistency issue, one possible solution is for each response to include a time-stamp which represents the time that the PDP receives a policy change. When the PEP receives inconsistent responses, it always uses the one with the most recent time-stamp.

3.2 Benefits

Using the pub-sub model to build authorization systems is expected to provide the following benefits:

- **Increased availability**. In the point-to-point architecture, a PEP is generally configured to send a request to only one PDP. In existing enterprise systems, however, the same data often resides in multiple locations, on multiple machines, and within a variety of applications. Consequently, multiple PDPs may have to be setup to resolve access requests for the same data. Our solution exploits this situation: using the pub-sub architecture enables a request to reach all PDPs that are able to resolve it. Even though some of the PDPs may fail, the chances that at least one will provide a response on time are higher. In other words, our design allows the collection of PDPs to appear as a single large reliable PDP matrix.
- **Reduced administration overhead**. We expect that decoupling PEPs and PDPs will reduce the human

costs of operating and administering authorization infrastructures. Consider the previous example of a failed PDP. After it is brought back and possibly relocated, only the ENS might need to be reconfigured rather than the case of all the PEPs depending on the failed PDP as in a system based on point-to-point communication.

- **Improved software development process**. Using an event-driven, standards-based pub-sub channel to provide a comprehensive communication framework between PEPs and PDPs will improve the software development cycle. The integration of PEPs and PDPs is faster and less expensive using a pub-sub channel than using a point-to-point architecture.

3.3 Requirements for the ENS

The ENS is the core component of our architecture. For the pub-sub system to provide the aforementioned benefits, the ENS should meet the following requirements:

- **Robustness.** The ENS itself should be robust to achieve the benefits of improving the overall system availability. Otherwise, it becomes the single point of failure. Therefore, the ENS should only be *logically* centralized, but have a robust implementation.

  Exiting pub-sub systems usually provide robust ENS implementations, where the ENS is implemented not only by a single server but also by a set of federated servers to reduce the single point of failure problem. For instance, in Siena [7] and Elvin [22], two wide-area content-based pub-sub networks, the underlying pub-sub infrastructure can be implemented as a collection of network servers communicating with each other in a peer-to-peer fashion. The failure of a server will not impact the system performance significantly.

- **Performance.** In the pub-sub architecture, the insertion of an additional software component (the ENS) between PEPs and PDPs, imposes run-time overhead, which may in turn degrade application performance. In particular, the ENS needs to spend additional time to find the potential subscribers for each message and to route the message to them. Therefore, the ENS should be efficient in finding a match even with a large number of subscriptions. Otherwise, it may become the performance bottleneck. When the time used by the ENS is far lower than the network latency or the computational cost at the PDP, its impact on the overall performance is negligible. Additionally, Section 3.4 presents two performance-oriented optimizations that reduce ENS overhead and improve performance.

- **Security.** Using a pub-sub architecture introduces new threats to the system. We focus on two attacks. First, *performance attacks*: a malicious participant can attempt to lower the system performance. For example, a malicious publisher can send a large number of junk events to the ENS to increase its load, and a malicious subscriber can register a large number of junk subscriptions in the ENS to increase its time in finding an interested subscriber. Second, *correctness attacks*: a malicious participant can subscribe to any ongoing request and publish false responses. If these false responses are used by the PEP, the system's correctness is compromised.

  To prevent these attacks, the ENS needs to provide an authentication and authorization mechanism to allow only authorized parties to subscribe or publish. Most existing commercial pub-sub systems provide such a mechanism. If the ENS does not have this feature, the PEP will need to verify each response to ensure its correctness. Detailed analysis of the security model and solutions will be the subject of our future work.

3.4 Performance-Oriented Optimizations

The ENS introduces additional overhead to process subscriptions and deliver events. This section proposes two approaches to reduce the impact of ENS overhead on the overall system performance.

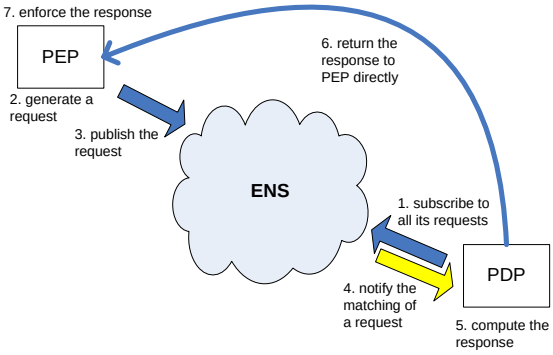
3.4.1 Bypassing the ENS when returning responses

The first technique requires a modification to the data flow. Instead of returning responses through the ENS, the PDP returns them directly to the PEP, as shown in Figure 3(a). To preserve the decoupling property of the pub-sub architecture, each request contains the address of the PEP that publishes it. Each PDP then can make a remote call to return the response without knowing the PEP address in advance.

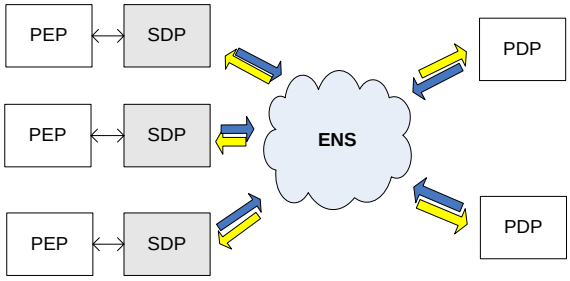
We expect that returning responses without the ENS can improve the system performance for the following two reasons: 1) the number of calls involved in resolving a request is reduced; and 2) the load of the ENS is also reduced. Our evaluation results in Section 4.4 confirm that this approach improves the performance. Note that this approach prevents cooperative caching at the ENS, in which the ENS caches responses made by PDPs and uses them to resolve the requests from PEPs.

3.4.2 Integration with authorization recycling

Another technique for improving system performance is by integrating the pub-sub authorization system with the Secondary and Approximate Authorization Model



(a) Technique I: Changed data flow to improve performance: the PDP returns responses to the PEP directly, bypassing the ENS.



(b) Technique II: A secondary decision point (SDP) is added to the pub-sub system.

Figure 3. Performance-oriented optimization techniques.

(SAAM) [8]. SAAM adds a secondary decision point (SDP) to the request-response model. The SDP is usually collocated with the PEP and caches the returned authorizations from the PDP. When the PDP is overloaded or unavailable, the SDP can resolve authorization requests not only by reusing previous cached authorizations (a.k.a. precise recycling) but also by inferring new authorizations from cached authorizations (a.k.a. approximate recycling). Therefore, the SDP provides an alternative decision making source.

Figure 3(b) shows the pub-sub authorization architecture after adding the SDP. The SDP first tries to resolve the request from the PEP. If the SDP fails, it then publishes the request to the ENS and subscribes to the response. The returned response is added to the cache maintained by the SDP. As the cache size increases, more requests can be resolved by the local SDP, thus reducing the time to obtain responses and the load of the ENS.

One challenge of introducing an SDP is maintaining cache consistency. If the SDP is unaware of the policy update at the PDP, it may compute false responses. The pub-sub architecture provides an efficient way to propagating policy update messages: the SDP can subscribe to the policy update messages published by the PDP; after removing the affected cache, the SDP can publish a result message to the ENS indicating whether or not the update is successful.

4 Experimental Evaluation

The previous section presents the design of the authorization system based on a pub-sub architecture. This section presents an experimental evaluation of our design. Our evaluation sought to understand the availability gain of using the pub-sub model and studied the impact of the ENS on system performance in terms of response time.

4.1 Prototype Implementation

We implemented a prototype of the proposed pub-sub authorization system on top of Siena [7], a content-based ENS developed in Java. We chose Siena for two reasons: first, it provides the necessary functionality to study the feasibility and performance; second, it allows us to customize the code for further development and investigation.

In Siena, events are published as a set of attribute and value pairs. Attribute names are simply strings, and values are from a predefined set of primitive types, for which a fixed set of operators is defined. The following sample code shows how a PEP publishes a request to Siena.

```
Notification e = new Notification();
e.putAttribute("subject", "Sean");
e.putAttribute("object", "/etc/passwd");
e.putAttribute("access", "read");
siena.publish(e);
```

The subscriber subscribes to events by specifying filters using the 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 a PDP subscribes to the request for an object in Siena.

```
Filter f = new Filter();
f.addConstraint("object", "/etc/passwd");
siena.subscribe(f);
```

Matching in Siena is accomplished by a Binary Decision Diagram. The routing paths for notifications are set at the

time of subscription. In a distributed Siena deployment, a new subscription is stored and forwarded from the originating server to all the servers in the network. This forms a tree that connects the subscriber with servers. Notifications are then routed towards the subscriber following the reverse path of the tree.

As a basis for comparison, we also implemented a simple authorization system using a point-to-point architecture. The PDP was implemented as an JAVA Remote Method Invocation (RMI) service, and the PEP obtained authorizations from the PDP over RMI calls.

4.2 Evaluation Platform

The experimental pub-sub authorization system consisted of one PEP, one or more PDPs, and a centralized Siena ENS server. We assumed that all these components trust each other. The PEP was responsible for randomly generating requests and performing system measurement. The PDP resolved requests using a role-based access control (RBAC) policy. The Siena ENS mediated the communication between PEPS and PDPs. All these components ran in separate cluster nodes, each equipped with four Intel Xeon 2.33 GHz processors and 4 GB of memory, running Fedora Linux 2.6.24.3.

4.3 Evaluating Availability

We expect that using the pub-sub architecture leads to increased availability in the presence of PDP failures as a request can reach multiple PDPs. Our evaluation aimed to quantify this effect. We used the *percentage decrease of failed requests* perceived by the PEP as a metric to measure the increase in availability. In each experiment, we first measured the number of failed requests in the point-to-point implementation, and then repeated the same experiment and measured the failed requests in the pub-sub implementation with various configurations. Finally we calculated the percentage difference in the number of failed requests of two implementations. An increase in this difference indicates an improvement of availability as more requests have been successfully resolved.

We studied the influence of the following two factors on the percentage decrease of failed requests: (a) the number of PDPs and (b) the *overlap rate* between the resource spaces of two PDPs, defined as the ratio of the objects owned by both PDPs to the objects owned only by the studied PDP. The overlap rate served as a measure of similarity between the resources of two PDPs.

In the experiment, a RBAC policy was first generated for each PDP. Each policy contained 100 subjects, 1000 objects, and 3 access rights. We controlled the object space of each PDP to simulate certain overlap rates between different PDPs. Each PDP then booted, subscribing to all the objects for which it is responsible. The PEP then started sending 5,000 randomly generated requests sequentially. Each request was resolved with the flow described in Section 3 (see Figure 2(b)).

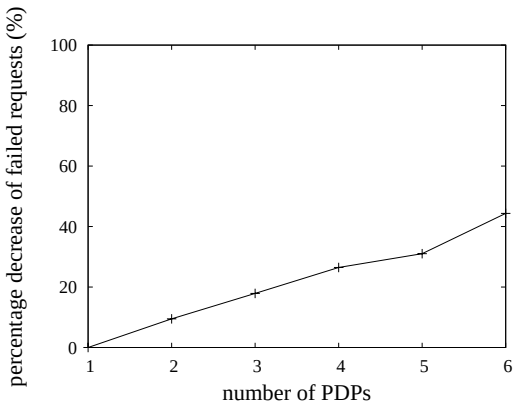
To simulate the failure of PDPs, each PDP switched between two modes: work mode and failure mode. In the work mode, the PDP computed and published a response after receiving a request. In the failure mode, the PDP simply ignored the request notification from Siena and did nothing. Therefore, a time-out request perceived by the PEP indicated the failure of the PDP. We used the number of elapsed requests as a measurement of time. We ensured that all the PDPs received all requests at the same pace so that their time clock was always synchronized. We used time-to-failure (TTF) to represent 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 times. The mean TTF (MTTF) was 100 requests and the mean TTR (MTTR) was 10 requests. We controlled the maximum TTF to be 500 requests and the maximum TTR to be 50 requests.

First, we studied the impact of the number of PDPs on the percentage decrease of failed requests. In the experiment, we varied the number of PDPs from 1 to 6 while fixing the overlap rate between PDPs at 10%. Figure 4(a) shows the results. As expected, the larger the number of the PDPs, the larger the percentage decrease of failed requests, which implies that the availability is improved. The reason is that the possibility to receive a response from other PDPs increased even when the studied PDP failed.

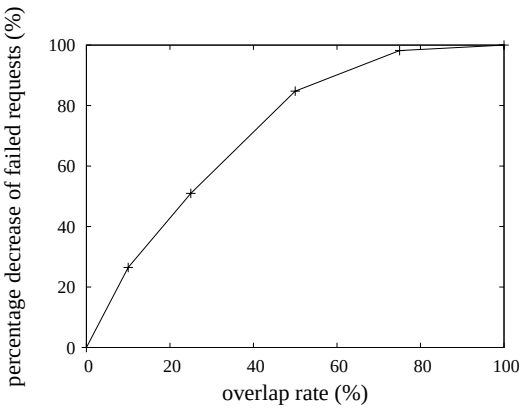
Second, we studied the impact of the overlap rate between PDPs on the percentage decrease of failed requests. In the experiment, we varied the overlap rate from 0% to 100% while fixing the number of PDPs at 4 (Figure 4(b)). As expected, the larger the overlap rate, the larger the percentage decrease of failed requests. When the overlap rate was 100%, the percentage decrease of failed requests was also 100%, which means all requests have been resolved. In addition, the results indicate that an increase of overlap rate provided diminishing returns. For instance, the first 10% overlap rate increase (from 0% to 10%) led to a 23% improvement, while the last 25% increase (from 75% to 100%) led to only 2% improvement. This suggests that the benefit of the pub-sub approach is most remarkable with a small resource overlap between PDPs.

4.4 Evaluating Performance

In evaluating performance, our goal was to understand how the use of the ENS impacts the system performance and how our two proposed performance improvement techniques help to mitigate this impact. We used response time



(a) Percentage decrease of failed requests as a function of the number of PDPs when the overlap rate is 10%.



(b) Percentage decrease of failed requests as a function of overlap rate when there are 4 PDPs.

Figure 4. Availability results.

as a metric of performance. The response time is measured as the time elapsed after the PEP generates a request until it receives the response for that request. In these experiments, we did not simulate the failure of PDPs. The PEP recorded the response time for each request. After every 500 requests the PEP calculated the mean response time and used it as an indicator of the response time for that period.

We have implemented two variations of the pub-sub architecture proposed in Section 3.4 to improve the system performance. In the first architecture, the PEP included its address in the request and the PDP returned response directly to that address over a Java RMI call (Figure 3(a)). We refer this approach as pub-sub-RMI. In the second architecture, we integrated pub-sub with SAAM by adding an SDP implementation [26] which is responsible for recycling previous authorizations and providing secondary authorizations to the PEP (Figure 3(b)).

Figure 5 compares the response time of the original pub-sub architecture (top curve), pub-sub-RMI architecture (middle curve) and point-to-point architecture (bottom curve). As expected, the pub-sub architecture led to longer response time, i.e., by 1.6ms on average, compared with the point-to-point architecture. In addition, using pub-sub-RMI helps to reduce the pub-sub response time, i.e., by 0.8ms.

We also studied the extent to which the system performance depends on the number of subscriptions in the ENS. First, we varied the number of subscriptions posted by the PDP: the PDP subscribed to either 100 or 1,000 objects. Figure 6(a) shows that the response time almost doubled with 1,000 object subscriptions. This result suggests that it is important to keep the number of PDP subscriptions small.

Second, we varied the number of subscriptions posted by the PEP. We ran an experiment where we disabled the "un-

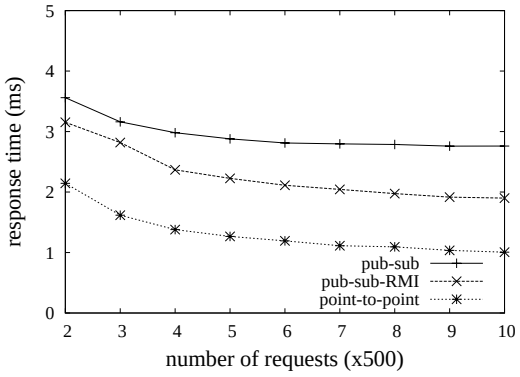
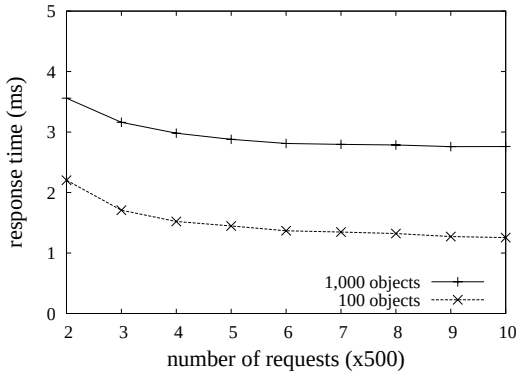


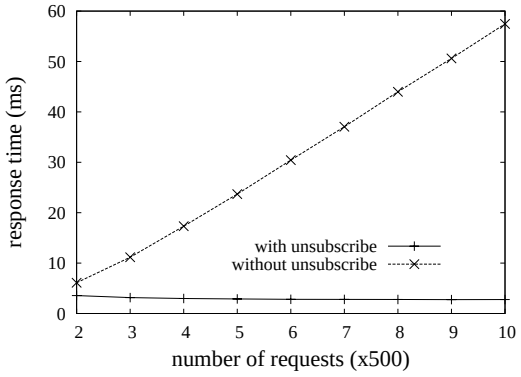
Figure 5. Response time comparison. Pub-Sub-RMI is the architecture in which the PDP returns responses directly to the PEP, bypassing the ENS.

subscribe" call (step 10 in Figure 2(b)) by the PEP, and thus the number of subscriptions kept increasing with the number of requests. Figure 6(b) shows that, without the "unsubscribe" call, the response time increased almost linearly with the number of requests, while with the "unsubscribe" call, the response time remains the same. This suggests that it is important for PEPs to remove stale subscriptions.

The results of availability evaluation suggest show that multiple PDPs with a large overlap rate provide high availability. Next, we studied whether this happens at the cost of system performance. We varied the number of PDPs from 1



(a) As the subscriptions posted by the PDP vary.



(b) As the subscriptions posted by the PEP vary.

Figure 6. The impact of the number of subscriptions on response time.

to 4. Each PDP subscribed to 1,000 identical objects (with 100% overlap rate). The result shows that an increase of the number of PDPs almost has no impact on the response time. A possible reason is that Siena optimized the matching algorithm for the situation when multiple subscribers have common interests (e.g., by storing only one copy of the subscription). Evaluating the case when PDPs post diverse subscriptions is subject to our future work.

We also ran an experiment to study the extent to which the integration of pub-sub with SAAM reduces the response time. The PEP sent 10,000 randomly generated requests in total, which accounted for one third of the total possible requests. We evaluated both precise recycling, where the SDP only used previously cached responses, and approximate recycling, where the SDP inferred new responses from cached responses. Figure 7(a) shows the results. As expected, both precise recycling and approximate recycling helps to reduce the response time, while approximate recycling achieve better results. This is because many requests can be resolved by the collocated SDP. When the cache size increases, the decrease in response time is more significant.

In the above experiment, we used a simple authorization logic: the PDP resolves any request in less than 1ms. In reality, however, the authorization logic might be complex and takes longer time. Therefore, in the next experiment, we manually added a 20ms delay to each response returned by the PDP to simulate a complex authorization logic. Figure 7(b) shows that using SAAM helped to reduce the response time more significantly in this case. In particular, the response time decreased from 22ms to 6ms when approximate recycling is used. The reason is that more and more requests can be resolved by the local SDP, instead of being resolved by the slow PDP.

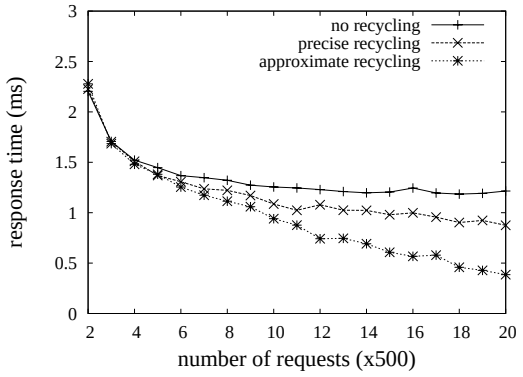
4.5 Discussion

Our evaluation confirms the analysis in Section 3.2: the use of pub-sub model leads to higher availability, and this improvement increases with the number of PDPs and their resource overlap rate. In particular, when the overlap rate is small, even a slight increase in the overlap rate leads to a significant improvement in availability. Therefore, even in a large-scale system without a large resource overlap between applications, our approach can be still beneficial.

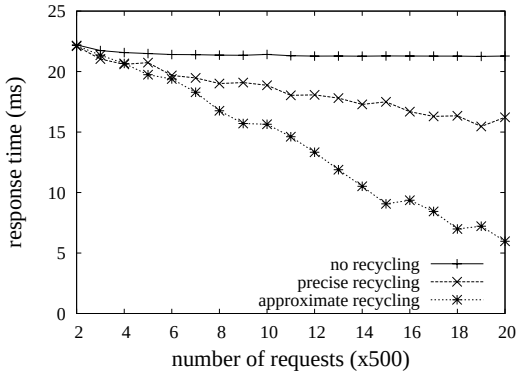
Additionally, our results confirm that the system performance is impacted: comparing the pub-sub architecture with the point-to-point architecture, we observed an 1.6ms increase in response time. However, the importance of this overhead must be judged in conjunction with the cost of making decisions at the PDP, which can be one order of magnitude larger. We have also demonstrated that it is important to reduce the number of subscriptions to the ENS, either by using alternative subscription mechanisms at the PDP or having "unsubscribe" call to remove stale subscriptions by the PEP. Our results also indicate that providing high availability by using multiple PDPs with large overlap rate does not impact the response time. Finally, our results confirms that our two proposed approaches to improving performance are effective. Specifically, by integrating pub-sub and SAAM, the response time is significantly reduced when the PDP's authorization logic is complex.

5 Related Work

Pub-sub systems have been an active research area. Early pub-sub systems use a subscription scheme based on the notion of topics, and have been implemented by several industrial solutions (e.g., [24]). Content-based systems improve



(a) PDP With simple authorization logic (authorization compute time at the PDP < 1 ms).



(b) PDP With Complex authorization logic (time = 20ms).

Figure 7. Response time results when pub-sub is integrated with SAAM.

over systems based on topics by introducing a subscription scheme based on the actual content of the considered events (Gryphon [1], Siena [6], Elvin [22].)

Due to their flexibility and scalability, pub-sub systems have been used as the basic communication and integration infrastructure for many application domains, such as Internet games [4], mobile agents [20], user and software monitoring [13], mobile systems [9], among others. The contribution of this paper lies in applying the pub-sub model to access control systems.

To improve the availability of access control systems, caching authorizations has been employed in a number of commercial systems [15, 11, 18], as well as several academic distributed access control systems [2, 5]. However, these and other approaches only compute precise authorizations and therefore are only effective for resolving repeated requests. Beznosov [3] introduces the concept of recycling approximate authorizations, and later Crampton et al. [8] formally define SAAM and introduce the concept of SDP. The SDP can resolve new requests by extending the space of supported responses to approximate ones. In other words, SAAM provides a richer alternative source for authorization responses than the existing approaches do, thus improve the availability of access control systems. To further improve the performance and availability of access control systems, Wei et al. [27] explore the cooperation between multiple SDPs and combine their cooperation with the inference.

6 Summary and Future Work

This paper presents an authorization architecture that uses the pub-sub model to improve the robustness and manageability of the access control system. We introduce the overall system design and define the requirements for the core event notification service: robustness, performance and security. Our evaluation demonstrates that our design achieved better availability, and the impact of integrating the pub-sub channel on performance is small and can be further reduced by returning authorizations bypassing the pub-sub channel or by integration with SAAM.

For future research, we will further refine the security analysis of our pub-sub authorization model and develop countermeasures to the identified attacks. Second, we will further explore how PEPs can handle the case when policies become inconsistent among PDPs. Third, we will develop application-specific load balancing techniques, since an ENS node in a distributed pub-sub authorization system may suffer from uneven load distribution due to different population densities and interests of PDPs.

Acknowledgments

Initial ideas of using pub-sub for access control have benefited significantly from the presentation and discussion of [3] at the New Security Paradigms Workshop (NSPW) '05. The authors are grateful to the anonymous reviewers for their helpful comments, and Kirstie Hawkey for improving the readability of this paper. Members of the Laboratory for Education and Research in Secure Systems Engineering (LERSSE) gave valuable feedback on the earlier drafts of this paper. Research on pub-sub by the first and third authors have been partially supported by the Canadian NSERC Strategic Partnership Program, grant STPGP 322192-05.

References

[1] 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.

[2] L. Bauer, S. Garriss, and M. K. Reiter. Distributed proving in access-control systems. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy (S&P'05)*, pages 81–95, Oakland, CA, 2005. IEEE Computer Society.

[3] K. Beznosov. Flooding and recycling authorizations. In *Proceedings of the New Security Paradigms Workshop (NSPW'05)*, pages 67–72, Lake Arrowhead, CA, USA, 20-23 September 2005. ACM Press.

[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] K. Borders, X. Zhao, and A. Prakash. CPOL: high-performance policy evaluation. In *Proceedings of the 12th ACM conference on Computer and Communications Security (CCS'05)*, pages 147–157, New York, NY, USA, 2005. ACM Press.

[6] 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.

[7] 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.

[8] 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'06)*, pages 111–120, Lake Tahoe, CA, USA, June 7–9 2006. ACM Press.

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

[12] 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.

[13] 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.

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

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

[16] 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.

[17] J. Markoff and S. Hansell. Google's not-so-very-secret weapon, 2006.

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

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

[20] 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.

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

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

[23] P. Strong. How Ebay scales with networks and the challenges. In *the 16th ACM/IEEE International Symposium on High-Performance Distributed Computing (HPDC'07)*, Monterey, CA, USA, 2007. ACM Press. Invited talk.

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

[25] 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. ACM Press. Keynote address.

[26] Q. Wei, J. Crampton, K. Beznosov, and M. Ripeanu. Authorization recycling in RBAC systems. In *Proceedings of the thirteenth ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 63–72, Estes Park, Colorado, USA, June 11–13 2008. ACM.

[27] Q. Wei, M. Ripeanu, and K. Beznosov. Cooperative secondary and approximate authorization recycling. In *Proceedings of the 16th ACM/IEEE International Symposium on High-Performance Distributed Computing (HPDC)*, pages 65–74, Monterey Bay, CA, June 27-29 2007. ACM Press.

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