Disclosure Control in Multi-Domain Publish/Subscribe Systems

Jatinder Singh
Computer Laboratory
University of Cambridge, UK
jatinder.singh@cl.cam.ac.uk

David M. Eyers
Dept. of Computer Science
University of Otago, NZ
dme@cs.otago.ac.nz

Jean Bacon
Computer Laboratory
University of Cambridge, UK
jean.bacon@cl.cam.ac.uk

ABSTRACT

Publish/subscribe is an effective paradigm for event dissemination over wide-area systems. However, there is tension between the convenience of open information delivery, and the need to protect data from unauthorised access. Publish/subscribe security models tend to focus on protecting the client API, or encrypting events and managing disclosure through key distribution. However, some application environments require more stringent, fine-grained controls governing precisely the data disclosed and transmitted given particular circumstances. In this paper, we present Interaction Control, a policy model that overlays context-aware, point-to-point (hop-level) controls onto a publish/subscribe network. The approach is unique as it allows granular control over i) the construction of the dissemination network, and ii) the information flows *within the network*. Interaction Control was designed considering legal obligations, to enable those responsible for information to transmit data on a *need-to-know* basis. Security policies set the bounds for communication, enforced only where necessary at specific points of the publish/subscribe process, to provide control while retaining the efficiency benefits of the paradigm. We present implementation details and results showing that any security overheads must be considered with respect to the overall network load.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*security and protection*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed applications*

General Terms

Security, Design, Legal Aspects

Keywords

publish/subscribe, security, information flow control, access control, policy enforcement, data governance

DEBS'11, July 11–15, 2011, New York, New York, USA.

1 Introduction

Publish/subscribe (pub/sub) is an effective paradigm for wide-area event distribution, in which *events* encapsulate self-contained data updates. *Clients* (information producers and consumers) are *decoupled*: producers produce (publish) events and consumers register their interest in (subscribe to) information they wish to receive. Pub/sub is *push-based*, where clients communicate through the pub/sub middleware. Routing is *information centric*, delivering relevant—based on the type, topic and/or content—events from producers to consumers as they occur. The paradigm is efficient and scalable, exploiting commonalities between client preferences to avoid redundant transmissions.

Client decoupling favours anonymous communication: the consumers are unaware of the information producers' identities/addresses/locations, and *vice-versa*. Such information could be encapsulated in the event itself, but even then the producer has no control over who receives their publications. The paradigm often includes the use of *brokers*—routing entities that interconnect to provide the pub/sub service. Clients communicate through brokers, where brokers inspect events (at varying degrees) to make routing decisions.

For some applications it may be enough to simply secure access to the entire pub/sub service, e.g. those services where clients register to subsequently receive information. This is typically sufficient for the financial services (stock quote) scenarios common to pub/sub research. However, there exist other application environments with more stringent security requirements that require wide-area notification services. Generally, pub/sub lacks the means to control the data released to a (particular) consumer. Often security concerns are context-dependent, in that the appropriate constraints for a transmission depends on the circumstances. Further, when considering wide-area services, such as at a national-level, events will flow between and through various domains of administrative control. Infrastructure (brokers) will be managed by different entities that co-operate to provide communication services. The security requirements can, and in practice will, often differ depending on the domains involved in a particular delivery operation.

This paper addresses these concerns through the presentation and evaluation of *Interaction Control* (IC), an approach that essentially overlays a *point-to-point* (i.e. hop by hop) security model onto a pub/sub network to enable precise control over connections and the information transmitted. It is novel in that it allows context-aware control *within* a broker network, at hop-level granularity, to account for notions of responsibility and the naturally varying levels of trust be-

tween components in a wide-area network. It enables security policy to be enforced at specific points of the pub/sub process and in specific circumstances, to facilitate control while retaining the efficiency benefits and delivery semantics (e.g. clients specifying their interests) of the paradigm.

IC was developed as part of our work concerning infrastructure for supporting healthcare services. We begin with a brief overview of the data governance requirements of England's *National Health Service* (NHS). We then detail IC, describing the policy rules that control broker-broker and broker-client connections, and that filter and transform information as it flows throughout the network. Next we describe our IC implementation, where controls are integrated into a pub/sub database system to facilitate policy enforcement, context management, event storage and audit. We present results indicating that despite the local (broker) overheads of policy enforcement, restrictions can actually improve the overall efficiency of a particular workload. We then conclude with discussion and areas for further research.

2 Healthcare Information

This work stems from our research into middleware for supporting federated national-level health services.[1] Healthcare is an interesting environment for the investigation of security issues, as information must be shared, yet protected.

The care process is highly collaborative, involving the sharing of health information between professionals from various health service providers, as well as with other supporting institutions, such as government and insurance companies. However, personal health data is highly sensitive, and remains so over time. Confidentiality underpins the carer-patient relationship. Those who handle personal health data as part of the care process are **legally responsible** for maintaining its confidentiality, which includes service providers that manage and maintain the technical infrastructure for their service [15]. In England, information may only be shared if patients have given consent, subject to certain exceptions. In many situations, privacy concerns *user* anonymity. Healthcare differs in that although patient data must remain confidential, system users are generally medical professionals, whose actions (send/query/receive) must be visible for reasons of accountability.

2.1 Information Sharing Protocols

For an organisation to meet its data management responsibilities, information is best shared on a *need-to-know* basis [16, 15]: disclosing only that information required given the circumstances. Appropriate disclosure is often context-dependent, e.g. access restrictions may be relaxed in emergency situations.

A national-level health service consists of a number of providers, whose information requirements differ depending on the particular service(s) they provide. For instance, a pharmacist requires different information from a pathologist, or an insurance agency. Often a health incident will be relevant to several professionals in several organisations, e.g. it may involve ordering medication, specialist testing and organising home care. A provider will have an understanding of the reasons for dealing with another provider, and thus the general information that the other party requires.

Health institutions define an *information sharing protocol* that describes precisely the data that may be shared in the particular circumstances [7]. It is designed to suit local practice and procedure, but must also account for patient consent, and other legal/general practice requirements. The protocol facilitates explanation of the data shared for a course of treatment [26], which is useful for obtaining consent. Patients tend to trust their physicians to act as their information gatekeepers [3], meaning a prudent information sharing protocol will often meet patients' confidentiality expectations. However, the general protocol must be qualified by any patient-specific disclosure preferences.

2.2 Domains

We define a *domain* as a unit governed by independent administrative policy that provides particular services [1]. Clearly, domains exist in many application environments. A domain naturally maps to an organisation (a legal entity), or any body that is responsible for its own procedures, defines its own policies and manages its own infrastructure. Health service examples include a hospital, surgery, pathology laboratory or insurance company. A domain hosts a number of *entities*, such as doctors, nurses, software, accountants and managers, often through service/employment contracts. As part of its operations, a domain collects, stores and forwards information relevant to the service it provides. Each domain must appropriately share information with entities in the local environment, and with other domains. It follows that a domain is responsible for the information it handles.

We assume that a domain has control over its local communication infrastructure (brokers), and defines its local policy in accordance with that imposed by higher-level domains, global directives and legislation.

2.3 Sharing Infrastructure

Healthcare is a data-driven environment and thus suits push-based communication. This is not only to notify and alert in situations of concern (e.g. emergencies), but also to ensure that the numerous parties/domains involved in the care process are kept aware of the current situation and operate on the latest representation of state. Unlike much work supporting healthcare, we do not focus on the patient record, but instead consider the streams/dataflows (events) necessary to support the health service. These flows represent patient data, the actions of practitioners, support services such as billing, inventory and insurance, governmental information requirements, etc. Indeed, an event-based messaging middleware is considered an integral component of technical healthcare infrastructure.[2]

2.4 Threat Model: Security & Responsibility

Security research is usually presented along with a defined *threat model* that describes adversaries, system attacks and the proposed countermeasures. In our case, we address concerns that are orthogonal—and complementary—to those of specific security mechanisms: we are concerned with providing the means for users of the system to meet their responsibilities. We consider control *only* as it pertains to pub/sub (cf. more general, non-middleware security approaches) because our interests concern transmission within information-centric networks. Pub/sub generally deals with open information delivery; our work considers controlling dissemination in pub/sub by allowing definition of the bounds for

[1] We describe healthcare from the English NHS perspective.

[2] For instance, see the NHS *Transaction Messaging Service (TMS)* and the *TMS Event Service (TES)* at `http://www.connectingforhealth.nhs.uk`.

communication. Domains are autonomous, and co-operate with entities and other domains by sharing information, but this does not imply mutual, absolute trust. Clearly, it is wholly inappropriate to allow one to access any information they desire. Here security is realised by domains controlling the data they transmit, in line with their legal and social responsibilities. There must be the means to allow sharing on a *need-to-know* basis. This is a technology directed at realising the controls required by law, rather than simply the mechanisms that enforce access control, for example.

In this model, responsibility is associated with data. Each recipient becomes duly obliged to protect the information that they receive. The responsibility for transmission passes with the data itself; a domain meets its data management obligations by passing information to a connected entity/ domain in accordance with (sound) local policy. It is not responsible for the recipient's shortcomings if the disclosure was appropriate. Indeed, such a responsibility model tends to exist in any environment where separate administrative domains interact. Of course, this assumes that a domain has some concept of why it communicates with another entity/domain: the information exchange fits within some premeditated arrangement. This, however, is not unreasonable given the existence of sharing protocols in healthcare, and more generally, workflows in other application areas. We use healthcare as a real-world illustration where the responsibility model is overt, and legally imposed. Security is realised by a domain sharing information in line with its sharing protocol, qualified by any patient consent preferences.

IC enables these sorts of governance regimes to be built into middleware, by allowing policy to be defined in brokers for enforcement in particular circumstances at specific points of the pub/sub process. Middleware enforcement means security policy is consistently applied across applications/clients. Assuming a domain controls its technical infrastructure (a NHS goal [6]), IC allows a domain to define policy to set the bounds for communication, ensuring that data leaves *its* broker network—transferred to clients, or brokers in remote domains—only when appropriate.

3 Related Work

This work aims to enable those responsible for information to manage their communications, through explicit control over the information channels and event flows in a pub/sub system. We found that the literature in the area of pub/sub security addresses different concerns from those described.

Much research considers the use of encryption in pub/ sub services, where events are encrypted and then transmitted throughout the network. Some transmissions may be to parties that are unauthorised to access the underlying data. However, access to event content is managed through an additional step of distributing encryption keys: keys are only provided to those who are authorised to access event data. Approaches differ in whether the broker network is trusted, i.e. whether the brokers have access to event content [25, 18], and how keys are generated and shared [17, 13]. The liberal transmission of encrypted events throughout a pub/sub network, where key allocations control security, is unsuitable where information is perpetually sensitive. This is because a compromised key, or broken encryption scheme[3] at *any* time in the future risks the inappropriate disclosure of a history of prior events. Such an approach runs against the described notions of control and responsibility. It follows that events, even if encrypted, should only flow to those authorised to receive that data. Further, encryption-based mechanisms impose key-management overheads, and tend not to easily deal with situations where access policy is context sensitive.

Other work concerns the enforcement of restrictions at the edges of a broker network. The delivery of events to a client can be controlled by restricting and/or validating subscription filters [14, 30]. These operate on event type/content. Access control mechanisms, such as *Role-Based Access Control* (RBAC) [2] and *Access Control Lists* [30] can protect access to the pub/sub API. These are described in the context of a single administrative domain, though [2] suggests that a *web-of-trust* can govern broker access to event types. Such approaches are limited in their flexibility. The responsibility model (§2) requires the possibility to govern *all* communications and connections, not just with clients (at the edge of the broker network), but also with other brokers—particularly where components reside in another administrative domain. Further, context-aware controls—considering more than just event type/content—are required to allow for more granular security policy. For example, the time of day, (client) location and situation's severity can be relevant to whether particular information should be disclosed.

A *scope* [11, 12] is a grouping structure that bundles sets of pub/sub brokers and clients, constraining the visibility of events to its members. A scope sets the boundaries for transmission, allowing control over the propagation of events to other scopes. A scope aligns well to the concept of domains. However, there are situations where different levels of visibility are required for members of the same scope/domain. Thus, it is necessary to enable control over the communication to *each* principal, regardless of domain structure.

Wun and Jacobsen [29] describe a generic policy framework that couples what are essentially *Event-Condition-Action* (ECA) rules with pub/sub operations. As the framework aims to be open and expressive, it addresses different concerns to those of a security infrastructure. For instance policy actions are not semantically defined, in that an action can undertake any (possible) function. This brings flexibility, though for security the functionality of policy actions should be defined to provide certainty, facilitate correctness and aid conflict resolution. Wun and Jacobsen describe policies for unstructured pub/sub overlays, where policy may be defined by clients, attached to events, advertisements and subscriptions to be enforced as they flow through the network. However, this is unsuitable in an environment of responsibility between federated co-operative domains, as it complicates policy combination and precludes separate notions of trust and accountability. Enforcement in [29] occurs with particular pub/sub operations, the authors concentrating on *post-matching* enforcement, where actions are executed after a filter match to avoid the overheads of separately evaluating policy conditions. However, this is inappropriate in certain security contexts, e.g. where a policy executed on delivery could circumvent a subscription filter (§5.1). For proper control, actions need not be coupled to pub/sub operations, but should be enforced where necessary.

The focus of IC differs from that of other pub/sub security approaches in that it brings about security by controlled disclosure, where those responsible for data meet their obli-

[3]For instance, we have seen increases in computing power cause encryption methods to be disregarded; e.g. 56-bit DES keys: `http://www.rsa.com/rsalabs/node.asp?id=2100`.

gations by setting granular, *contextually-sensitive* bounds for pub/sub communication. IC is unique in that it explicitly addresses pub/sub security by enabling specific control over *all* connections (topology construction) and event flows in a pub/sub network, including those between brokers. As a broker enforces local policy against its direct connections, issues of trust remain at the application-level. Our previous publications regarding IC present our initial work with respect to enforcement [24] and the controls as they apply to events [22, 23]. Here we present the whole, mature model, describing for the first time how the governance mechanisms apply in a *distributed* broker network, how broker interconnections are controlled, the management of requests and some engineering specifics including overhead analysis.

4 Interaction Control: Rules

The goal of *Interaction Control* (IC) is to provide the mechanism for realising a need-to-know, local responsibility model in a pub/sub middleware. This involves giving a broker, managed by a domain with a certain responsibility, the ability to govern all transmissions—including events, advertisements and subscriptions—to directly connected clients and brokers. Control is achieved through a policy model that enforces rules at specific points of the pub/sub process. Each broker maintains a set of rules to enforce on its direct connections.[4] Here we describe the specifics of these rules.

Background and Assumptions Like much pub/sub work, we consider application-layer routing. *Transport-layer security* [9] protects lower-layers of the stack.

We describe IC for *type-based pub/sub*, where each event conforms to a particular type, from the set of types τ. Each type $t \in \tau$ consists of a name and a particular set of attributes of specific data types. Events flow through a *channel*, which is a unidirectional, typed (logical) communication path between a broker and a connected client/broker. A number of channels can exist within a connection.

Clients communicate with brokers, where brokers interconnect to form the distributed pub/sub service. The only assumption of trust is that a client entrusts its local (directly-connected) broker to operate on its behalf. An event dissemination tree is built by brokers propagating advertisements and subscriptions [28], similar to the popular advertisement forwarding approach of Siena [4]. Also like Siena, we assume that brokers hold a particular position in the network topology—that *links* (broker interconnections) exist for a reason, to share particular information with specific brokers. This is in line with the notions of domain knowledge, co-operation and responsibility as described in §2. Note that this does not entail that brokers are continually connected; instead, the *possible* connections between brokers are pre-authorised (discussed later). In IC the connections themselves are dynamic, in the sense that a broker may connect/disconnect from another.

Broker Context Disclosure policy is encoded in rules defined for (particular) brokers. A broker enforces policy against its direct connections in accordance with its local ruleset. IC rules are context-senstive, where context encapsulates anything accessible by the enforcing broker. Thus, "context" is implementation specific. Rules reference context using sets of *predicates*, which a broker evaluates as part of the enforcement process. *Credential predicates* assert characteristics about a principal, including their unique identifier, group memberships, qualifications, roles or certificates that they hold. Such predicates define the class of users to which a rule applies. *Environmental predicates* refer to other aspects of context, such as system type, workflow state or patient status (e.g. stable, critical). See [22] for more details regarding IC predicates.

A *permission attribute* consists of an attribute name and type, the value for which a client must include with their advertisement/subscription. Rule predicates can reference the values of the permission attributes supplied in the request. This supplements broker-accessible context to allow computations with data outside the system, e.g. requiring the inclusion of a `patient_id` enables rules to verify a treating relationship between the subscriber and patient.

4.1 Authorisation Rules

Authorisation rules specify when to authorise the establishment of connections and event-channels between components. There are two types of authorisation rule:

Connection Authorisation *Principal authorisation rules* govern the connection of a principal (client/broker) to a broker. We define $\mathcal{C}$ to be the complete set of credential predicates. Each rule refers to a set of credential predicates $C \subseteq \mathcal{C}$ of the remote principal. This brings flexibility, in that a rule can apply to a specific principal by referencing its unique `id`; or to sets of principals, e.g. by referencing a role such as `doctor`, thus avoiding a separate rule for each. The rules apply equally to connecting brokers as they do to clients. These rules are simple as they merely concern connection, other rules govern channel establishment and transmission.

Request Authorisation An advertisement or subscription can be characterised as a *request*. A client issues a request to a broker to publish (through an advertisement) or subscribe to information. If authorised, a channel is established for the (directional) flow of events of the specified type.

Request authorisation rules define the circumstances in which channel establishment is allowed. Each such rule can be represented as a tuple of the form: $(rt, t, C, E) \in RT \times \tau \times \mathbb{P}(\mathcal{C}) \times \mathbb{P}(\mathcal{E})$. The set of *request types* is defined as $RT = \{\texttt{advertisement}, \texttt{subscription}\}$ and is used to indicate whether the rule applies to advertisement or subscription requests. The event type that this rule refers to is t. We use $\mathbb{P}$ to mean power-set: the set of all subsets. The target of the rule is defined by the set of credential predicates ($C \subseteq \mathcal{C}$) that are matched against those held by the requesting client. A set of environmental predicates ($E \subseteq \mathcal{E}$) further refine the circumstances in which the rule applies. To enable evaluation, a client must supply the permission attributes required by the rule's predicates.

Channels are durative, and persist until the channel is closed. Authorisation depends on context, thus a change in state can affect rule applicability. As such, the environmental predicates of an authorisation rule can be defined as *monitored*, causing re-evaluation of the request should the value(s) change. For example, a rule might authorise a doctor to subscribe to data while on duty in a ward: $C = \{\texttt{doctor}\}, E = \{\texttt{onDuty(}\textit{user}\texttt{)}\}$. If the `onDuty(`*user*`)` predicate is monitored, the request will be re-evaluated when the doctor ends his shift, in this case closing the channel.[5] As

[4] Disclosure policy is encoded in rules distributed to specific brokers to effect appropriate disclosure. A domain may have a number of brokers, but may define different rules for each, depending on the particular broker's role—e.g. in §7.1.

[5] Assuming that no other rule authorises the channel.

credentials represent the intrinsic characteristics of a principal, credential predicates are implicitly monitored as they define the rule's target(s). For example, if a doctor is struck-off, it is important that his access rights are reconsidered.

Request authorisation rules control access to event types by ensuring the legitimacy of event channels. Such rules are useful as they avoid the (potentially expensive) evaluation of authorisation conditions on each access attempt (event). Once a channel is established, individual events are controlled through imposed conditions and transformations.

4.2 Event Controls

The following two rules control the propagation of events through an event channel.

Imposed Conditions restrict the tranmission of events through a channel. They are similar to subscription filters, except that they are specified by policy rather than the subscriber. For example, a filter can ensure that details of a condition are not sent to a particular doctor.

An imposed condition rule r is a tuple: $(rt, t, C, E, R, h) \in IP \times \tau \times \mathbb{P}(\mathcal{C}) \times \mathbb{P}(\mathcal{E}) \times \mathbb{P}(\mathcal{R}) \times H$. The rule restriction predicates $R \subseteq \mathcal{R}$ act to filter the event. These are evaluated in the context of an event instance, thus they may reference attributes of the event type. Also specified are a subset of credential predicates from $\mathcal{C}$, an event type (from τ) and an interaction point $IP = \{\texttt{publication}, \texttt{subscription}\}$ that defines the target channel type for the imposed conditions.

Restriction filters may encode sensitive information. For instance, the restriction $\texttt{Treatment} \neq \texttt{HIV}$ might exist to prevent a particular patient's information from flowing to one of their doctors. Revealing this restriction to the doctor suggests that the patient could be HIV positive. Each rule selects from the set $H = \{\top, \bot\}$ as to whether the imposed conditions are disclosed to the client or not. If hidden, the filters are imposed silently: publications appear to be accepted but are ignored, while subscribers have events filtered without their knowledge of the restriction.

Event Transformations These transformation rules alter an event. They can enrich, degrade or produce new events that are related to the original event in some application-specific manner. While typically considered for interoperability [27, 11], from a security perspective transformations allow more than binary (permit/deny) access control, as event data can be tailored to the situation. Further, middleware transformations avoid clients publishing multiple instances of a semantically similar event with differing levels of visibility (see §7.2). An example transformation might obfuscate location data for a remotely monitored patient, save in emergency situations. In §7 we describe a transformation rule to effect data segmentation regarding prescriptions.

A transformation rule r is a tuple: $(ip, t, C, G, f, t', c) \in IP \times \tau \times \mathbb{P}(\mathcal{C}) \times \mathbb{P}(\mathcal{G}) \times F \times \tau \times D$ where t, ip and C perform the same previously discussed function of relating the rule to the relevant channels. $G \subseteq \mathbb{P}(\mathcal{G})$ is the set of predicates that define when the transformation is performed. G is evaluated in the context of the event—if G holds, the transformation applies.[6] The transform occurs through the function $f : \tau \rightarrow \tau \in F$, which takes an event of type t and returns an event of type t': an altered event of the same or another type.

For flexibility, the $c \in D = \{\top, \bot\}$ tuple element allows each rule to specify whether or not it is *consumable*. Consumable functions prevent the original event from propagating further—only the result from the function proceeds to the next stage of the pub/sub process. Non-consumable transformations allow all events to proceed.

[6]This is subject to conflict resolution definitions—see §5.

4.3 Broker Controls

As mentioned, brokers interconnect to form a distributed dissemination network. An event dissemination tree is built by the brokers propagating advertisements and subscriptions [28], similar to the advertisement forwarding approach of Siena [4]. A broker connects to another via a *link*, which (logically) differs from a client connection as brokers only forward requests through links. Principal authorisation rules authorise links, meaning that rules can apply to classes of brokers, avoiding the enumeration of every broker combination. This better suits the scale of national-level services.

A broker maintains a set of policies to control the flow of information to *directly connected* principals. IC does not distinguish between events and requests received from clients and those from brokers that may be forwarding on behalf of others. Instead, an event received through a link is treated as a publication from the adjacent (remote) broker, and an advertisement or subscription received through a link is characterised as a request issued by the adjacent broker. IC rules are defined for, and enforced against, brokers just as they are for clients—subject to the same policies and enforcement processes. In this way, IC is a *point-to-point* security model since a rule *only* concerns interactions with the next hop. This is in line with the described model of responsibility, giving each domain *local control* over the transmissions of their brokers. Rules are defined to only restrict flows when necessary, concerning certain transmissions in particular circumstances. In this way, rules set the bounds for transmission. Requests and events are able to flow freely to allow wide-scale distribution, subject to the (necessary) constraints imposed by intermediate brokers (domains).

4.4 Request Forwarding and Processing

If a broker authorises a request and creates a channel, the request is forwarded to other brokers to establish the dissemination network. This requires control.

The mechanisms for controlling request propagation are similar to those for events, where requests can be transformed and filtered before transmission to the adjacent broker. The rules relevant to a broker depend on its position in the network infrastructure. Such restrictions are defined by administrators, who have specific knowledge/concerns of (local) network topology. Forwarding restrictions set the *general* boundaries for interaction.

Request Filters Conditions can be imposed on links to filter the requests forwarded to a broker. This is to protect any sensitive data related to what is on offer; e.g. filters can ensure that advertisements are forwarded only to brokers authorised to subscribe to the type, or that subscriptions are forwarded only to brokers in domains with a treating relationship to the patient who is the subject of the request.

The rules take the form $(rt, t, C, R_r) \in RT \times \tau \times \mathbb{P}(\mathcal{C}) \times \mathbb{P}(\mathcal{R})$. They are defined for a particular request type (rt), advertisement/subscription, and event type (t). The set of credential predicates (C) refers to that of the remote broker. The restrictions (R_r) are evaluated in the context of a request pertaining to an event type, *not* in the context of the event itself; thus the filter predicates reference *request content*, along with other aspects of environmental state.

Request Transformation rules allow a request to be tailored specifically to a remote broker. The rules facilitate interoperability, e.g. translating an identifier from a local system to a shared NHS ID, and controlled disclosure, e.g. anonymising identifiers or removing any sensitive information contained within the request.

Request transformation rules alter a request before propagation to an adjancent broker. A request transformation rule is a tuple of the form $(rt, t, \varsigma, G_r, f) \in RT \times \tau \times S \times \mathbb{P}(\mathcal{G}) \times F$ defined for a particular request type rt. The rule's guarding predicates (G_r) are evaluated in the context of the request. The transformation function f takes a request and returns a modified one. Request transformations implicitly consume (i.e. replace) the incoming request tuple. The rules are defined for a particular link, specified by ς, which identifies the principal authorisation rule authorising the connection to the adjacent broker. Events convey information, and thus it may be appropriate to transform and transmit a number of events. Request transformations, however, are enforced on (request) propagation to govern the establishment of channels with a particular broker. Thus only a single transformation function may be defined for an event type and link.

4.5 Rule Fragmentation

We do *not* encapsulate all restrictions—authorisations, restriction filters and transformations—into a single rule structure. Instead, rules are defined independently to allow a many-to-many relationship between the rules. This brings flexibility as it enables different sets of rules to apply in different circumstances. Further, it removes the need to reauthor existing rules to deal with specific requests, e.g. we avoid modifying all rules concerning an event type when introducing a filter for an individual patient.

5 Interaction Control: Enforcement

Each IC broker maintains a set of rules that it enforces at particular points of the messaging process. In this section we describe enforcement. Given that IC rules are context sensitive, several can apply at an enforcement point. We begin by detailing the general process of enforcement. We then discuss the application of multiple rules, and methods for resolving conflict.

5.1 Event Flow Enforcement

A broker enforces its policies as an event passes through a particular enforcement point. A broker's processing of an event is illustrated in Fig. 1.

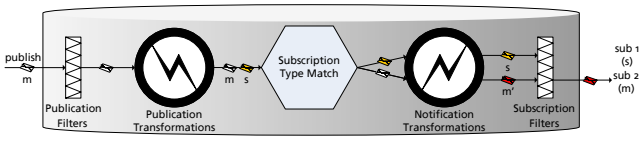


Figure 1: A broker's enforcement process for the publication of the event `m`.

A publication first is validated against any conditions imposed on the publication channel. If the filters are satisfied, the event is subjected to the relevant publication transformations. In this example, one transform applies that takes event `m` and produces `s`, an event of a different type. As it does not consume the original event, both events move to the delivery phase. Note the output of a transformation function moves to the next stage of processing (see §5.3).

Delivery involves moving a copy of the event through each active notification (subscription) channel for its type. In this example, there exists one subscription for each type. The events are subjected to the notification transforms applicable to the subscriber in the circumstances. Here, a transformation function exists for `sub 2` that consumes event `m`, returning a modified version `m'`. There is no transformation defined for `s`, so it passes through unperturbed. The final stage involves evaluating the event against the subscription (and imposed) filters. Only event `m'` is delivered, as `s` fails to satisfy `sub 1`'s filter.

Subscription filters act as a barrier to prevent certain information from leaving the broker. IC involves a two-phase subscription matching process. First, the events are matched against active subscriptions considering only the event type; filter predicates are applied *after* the notification transformations. This prevents notification transformations from circumventing any filter restrictions.[7]

5.2 Request Enforcement

A connection must exist before requests and events can be transmitted. A broker will allow a connection if the request is authorised by a principal authorisation rule. On receipt of an advertisement request, the broker determines whether a request authorisation rule permits the request in the circumstances. If so, the channel is established. The advertisement is then forwarded to each adjacent broker after the execution of any request transformation, subject to validation against any advertisement filters defined for the link and event type. The request is only forwarded to those brokers that have not already received a similar advertisement.[8]

The process is similar for subscription requests. After the establishment of the subscription channel, the subscription request is forwarded to the adjacent brokers that advertise the event type, subject to any subscription transformation functions or filters defined for the advertising brokers.

5.3 Multiple Rules

As rules are context sensitive, it is possible that several rules (of the same type) apply at an enforcement point. An authorisation rule must hold to authorise a connection or the establishment of a channel. The rule's monitored conditions and the principal's credential allocations are monitored to trigger re-evaluation should values change. A number of imposed condition rules may apply at an enforcement point. The filter predicates of all relevant imposed condition rules are evaluated in conjunction, along with any client specified (e.g. subscription) filters. As request transformations are connected to (link) authorisation rules, only one request transformation applies per link.

In the appropriate context, a number of event transformation rules might apply to an event. In this situation, the transformation functions are executed in parallel, in that each function takes as input (a copy of) the original event, the output of which moves to the next stage of processing. An output event is not subject to further transforma-

[7] As opposed to the *post-matching* [29] application of policy.
[8] Our implementation only permits a broker to forward an advertisement to those that have the possibility of subscribing to the event type, as defined by its authorisation rules.

tion functions at the same enforcement point. This avoids complex transformation loops that can be difficult to reason about, particularly when the output is of the same type. An event transformation function is executed only once per event at an enforcement point, regardless of the number of policies causing the function to execute. This is to avoid duplicates resulting from multiple function invocations. The original (input) event will not propagate if any (applied) transformation is consumable.

5.4 Policy Conflict

Often it is appropriate that multiple policies apply at an enforcement point, however there will be situations in which policies *conflict*, in that they are incompatible. A common example of conflict concerns specialisation or exception, e.g. where a rule concerning a specific patient should override that for patients generally. It is argued that application-level conflict resolution is often better addressed by careful policy (re)authoring, rather than automated resolution [5]. We do not attempt to resolve conflicts automatically, as doing so in a complex environment such as healthcare is difficult and dangerous. Instead, we provide the tools for policy authors to detect possible conflicts, which they can ignore, redefine the policy-set, or specify a runtime resolution strategy. This is practicable given that IC policies are local to a broker, maintained by a single administrative domain.

Conflict Detection The first step in dealing with policy conflict is to determine the rules that have the potential to conflict. This is necessary given the declarative nature of our rules. Rule predicates can be statically compared with others to detect the situations in which multiple rules simultaneously apply. It is then for the policy author to decide whether the rules in fact conflict.

If rules *statically* overlap, then they all necessarily apply in certain conditions. Otherwise, it is some coincidental set of circumstances that causes the rules to apply. This is termed *dynamic* conflict, as there is the potential for the rules to apply depending on context. Such classification highlights the potential seriousness of a conflict, as statically conflicting rules are directed at similar targets. Other considerations can assist policy analysis, e.g. two notification transformations with the same output type may indicate a policy error, as the subscriber may be misled by receiving several (similar) events of the same type. There may also be other application/domain-specific considerations relevant to ranking a conflict, such as rules authored by the inexperienced.

Conflict Resolution An obvious method for handling conflict is to 'author-out' any issues by redefining the policy set. This involves detecting and presenting potential conflicts to a policy administrator who deals with resolution. There are strong arguments that this is preferable to automated strategies [5, 19]. However, re-definition may be inappropriate for certain classes of conflict, e.g. if the conflicts occur infrequently, or only in particular situations. As such, our model provides for the definition of constraints—ordering, overriding and explicit declarations of incompatibility—that instruct the system on how to enforce the policies at runtime. The constraints are defined as separate entities that refer specifically to the rules involved in the conflict. This provides certainty and visibility as to how the rules are combined. The enforcement process is depicted in Fig. 2.

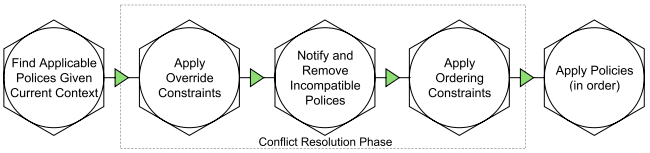


Figure 2: The process of policy enforcement.

6 IC-Database Integration

Our implementation integrates IC into *PostgreSQL-PS* [10, 28]: an extended PostgreSQL[9] database system that includes type-based pub/sub functionality. Databases are an integral component of large-scale infrastructure. A coupled database-messaging infrastructure has advantages over separate systems: a single interface and common type system, simplified replication, transactional delivery and improved performance (see [10] for discussion). Such an environment is particularly attractive for implementing IC functionality. An integrated pub/sub-database system enables IC predicates access to a rich representation of state: anything accessible from the broker, including event details, stored data, type schemata and stored procedures (that may call external services, e.g. shared credential services [22]). This provides much contextual information on which to base disclosure decisions. Persistence is facilitated, important not only for maintaining current (business/workflow) state, but also because the transfer of sensitive information often requires audit. Further, we can leverage existing database functionality, such as transactions, stored procedures and active rules to realise IC functionality and manage contextual change.

PostgreSQL-PS takes an advertisement-based approach to routing, where each database instance is a pub/sub broker. The filter model is highly expressive, allowing predicates to reference anything accessible by the query engine of the database-broker (subject to permissions). A store-and-forward approach is used for reliable delivery. We represent policy in the same format as events, so that rules may be defined using the same messaging infrastructure.

6.1 Hook Rules

We implement IC as a data control layer that operates above the pub/sub-database system. IC interacts with the pub/sub system through *hook rules* [24], which are active (ECA) rules, somewhat like triggers, that execute at specific points of the pub/sub process. Hook rules are distinct from IC rules, in that they are used to support the implementation of IC functionality.[10] A hook rule provides a *callback* mechanism, which we use so that the pub/sub layer executes a function in the appropriate circumstances to effect some data control operation(s). Hook rules are transformational: here a data structure containing relevant data/state is passed to a function in the data control layer, the output of which is returned to the pub/sub layer on which processing continues. Hook rules may be conditional, defined with (SQL) predicates that must hold for the rule to be enforced. Table 1 provides an overview of the hook rules used to realise IC and Fig. 3 illustrates their points of enforcement.

[9] `http://www.postgresql.org`

[10] There is not always a direct, *one-to-one* mapping between hook rules and IC rules. This is because hook rules are distinct, merely providing the callback (ECA) functionality for realising IC.

Table 1: Hook rule types and their associated description.

Rule Type	Input/Output	Purpose
CONNECTION VALIDATOR	Connection Details	Validates and authorises a connection. Establishes advertisement forwarding restrictions for brokers.
REQUEST VALIDATOR	Request	Authorises and processes the incoming request.
LINK ADV PROCESSOR	Advertisement	Executed when an advertisement is received through a link. Establishes subscription forwarding restrictions.
REQUEST TRANSFORM	Request	Modifies the request for delivery to a specific broker.
RESOLVE TRANSFORMS	Applicable Rules	Resolves any conflicts between applicable transformations at an interaction point.
EVENT TRANSFORM	Event	Executes the transformation function on the event.

The *validator* hooks use a function to determine whether the request/connection is authorised by any rules (after conflict resolution). For an authorised request, the *request validator* function loads the appropriate imposed conditions as channel filters, and creates the *event transform* hooks relevant to the newly created channel. Given that the applicable event transformations (hooks) are determined at runtime—evaluated in the context of an event—separate conflict resolution (*resolve transform*) hooks are required. The *request validator* function also creates the active rules to cause request re-evaluation on a change in monitored condition.

The advertisement restrictions relevant to a remote broker are loaded on link (connection) establishment by the *connection validator*. This is because advertisements are forwarded immediately after connection, or on receipt from another broker. Subscription forwarding restrictions, however, are loaded in response to the receipt of an advertisement from a broker. This is enabled by the *link adv processor* hook.

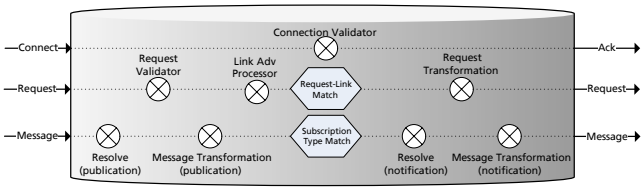


Figure 3: Hook rule enforcement points.

6.2 Audit

An IC implementation should facilitate audit by each broker recording data surrounding the operations that it undertakes. Such information is important as it provides *evidence* that one is meeting their data management responsibilities. In this way, audit brings about accountability. This information can also help to identify system or higher-level (organisational) issues, and may assist in discovering policy errors.

The database-broker environment greatly simplifies the implementation of audit processes. We use triggers to record in dedicated (unmodifiable) tables the details of pub/sub and IC operations including: request processing, connection/link state, event receipt/delivery, and the transformations and filters applied. This includes recording information of the applicable policies and those enforced (including details of conflict resolution), the clients/brokers involved and details of changes in context. Internal identifiers are assigned to processes to enable tracking of flows throughout the system, i.e. from receipt of an event to delivery (of it and its derivatives) to a number of clients. As audit data is persisted in tables, it can be queried using existing SQL constructs, and archived/managed in a similar manner to other data.

Audit may also be active, where the existing infrastructure is used to raise alerts (create new events) in particular situations; for example, to inform of a processing exceptions, policy incompatibilities or some higher-level concern.

7 Case Study: A Prescribing Scenario

In this section, we describe the use of IC in a scenario based on real-world health requirements. The focus of IC is to allow broker policy to set the boundaries for controlled dissemination. This is in contrast to other pub/sub security approaches, which rather than being tied to application-level semantics, focus on providing specific security mechanisms (see §3): security enforced at network edges ignores intermediaries, encryption-based schemes are inflexible and raise accountability concerns. Existing systems do not aim to provide a unified link between application concerns and granular, contextually-aware controls over connection/link establishment, event filtration and security transformations.

The enforcement of security policy necessarily imposes a processing overhead. However, any such overhead should be considered alongside whether it offers savings in data transmission and/or processing throughout the network. In this section, we apply IC to a prescribing scenario, presenting the rules necessary for restricting the data-flows of scenarios based on real-world requirements and to demonstrate that IC can in fact improve net performance of a pub/sub system.

Prescriptions are an integral part of care provision. Here we present two different implementations of a prescribing scenario, one from the perspective of a single *Surgery* broker that supports local and homecare services, the other considering a distributed broker network of a *Hospital*. Both have much the same restrictions. The dataflows of this scenario are presented in Fig. 4. This scenario, although simple, illustrates the need to control data flows to parties involved in the care process.

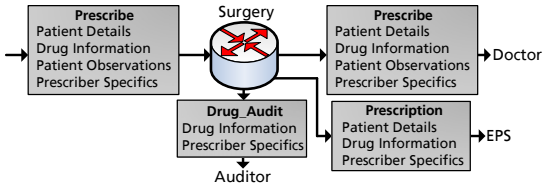


Figure 4: Data flows for `prescribe` events.

In providing care, nurses record symptoms and notes concerning patients' well being, and may also prescribe medication, though only for drugs within their competence [8]. As part of treatment, a nurse might publish a `prescribe` event[11] that encapsulates all data concerning a prescribing

[11]The notion of event-driven workflow is well established in the NHS, see §2.3.

incident, including the reasons for the medications. This information requires storage in the provider's care database(s), and must be transmitted to other domains for audit and dispensing purposes. Doctors access information from their surgery. A doctor may request notification of when drugs are prescribed for some patients, as this may indicate a situation of concern. An authorisation rule allows a doctor to subscribe to `prescribe` events for patients that they treat:[12] (`subscription`, `prescribe`, {`doctor`}, {`treatsPatient`(`user`, `att.patientid`)}) $\in RT \times \tau \times \mathbb{P}(\mathcal{C}) \times \mathbb{P}(\mathcal{E})$.[13]

Prescriptions must flow to the *Electronic Prescription Service* (EPS), a domain that operates as a clearing-house, assisting pharmacies in dispensing and reimbursement. The EPS requires prescription information, but *not* the reasons (notes or observations) for the medication. A transformation rule converts all `prescribe` events into a `prescription`, which involves copying relevant details from the `prescribe` events, and augmenting them with extra patient information, such as the patient's address and date-of-birth, and their surgery and/or hospital details. The rule is defined as: (`publication`, `prescribe`, $\emptyset$, $\emptyset$, `toPrescription`, `prescription`, $\perp$) $\in IP \times \tau \times \mathbb{P}(\mathcal{C}) \times \mathbb{P}(\mathcal{G}) \times F \times \tau \times D$, where an empty predicate set is interpreted as a need to always apply the rule. An authorisation rule allows the EPS to subscribe to all `prescription` events.

The auditor must be informed when certain 'controlled drugs' are prescribed. The audit is prescriber focused, thus the auditor will only receive patient details in exceptional circumstances. To balance confidentiality with the duty to inform the auditor, a `drug_audit` event is created through a transformation function that removes sensitive information by filtering various fields from prescriptions for controlled drugs. However, when a prescriber is under suspicion, the auditor may receive their `prescribe` events to provide detailed information to assist the investigation. This is because it is in the interests of public safety to ensure that the prescriber is acting appropriately. This transmission occurs in line with patient consent, which might be obtained when providing care, or perhaps in reference to the individual investigation. This is enforced by an imposed condition rule: (`subscription`, `prescribe`, {`auditor`}, {`auditConsent`(`msg.patientid`)}, {`investigating`(`msg.prescriberid`), $\perp$}) $\in IP \times \tau \times \mathbb{P}(\mathcal{C}) \times \mathbb{P}(\mathcal{E}) \times \mathbb{P}(\mathcal{R}) \times H$.

7.1 Distributed Scenario

We extend the scenario to a distributed environment, where the hospital (an administrative domain) must also control the dataflows throughout its own broker network. Fig. 5 depicts the topology, where a number of nurses are assigned to work the wards. Each *ward* has its own broker. The *central* (ward) broker manages all ward information, receiving (subscribing to) all events from the wards. The hospital *dispenser* acts as the local pharmacy, and thus is responsible for distributing information to the EPS, and forwarding audit events for controlled drugs. The *local trust* broker (exclusively) manages the information flows to local health institutions. Here there are two *surgeries* who have doctors subscribing to information on their patients, as well as the regional *auditor* who receives all `drug_audit` events and the `prescribe` events issued by nurses under investigation. The hospital is responsible for sending the appropriate patient information to the surgeries, who must then pass on the information to the relevant doctors. For both examples, the appropriate principal and request authorisation rules are defined to enable the described connections and data flows.

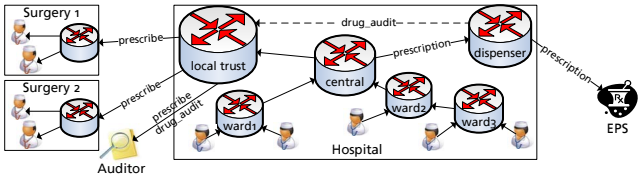


Figure 5: The distributed prescribing environment.

7.2 Experimental Setup

Each broker consisted of an integrated IC-PostgreSQL-PS instance running on its own Intel Core 2 Duo 2.4Ghz CPU machine with 2GiB of RAM. The clients were distributed amongst a number of machines on a different subnet from the brokers. We present the mean values over 10 trials. The environment contains 1,000 active patient records, where four doctors manage 250 patients each. Five nurses attend to the patients, raising `prescribe` events for the drugs they prescribe. Each doctor subscribes to `prescribe` events pertaining to the top 25 (10%) most critically ill patients that they treat. In the trials, the event size is fixed for each event type, as in practice many attributes are fixed length identifiers that refer to treatments, drugs and patients.

The workload involves each nurse publishing 1,000 events, 200 of which concern patients to which doctors are subscribed.[14] All of the `prescribe` events are transformed into `prescription`s, and those for controlled medications into a `drug_audit` event. The auditor receives all `prescribe` messages published by the single nurse that is under investigation (all patients have consented). To better gauge the processing overheads of IC, it is necessary to compare that to an implementation of *vanilla* pub/sub; i.e. one lacking restriction functionality.[15] Without IC, the publishers become the sole source of information, who must deal with confidentiality concerns by publishing separate events for each level of visibility. This involves publishing two or three events for each prescription: `prescribe` relevant for the doctors/surgery/hospital/auditor, `prescription` as relevant for the EPS, and if the drug is controlled, a `drug_audit` event. We implement the vanilla pub/sub scenario using the same database-broker infrastructure,[16] to ensure that the information is reliably recorded, audited, processed and delivered. However, the vanilla implementation is unrestricted—i.e. without any IC rules. To enable comparison we assume

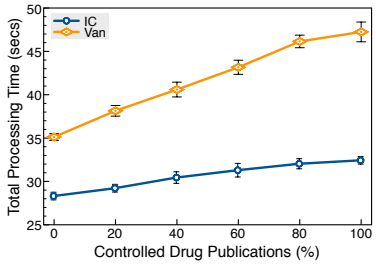
[12]For want of space, we only present only the most pertinent rules to serve as examples. The complete set of rules governing this scenario can be found in [20].

[13]Note that `att` refers to permission attribute values, `msg` to event content, and `user` to the client's unique idenitifer.

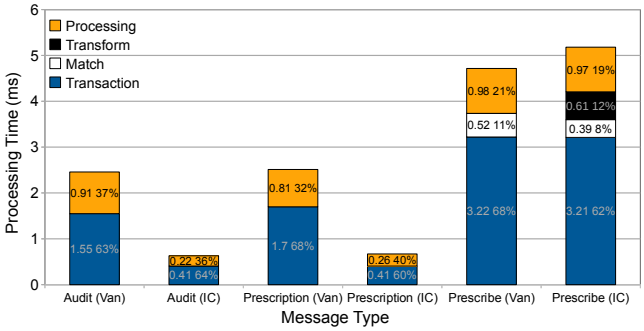
[14]In practice prescriptions occur relatively infrequently, i.e. in the order of tens per day. However, a large number of publications were used in this scenario to provide a *general* indication of performance, accounting for database optimisations such as write buffers and caching, and to ensure that nurses publish simultaneously to interleave processing.

[15]To reiterate, we do not compare IC to other pub/sub systems as we are unaware of any that provide a complete set of comparable controls. Instead, here the comparison is to illustrate that the overheads of policy enforcement must be considered with respect to overall system performance.

[16]Note that [10] compares the performance of (vanilla) PostgreSQL-PS to other pub/sub implementations.



(a) Workload processing comparison



(b) Processing time per event type

Figure 6: Single broker performance results.

that the vanilla subscribers are honest and cooperative, issuing filters in line with the restrictions otherwise required by the scenario.

7.3 Single-Broker Comparison

Prescriptions for controlled drugs require more processing to create and deliver `drug_audit` events. Therefore, this scenario consisted of five workloads, each with a different percentage of controlled drug prescriptions. Fig. 6(a) shows that the overall processing time for the vanilla implementation is significantly greater than that of the IC model for each workload. This is because the vanilla approach involves a broker receiving 2–3 times the number of publications to account for varying levels of data visibility, where each publication is subject to all event processing operations.

To give a clearer indication of the overheads, Fig. 6(b) presents a breakdown of processing time per event type. Transactions are necessary for reliable processing of events; however, transactions also impose an overhead. Here, the transactional overheads (begin/commit) of event processing outweigh those imposed by the transformations.

The figure also shows that the `drug_audit` and `prescription` events for the IC approach involve a significantly lower processing time than their vanilla counterparts, due to the fact that the transformed events are subject to fewer messaging operations. The *match* category refers to the time the query engine spends in evaluating subscription filters. This is negligible where a subscription is filterless (i.e. match all), as for `prescription` and `drug_audit` subscriptions.

`Prescribe` events take greater processing effort as they are subject to more subscriptions, and thus involve more processing and transactions. The IC approach is slower at processing a single `prescribe` event due to the overheads imposed by the transformations; however, the difference is comparatively small with respect to the overall processing time for the event type. Considering the total processing time for a prescription *incident*, i.e. accounting for all information flows for the action of prescribing, the IC implementation is shown to be more efficient, taking 6.48ms compared to 9.82ms for the vanilla approach.

7.4 Distributed Broker Scenario

Again, we compare a vanilla and IC implementation, however there is a slight variation in the routing paths between the approaches.[17] As the dispenser is responsible for all pharmaceutical data, a transformation enables the central ward to pass the prescription events to the dispenser, who forwards them to the EPS service. The dispenser is also responsible for producing the audit events. In the vanilla approach the routing paths are determined solely by (unrestricted) advertisement/subscription propagation. This means that `drug_audit` events are produced by the nurses (publishers) rather than the dispenser, and thus flow directly from the central broker to the local trust broker for distribution to the auditor. This experiment uses the same workload, except the number of controlled drugs publications is fixed at 40%. The nurse in Ward3 is under investigation.

We observed that for the workload, IC resulted in a ∼36% reduction in events transmitted, and ∼33% reduction in processing time. To show the effects of distribution, Fig. 7 presents a breakdown of processing for each broker in the topology. Each IC broker incurs equal, or significantly less overhead than its vanilla counterpart, except for the dispenser whose auditing responsibilities require additional operations in the IC implementation.

Generally, pub/sub models aim to reduce multiplicative event fan-out by pushing subscriptions as close to the publisher as possible. Similarly, transformations can improve distribution by routing a single copy of an event as far as possible before transforming it as relevant to the subscriber. This is reflected in Fig. 7(d),[18] where the prescription and audit events resulting from a transformation travel only one hop, as opposed to the vanilla implementation where events travel the whole path from the publisher to the subscriber. The other subfigures show that this reduces the event count, byte count and processing time. Where transformations do not impact on propagation, i.e. in the delivery of `prescribe` events, the results are similar for both approaches.

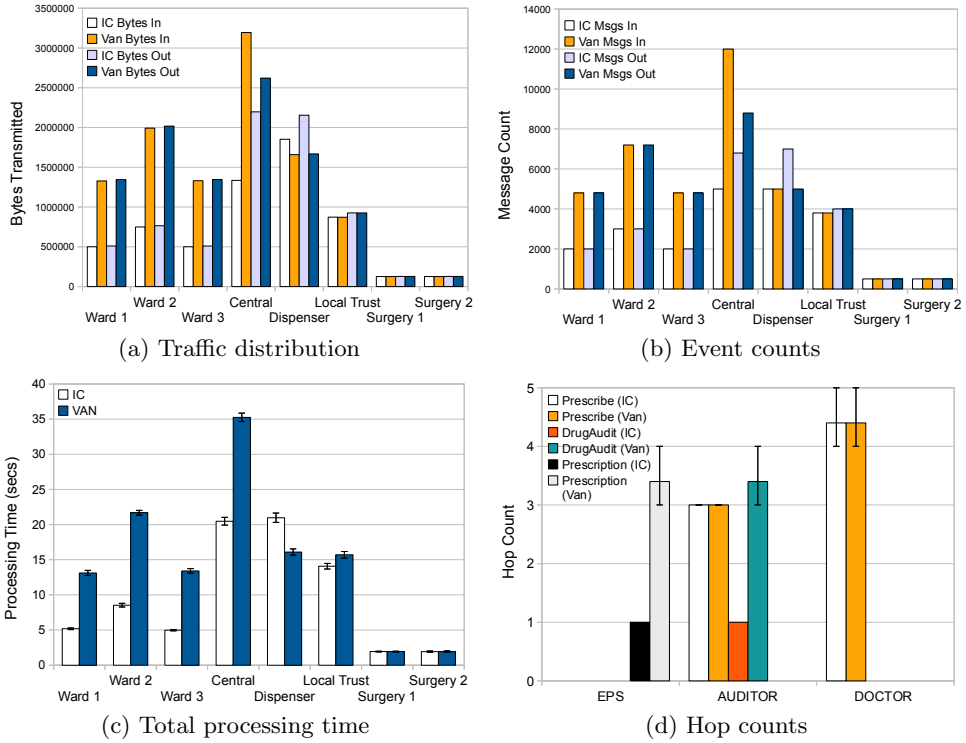
7.5 Scenario Discussion

As IC involves policy enforcement and event processing, it is useful to consider its overheads. Although our focus is on security rather than performance, this section demonstrates that the overheads brought by enforcing security policies should be considered with respect to the overall savings (in transmission/processing) throughout the infrastructure.

We observe that a significant proportion of processing time relates to transactions. We argue that transactions are necessary in an implementation because sensitive information is typically important, and thus must be reliably stored, audited, processed and delivered. It follows that transactional and storage overhead is likely to be incurred regardless of the implementation. In this scenario, transformation functions, despite involving queries on stored data, did not greatly impact overall performance as the operations were comparatively less expensive than the transactions.

[17] This is indicated in Fig. 5 by the dashed line for the `drug_audit` event.

[18] The bars in Fig. 7(d) represent the mean, and the errorbars the maximum and minimum hops travelled.



(a) Traffic distribution (b) Event counts

(c) Total processing time (d) Hop counts

Figure 7: Performance of the IC and Vanilla implementations in the distributed scenario.

Intuitively, processing overheads can be reduced given that restrictions and transformations can limit information flows, thereby reducing event fan-out. This, of course, depends on the nature of the scenario; clearly, the results will vary depending on the complexity of the operations (e.g. transformation functions), and the representation of context. These are application-level concerns, where performance characteristics depend on the specific requirements of the scenario and the implementation environment. That said, our experiments indicate the scalability of IC, where policy enforcement does not necessarily degrade performance, but in certain circumstances improves efficiency, particularly when considering the overall workload (net performance).

Although a vanilla implementation is unrestricted, and thus useful for performance comparisons as it avoids policy overheads, it is important to remember the differences regarding security. A vanilla approach assumes that clients are trusted i) to publish events with the appropriate level(s) of visibility for all potential recipients, and ii) to (only) subscribe to information for which they are authorised. Information is routed through the brokers in an uncontrolled manner, thus brokers are also trusted to behave appropriately. This is unsuitable in situations where data is sensitive, as even without malice, there are negligence and curiosity concerns. Further, the vanilla approach is insufficient even in an environment of overarching responsibility, as accountability is diminished. As described, other pub/sub security models lack a comparable set of controls, meaning that they would address only a subset of these issues.

8 Concluding Remarks

IC rules concern transmission (only) to the next hop. This is because a broker is not qualified to make decisions regarding the event flows of other brokers, especially those in other domains. To do so is impractical, as it requires information of the local processes, context and policies of each broker along the dissemination path. Further, enforcing disclosure policy at the subscriber's broker fails to protect inter-broker communications, while enforcing at the publisher's broker in the worst cases causes a separate event to be routed for each subscription. Aside from policy management and scalability concerns, such approaches complicate notions of responsibility and accountability.

Typically in practice, databases form an essential component of large-scale infrastructure. In addition to the fact that sensitive information not only requires transmission, but also storage and audit,[19] we have integrated IC into a database system to show that such controls can both operate with, and leverage from, technology already commonplace.

Local control underpins the model of responsibility. If domains manage their technical infrastructure, it seems reasonable to couple the control mechanisms with the data they protect. General pub/sub is by itself inappropriate for environments where information is sensitive, thus some security-related overheads will be imposed regardless of the implementation. As future work, we hope that gathering further experience in real deployments will allow us to make quantified statements of scalability: data is required concerning the number of clients, event types and policy rules, event load, rule complexity, connection churn, and other variables and constraints. Of course, this is a moving target. That said, although simple, our case study based on real-world requirements indicates that an IC implementation is possible by extending existing technology, and that the model can encapsulate the data flow requirements of real

[19] See [21] for a discussion of security issues concerning the replay of historical events.

care processes. While policy enforcement necessarily introduces overheads, we have shown that these must be considered with respect to any savings in overall network activity.

IC essentially overlays a point-to-point security model over a distributed pub/sub service. This is a direct product of the responsibility model (local-control). Domains process and share information for a particular purpose, meaning that connections and the associated security constraints form naturally. A domain maintains a sharing policy that it implemets within the brokers it controls. In this way, a broker's policy store merely reflects a domain's trust and security concerns. Local enforcement can lead to *Chinese-Whispers* effects, where an event/request changes or is filtered as it moves through the network. This is the result of the responsibility model. *End-to-end* concerns could arise if policy is incorrectly formulated (at the high-level) or specified (at the system-level); though clearly policy issues will affect any approach that enforces application-level considerations. IC rules enable the definition of the *boundaries* for communication, allowing events to flow freely where authorised. The purpose of IC is to allow the advantages of a scalable, push-based, subscription-oriented distribution paradigm safely in environments of sensitive information, by allowing security constraints to be imposed where necessary.

Ultimately, information protection is not just a technical issue. There are legal requirements and social pressures regarding the use and management of sensitive information. Technical infrastructure must facilitate data sharing, storage and protection, in line with higher-level concerns. IC is designed specifically for environments of federated policy, where those active in the environment are responsible for the information they access, generate and manage. It is novel in that it brings the notion of local-control to the pub/sub middleware, so that those responsible for information encode rules in their brokers to restrict data flows where necessary. The model is unique as it enables control in situations where wide-area notification services are required across administrative boundaries and/or where security policy dictates that data visibility must vary between clients and/or network components (brokers). It is not a security *panacea*—indeed, it is unlikely such a thing exists—but instead provides a mechanism for control, giving those managing data the ability to meet their data sharing and protection obligations. Local control and responsibility brings accountability, which is necessary for the protection of sensitive information.

Acknowledgments

The authors acknowledge the Technology Strategy Board (TS/H000062/1) and EPSRC (RG55622) for their support.

9 References

[1] J. Bacon, D. M. Eyers, J. Singh, and P. R. Pietzuch. Access control in publish/subscribe systems. In *DEBS '08*, pages 23–34. ACM, 2008.

[2] A. Belokosztolszki, D. M. Eyers, P. R. Pietzuch, J. Bacon, and K. Moody. Role-based access control for publish/subscribe middleware architectures. In *DEBS '03*, pages 1–8. ACM, 2003.

[3] Bolton Research Group. Patients' knowledge and expectations of confidentiality in primary health care: a quantitative study. *British Journal of General Practice*, 50:901–902, 2000.

[4] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[5] R. Chadha. A cautionary note about policy conflict resolution. In *MILCOM*, pages 1–8. IEEE, 2006.

[6] L. Darzi. *High quality care for all: NHS Next Stage Review*. Department of Health, 2008.

[7] Department of Health. Confidentiality: NHS Code of Practice, 2003.

[8] Department of Health. Safer management of Controlled Drugs, 2007.

[9] T. Dierks and C. Allen. *The TLS Protocol (RFC 2246)*. Internet Engineering Task Force (IETF), 1999.

[10] D. M. Eyers, L. Vargas, J. Singh, K. Moody, and J. Bacon. Relational database support for event-based middleware functionality. In *DEBS '10*, pages 160–171. ACM, 2010.

[11] L. Fiege. *Visibility in Event-Based Systems*. PhD thesis, TU Darmstadt, 2004.

[12] L. Fiege, A. Zeidler, A. Buchmann, R. Kilian-Kehr, and G. Muehl. Security aspects in publish/subscribe systems. In *DEBS '04*, pages 44–49. IEEE, 2004.

[13] H. Khurana. Scalable security and accounting services for content-based publish/subscribe systems. In *SAC '05*, pages 801–807. ACM, 2005.

[14] Z. Miklós. Towards an access control mechanism for wide-area publish/subscribe systems. In *ICDCSW '02*, pages 516–524. IEEE, 2002.

[15] NHS Care Record Development Board. The care record guarantee—our guarantee for NHS Care Records in England, 2009.

[16] NHS Information Authority. Share with Care! People's views on consent confidentiality of patient information, 2002.

[17] L. Opyrchal and A. Prakash. Secure distribution of events in content-based publish subscribe systems. In *SSYM'01*, pages 21–21. USENIX, 2001.

[18] L. I. Pesonen, D. M. Eyers, and J. Bacon. Access control in decentralised publish/subscribe systems. *Journal of Networks*, 2(2):57–67, 2007.

[19] C.-C. Shu, E. Y. Yang, and A. E. Arenas. Detecting conflicts in ABAC policies with rule-reduction and binary-search techniques. In *Policy '09*, pages 182–185. IEEE, 2009.

[20] J. Singh. *Controlling the dissemination and disclosure of healthcare events*. PhD thesis, University of Cambridge, 2010.

[21] J. Singh, D. M. Eyers, and J. Bacon. Controlling historical information dissemination in publish/subscribe. In *MidSec '08*, pages 34–39. ACM, 2008.

[22] J. Singh, D. M. Eyers, and J. Bacon. Credential management in event-driven healthcare systems. In *Middleware '08 Companion*, pages 48–53. ACM, 2008.

[23] J. Singh, L. Vargas, and J. Bacon. A model for controlling data flow in distributed healthcare environments. In *Pervasive Health '08*, pages 188–191. IEEE, 2008.

[24] J. Singh, L. Vargas, J. Bacon, and K. Moody. Policy-Based Information Sharing in Publish/Subscribe Middleware. In *Policy '08*, pages 137–144. IEEE, 2008.

[25] M. Srivatsa and L. Liu. Secure event dissemination in publish-subscribe networks. In *ICDCS '07*, page 22. IEEE, 2007.

[26] M. A. Stone, S. A. Redsell, J. T. Ling, and A. D. Hay. Sharing patient data: competing demands of privacy, trust and research in primary care. *British Journal of General Practice*, 55:783–789, 2005.

[27] D. Sturman, G. Banavar, and R. Strom. Reflection in the Gryphon message brokering system. In *Reflection Workshop, OOPSLA '08*, 2008.

[28] L. Vargas, J. Bacon, and K. Moody. Event-Driven Database Information Sharing. In *BNCOD '08*, pages 113–125. Springer, 2008.

[29] A. Wun and H.-A. Jacobsen. A policy management framework for content-based publish/subscribe. In *Middleware '07*, pages 368–388. Springer, 2007.

[30] Y. Zhao and D. C. Sturman. Dynamic access control in a content-based publish/subscribe system with delivery guarantees. In *ICDCS '06*, pages 60–68. IEEE, 2006.