A Scalable Approach for the Secure and Authorized Tracking of the Availability of Entities in Distributed Systems

Shrideep Pallickara, Jaliya Ekanayake and Geoffrey Fox

Indiana University
Community Grids Lab
Bloomington, IN 47404 USA
{spallick, jekanaya, gcf}@indiana.edu

Abstract

As the scale and proliferation of distributed applications continues to increase a need often arises to track the availability of entities that comprise the distributed system. An entity that is part of such a distributed system could be a resource, a service that provides a set of exposed capabilities, an application or a user. In this paper we present a transport-independent scheme for tracking the availability of entities in distributed systems. The scheme enforces the authorized generation and consumption of traces (encapsulating entity availability). The scheme also facilitates the secure distribution of traces while coping with some classes of denial of service attacks.

1. Introduction

Over the past decade we have witnessed the proliferation of distributed applications. This is fuelled in part by advances in networking technology combined with the advent of cheaper and ever more powerful devices. An entity that is part of such a distributed system could be a resource, a service that provides a set of exposed capabilities, an application or a user.

Interactions, such as control messages, protocol handshakes, actions and data interchange, between entities that are part of a distributed system are predicated on their availability. For example, an application may be interested in the availability of a resource at all times. Similarly, a user would be interested in the availability of a given service. Entities thus need to be aware of each other's availability at regular intervals. In several cases remedial actions are taken in response to the failure/ unavailability of given entity.

1-4244-0910-1/07/$20.00 © 2007 IEEE.

Before we proceed further, an explanation of the terms used in this paper is in order. An entity whose availability is being probed is referred to as a *traced* entity. The entities initiating a probe are referred to as *tracker*s. The process of probing, and subsequently becoming aware of, the availability of an entity is referred to as *tracing*. The different states corresponding to a traced entity is referred to as its traces.

There are two approaches to tracking the availability of entities – push and pull. In the push model the traced entity issues messages to the trackers at regular intervals. Receipt of such messages at the trackers signifies the availability of the entity; the lack of receipt indicates potential problems. A tracker may deem a traced entity to have failed if it does not receive such messages for a prolonged duration of time. In the pull model the trackers ping the traced entity at regular intervals. Responses, or the lack thereof, from the traced entity form the basis for determining whether a traced entity is available or not. In the push model the complexity at the traced entity is higher since it needs to send messages to every tracker at regular intervals. In the pull model, on the other hand, the complexity at the tracker is higher since it needs to keep track of every traced entity.

In the simplest scheme, every entity would issue messages at regular intervals when they are present within the system. If there are N entities within the system, with each of them issuing one message at regular intervals, every entity within the system receives (N-1) messages. If every entity issues one such message per second, there would be Nx(N-1) messages within the system every second. As the scale of the system increases, the complexity and costs associated with this approach increases, and the limits of this approach become apparent since every entity within the system would be inundated with messages.

There are three other critical issues that need to be addressed in these settings. First, in large distributed systems the transport protocols over which entities initiate communications is large. If an entity is required to cope with this in its message exchanges with other entities, the complexity at a given entity increases substantially. Second, only authorized entities should be part of the tracing process. The third issue is that of security. Here, message exchanges would need to be secured so that the information contained therein is not used to launch denial of service attacks.

In this paper, we present our solution to this problem. The characteristics of this solution are enumerated below.

1. Number of Messages: Messages are issued only if there are entities interested in tracking an entity. Additionally, tracking entities may register only for change notifications; here, traces will be issued only if there is a change in the status of the traced entity.
2. Transport Independent: Entities do not have to deal with the complexity of the underlying transports.
3. Authorization: Only authorized entities would be allowed to track an entity.
4. Security: Message exchanges, related to availability, are secured cryptographically. Only entities in possession of the appropriate security keys can decipher the message contents.
5. Denial of Service attacks: The scheme also copes with a few types of denial of service attacks.

The remainder of this paper is organized as follows. In section 2 we provide an overview of the publish/subscribe systems and the NaradaBrokering system which is based on this paradigm. In section 3 we outline our tracking. Sections 4 and 5 deal with the authorization and security issues related to this scheme. In section 6 we present our performance benchmarks. Section 7 surveys the related work in the area. Finally, in section 8 we outline our conclusions and future work.

2. NaradaBrokering Overview

We have implemented our scheme in the context of the NaradaBrokering substrate [1-3], which is based on the publish/subscribe paradigm (discussed in section 2.1). In NaradaBrokering this middleware is itself, a distributed infrastructure comprising a set of cooperating router nodes known as *broker*s. A broker performs the routing function by routing content along to other brokers within the broker network. Producers and consumers don't interact directly with each other. Entities are connected to one of the brokers within the broker network, an entity uses this broker, which it is connected to, to funnel messages to the broker network and from thereon to other registered consumers of that message. All messages contain topic information within them; this topic information forms the basis of routing of messages. When a broker receives a message from a producer, it checks to see the message should be routed to any of the consumers that are connected to it; this broker will then proceed to route the message to other brokers within the network that have consumers interested in consuming this message.

2.1 Publish/Subscribe Systems

The publish/subscribe paradigm is a powerful one, in which there is a clear decoupling of the message producer and consumer roles that interacting entities/services might have. The routing of messages from the publisher to the subscriber is within the purview of the message oriented middleware (MoM), which is responsible for routing the right content from the producer to the right consumers. In publish/subscribe systems a subscriber registers its interest in messages by subscribing to *topics*. In its simplest form these topics are typically "/" separated Strings, for example StockQuotes/Companies/Adobe. When a publisher issues messages on a specific topic the middleware substrate routes the messages to all, and only those, subscribers that have registered interest in the topic.

2.2 The Topic Discovery Scheme

Interactions between entities in publish/subscribe systems are predicated on the knowledge of the topic that will be used for communications; the publisher will publish over this topic while the subscriber registers a subscription to this topic. The topic discovery and creation scheme [2] in NaradaBrokering facilitates the creation, advertisement and authorized discovery of topics by entities within the system. The discovery process is a distributed process and is resilient to failures that might take place within the system. Topic creators can advertise their topics and can also enforce constraints related to the discovery of these topics. Specifically, a topic creator may require the presentation of appropriate credentials (a X.501 security certificate) prior to being able to discover a topic. This discovery scheme provides a solution for issues such as

1. Provenance –- The system can verify easily the owner of a certain topic.
2. Secure discovery –- A topic owner can restrict the discovery of a topic only to authorized entities or those that possess the valid credentials.

These capabilities are provided by specialized nodes – Topic Discovery Nodes (TDNs) – within the system. Since a given topic advertisement will be stored at multiple TDN nodes, this scheme sustains the loss of TDN nodes due to failures or downtimes. Additional details regarding this scheme can be found in Ref [2].

3. The Tracing Scheme

In our scheme we use a combination of the push and pull styles described in section 1.0. In addition to the traced entity and the trackers that are involved in the tracing there is an additional component: the broker which the traced entity is connected to. This broker is responsible for polling – the pull part – the traced entity at regular intervals and for generating – the push part – traces for the traced entity.

We leverage the pub/sub style of communications in the exchange of traces between the entities: trace information is encapsulated in messages that have topic information associated with them. This trace information includes information related to the traced entity's state, state transitions, network metrics and usage statistics. Not all trackers would be interested in all the traces related to a traced entity. The number, frequency and volume of traces received at a tracker vary with the type of trace information that it is interested in. To facilitate greater selectivity in the trace information at any given tracker, traces related to an entity are issued over different topics. Thus a tracker may register to receive all traces or only state transitions related to a traced entity.

We impose restrictions on who is authorized to discover topics related to trace information. Furthermore, we also impose restrictions on the actions, either publish or subscribe, that are allowed over these topics. Messages encapsulate trace information need to also unambiguously establish the source of the trace and the authorization to issue this trace information. We also incorporate strategies to cryptographically secure individual traces and the secure distribution of keys to decipher the encrypted contents.

3.1 Trace topic

In our scheme an entity will be traced only if it specifically issues a request for this. There is a sequence of actions that need to be taken by an entity before it can be traced. An entity must first create a topic corresponding to its availability tracing. To do this, the entity must create a topic creation request which is sent to the TDN which is responsible for the generation of the trace-topic. This topic creation request includes four key components. First, the entity includes its credentials – a X.509 certificate – that is used by the TDN to establish provenance for the trace topic that it would create.

Second, the entity specifies the descriptor to be associated with the topic. During topic discovery, the queries are evaluated against the topic descriptors associated with topics stored at the TDN. The topic discovery scheme provides support for variety of query formats, for purposes of simplicity to enable discovery of trace topics, a traced entity specifies the topic descriptor for the trace topic to be Availability/Traces/Entity-ID. Where Entity-ID corresponds to the identifier associated with the entity in question. This topic descriptor also ensures that trackers can construct appropriate discovery queries to discover the trace topic simply by utilizing the Entity-ID of the traced entity.

Third, a traced entity must also specify discovery restrictions that should be associated with the trace topic. These discovery restrictions specify who is authorized to discover the trace topic associated with the entity's availability. Discovery requests initiated by entities that have not been authorized to discover a given topic will be ignored by the TDN.

Finally, the topic creation request also specifies the lifetime associated with the trace topic. Lifetimes enable an entity to control the validity duration of the trace topic.

Upon receipt of this topic creation request containing the credentials, the topic descriptor, the discovery restrictions, and the topic lifetime the TDN generates a UUID which is trace topic associated with the entity. The UUID is a 128-bit identifier that is guaranteed to be unique in space and time. Generation of the UUID is done at the TDN so that no entity is able to claim some other entity's topic as its own. The TDN then proceeds to create a cryptographically signed topic advertisement that includes the newly created topic, along with the credentials, descriptors, discovery restrictions and lifetime. This advertisement establishes the ownership of the topic. This advertisement is stored at the various TDNs and is also routed back to the traced entity.

The TDN guarantees that discovery requests, targeted at discovering the trace topic associated with an entity, will not be satisfied unless these requests demonstrate possession of valid credentials that are conformant with the discovery restrictions specified in the original topic creation request.

Leveraging the Trace Topic. This trace topic is then used to construct *derivative* topics related to tracing the entity in question. The derivate topics are a combination of a static prefixes and suffixes that are combined with a given trace topic; an example of a derived topic is Constrained_Publish/Broker/Traces/TraceTopic/ChangeNotifications. These derivative topics are used to publish different *types* of trace information corresponding to the traced entity. Furthermore, in some cases actions (such as publishing) on a given derived topic still require the traced entity's authorization: this is typically delegated by the traced entity through the creation of cryptographic security token that demonstrates the delegation. Having multiple derived topics is also beneficial since it allows trackers to be selective about the trace information that they are interested in.

Constrained Topics. These are the equivalent of systems topics. The structure of the constrained topic reveals the constraints associated with the topic. These constraints correspond to limits on performed actions, proof of authorization for performing the action, security and propagation of these actions. The structure of a constrained topic is the following:

/Constrained/{Event Type}/{Constrainer}/{Allowed Actions }/{Distribution}/{Other "/" separated Suffixes}

We now include a discussion of each of these elements

{Constrained}: This elements takes only one value: Constrained. This keyword at the very beginning of a topic structure identifies that topic as a constrained topic.

{Event Type}: This element identifies the content of messages issued over this topic, default value: RealTime

{Constrainer}: This element identifies either the Broker (default) or the entity (in which case, the Entity-ID would be specified) that is allowed to perform the actions outlined in the {Allowed Actions} element.

{Allowed Actions}: This element describes the actions that can ONLY be performed by the constrainer. The values that this element can take include Publish, Subscribe or PublishSubscribe [default]. In the case of a PublishOnly constraint, entities are allowed to subscribe to messages issued over this topic. In the case of SubscribeOnly constraint, no entities are allowed to subscribe to the topic. Finally, in the case of PublishSubscribe no entities are authorized to perform any actions over the corresponding constrained topic: typically brokers would exchange administrative messages using such constrained topics.

{Distribution}: This element imposes restrictions pertaining to the distribution of allowed actions over this topic. The two values this element can take are `Suppress` and `Disseminate` (default). In the case of `Publish_Only` actions combined with `Suppress` distributions, messages issued by the constrainer are not distributed to other brokers within the broker network. Similarly, in the case of a `Subscribe_Only` action combined with `Suppress` distribution, the constrainer's subscriptions are not propagated within the broker network.

An example of a constrained topic is /Constrained/Traces/Broker/Subscribe_Only/Limited/Trace-Topic. In cases, where the elements do not appear in the constrained topic structure, default values for that element are assumed: thus /Constrained/Traces/Broker/PublishSubscribe/Limited and /Constrained/Traces/Limited are equivalent topics.

3.2 Registration of the traced entity

In the section we describe the steps taken by an entity interested in being traced to initiate the tracing process. Once an entity is ready to be traced, it creates the corresponding trace topic as specified in the previous section. The entity then proceeds to securely discover a valid broker within the broker network using the broker discovery scheme described in Ref [3]. The entity then needs to register with a broker and specify an interest in being traced. This trace registration message is issued over the following constrained topic /Constrained/Traces/Broker/Subscribe-Only/Registration/. In this registration message the traced entity includes the following:

1. Its identifier and credentials.
2. The trace topic advertisement, which establishes the trace topic provenance
3. The request identifier associated with the message. This is used to correlate any response that would be received for this message.
4. The entity also demonstrates possession of its credentials (and tamper evidence) by signing the message. The signing is done by computing the checksum for the message and encrypting this message digest with its private key.

Upon receipt of this message, the broker cryptographically verifies the message contents. First, the broker checks for proof of possession of the corresponding private key; here, we should be able to access decrypt the message signature with the entity's public key. If the decryption process is successful, we have access to the message digest. We then check the message digest for tamper-evidence; this is done by checking to see if the checksums/digest of the message content matches the one that was retrieved. If there is any error in the verification process, an error message is returned back to the entity.

If the verification process is successful, the broker then proceeds to generate a session identifier, and issue a successful registration response. This response includes:

- The request identifier contained in the original message.
- The newly generated session identifier.

The response message is encrypted with a randomly generated secret key, and this secret key is encrypted using the entity's public key. This way, only the entity in question is able to decipher the contents of the message.

The broker also proceeds to subscribe to the following topic. /Constrained/Traces/Broker/Subscribe-Only/Limited/Trace-Topic/SessionId. Upon receipt of the response message at the traced entity, the entity proceeds to subscribe to the following constrained topic /Constrained/Traces/Entity-ID/Subscribe-Only/Trace-Topic/SessionId.

3.3 Broker operations

The broker is responsible for failure detection of the traced entity and reporting the status of the traced entity to the trackers. The traces reported by the broker to the trackers, and summarized in Table 1, include:

- Constant updates on the continued availability of a traced resource

- Information about individual pings initiated by a broker
- Change in the status of a traced entity
- State transition information about a traced entity
- Information pertaining to network usage and the load at a given traced entity

Messages issued by a traced entity to the tracing broker are published over /Constrained/Traces/Broker/ Subscribe-Only/Limited/TraceTopic/SessionId, while messages issued by a tracking broker to the traced entity are issued over /Constrained/Traces/Entity-ID/ Subscribe-Only/Trace-Topic/SessionId.

Trace type	Description
INITIALIZING, RECOVERING, READY or SHUTDOWN	This is the *state information* reported by a traced entity to a broker.
FAILURE_SUSPICION, FAILED, DISCONNECT	Broker generated traces about an entity's *failure detection*
GUAGE_INTEREST	Trace to *gauge interest* among trackers in tracing an entity
JOIN, REVERTING_TO_ SILENT_MODE	Trace issued when an entity has *requested tracing*, and when it has decided to *disable tracing*.
ALLS_WELL	*Heartbeats* issued at regular intervals indicating that an entity is still active
LOAD_INFORMATION	Indicates the *load* information at an entity: CPU Info, Memory Usage and Workload
NETWORK_METRICS	Metrics about the *network realm* in which an entity operates: Loss rates, transit delay and bandwidth

Table 1: Traces reported by a broker to the trackers

Pings, Ping Responses and Network Metrics. A broker issues pings at regular intervals to the traced entity. Upon receipt of this ping message, the traced entity is expected to issue a ping response back to the broker. The ping message issued by a broker contains a monotonically increasing message number and the timestamp (at the broker) at which it was issued. A ping response associated with a ping must include both the message number and timestamp contained in the original ping. The message number allows a broker to keep track of message losses and out-of-order delivery, while the timestamp allows the broker to compute network latencies.

Determining Failure at a Traced Entity. For every traced entity, a broker maintains information about the previous pings that it had issued. This includes information about when the traced entity was last pinged, and the response times (and loss rates) associated with the last 10 pings. An entity is pinged based on whether the ping interval has elapsed. Depending on the history of the past pings and the duration for which a traced entity has been active, this ping interval is varied. If consecutive pings do not have responses associated with them, the ping interval is reduced to hasten the failure detection of the entity.

If a ping response is not received for a set of successive pings issued at the established ping intervals, a FAILURE SUSPICION trace is reported to the trackers. Lack of responses, from a failure suspected traced entity, for additional pings issued is taken as a sign that the traced entity has failed, and a FAILED trace is issued to the trackers.

State Information from a Traced Entity. A given entity could be in one several states during its presence within the system. These states include INITIALIZING, RECOVERING, READY or SHUTDOWN. A traced entity notifies the broker whenever the state transitions occur, which in turn reports this to the trackers.

Load Information and Network metrics. A traced entity can also issue reports about changes in the load utilization on the machine that is hosting it. The load metrics reported can include changes in both memory and CPU utilization. Depending on the distributed application in question, knowledge of such information can enable trackers to arrive at better decisions while determining the entity to leverage in distributed settings.

Trackers may also be interested in tracing network realm in which the entity operates. Since all interactions from an entity are funneled by the broker that it is connected to, the behavior of the link connecting the broker and the traced entity is extremely important. The nature of the pings and the corresponding responses allow a broker to determine the loss rates, latency and out-of-order delivery rates over the link.

Publishing Trace Information. To enable trackers greater selectivity in the trace information that it chooses to receive, the tracing broker publishes traces on different constrained topics (summarized in Table 2). Furthermore, as we discuss in section 3.5, these traces are issued ***only*** if there are trackers interested in receiving these traces.

The first time a traced entity registers with a broker, the broker issues a JOIN trace on /Constrained/Traces/Broker/Publish-Only/Trace-topic/ChangeNotifications. Other traces published on this topic include FAILURE_SUSPICION, FAILED, DISCONNECT and REVERTING_TO_SILENT_MODE.

Upon receipt of Ping responses from a traced entity, a broker issues the ALLS_WELL trace on the following topic: /Constrained/Traces/Broker/ Publish-Only/Trace-topic/AllUpdates. It is expected that the number of entities interested in receiving these traces would be quite small.

State transition information reported by a traced entity, are reported by the broker on the following topic: /Constrained/Traces/Broker/ Publish-Only/Trace-topic/StateTransitions. Load and network metrics associated with a traced entity are issued over /Constrained/Traces/Broker/Publish-Only/Trace-topic/Load and /Constrained/Traces/Broker/ Publish-Only/ Trace-topic/NetworkMetrics respectively.

Trace type	Topic Information
INITIALIZING, RECOVERING, READY or SHUTDOWN	/Constrained/Traces/Broker/ Publish-Only/Trace-topic/StateTransitions
FAILURE_SUSPICION FAILED, DISCONNECT	/Constrained/Traces/Broker/ Publish-Only/Trace-topic/ChangeNotifications
GUAGE_INTEREST	/Traces/Trace-topic/Request-Response
JOIN, REVERTING_TO_SILENT_MODE	/Constrained/Traces/Broker/ Publish-Only/Trace-topic/ChangeNotifications
ALLS_WELL	/Constrained/Traces/Broker/ Publish-Only/Trace-topic/AllUpdates.
LOAD_INFORMATION	/Constrained/Traces/Broker/ Publish-Only/Trace-topic/Load
NETWORK_METRICS	/Constrained/Traces/Broker/ Publish-Only/ Trace-topic/NetworkMetrics

Table 2: Topics associated with various traces

3.4 Registering to receive traces

Trackers interested in received traces, corresponding to an entity, must first discover the trace topic that has been registered by that entity. A tracker needs to include its credentials in the discovery request; the discovery query has the form /Liveness/Entity-ID, where Entity-ID corresponds to the entity identifier. If the tracker is not authorized to discover the trace topic no response would be received for this query, and the tracker cannot proceed.

If this discovery request is successful, the tracker can proceed to subscribe to the appropriate constrained topics over which different *types* of trace info is published.

3.5 When to publish the traces

In our scheme, traces are issued by a broker only if there are entities that are interested in receiving traces corresponding to a traced entity. To determine if there are any such trackers, the tracing broker issues a GUAGE_INTEREST message on /Constrained/Traces/Broker/Publish-Only/Trace-topic/ Interest. Trackers interested in tracing the entity respond by outlining their interests in any combination of change notifications, all-updates, state transitions, load information or network metrics. This response is published over /Constrained/Traces/Broker/ Subscribe-Only/Trace-topic/Interest

4. Authorization

In this section we discuss issues related to authorization in our framework; specifically, we outline how actions related to the tracing process are restricted. In our authorization scheme we cover the generation, consumption and the routing of these trace messages.

4.1 Subscribing to trace information

Information about traces related to an entity are published on topics comprised of static information, and the trace topic previously registered by the entity. Since the trace topic is based on a randomly generated 128-bit UUID it is extremely difficult to determine or "guess" this information. Thus, it is very difficult for unauthorized trackers to receive trace information about an entity. Since the broker network routes trace messages only to those trackers that previously registered an interest in them, the traces are received only by authorized trackers.

4.2 Individual trace messages

As discussed in section 3.2 an entity needs to demonstrating possession of valid credentials during registration. These credentials are used by a broker to check the validity of other trace messages initiated by the entity. Since our scheme is independent of the underlying transport, we require individual trace messages initiated by a traced entity to demonstrate possession of credentials. For every trace message (including ping responses) initiated at a traced entity, this entity cryptographically signs the trace message. This allows the broker, with which it interacts, to verify both the source of the message as well as whether the message has been tampered with.

4.3 Publishing trace information

Trace information are published on constrained topics of the form /Constrained/Traces/Broker/Publish-Only/. Publishing over these topics is within the purview of the brokers: no entity can publish over these topics. Furthermore, the broker generating these trace messages needs to demonstrate that it is indeed authorized by the traced entity to do so.

A given traced entity needs to explicitly authorize a broker to publish its trace information. To do this, after the entity completes the registration process, the entity also generates an asymmetric key pair. The entity then proceeds to generate an authorization token that includes:

1. Trace-topic information
2. The randomly generate public key.
3. The rights associated with the traces (either publish or subscribe). For a broker, this is set to publish.
4. The duration for which these rights are valid. A traced entity will typically keep this duration short enough to correspond to its expected presence within the system. An entity can generate a new token, once a token is closer to expiration.

The entity then proceeds to sign this token to provide tamper-evidence and to enable verification of the creator of this token. The entity's signature is also part of this authorization token.

One reason why we use randomly generated key-pairs within the token is to ensure that no other broker within the network is aware of the broker that a given traced entity is connected to. Inclusion of the broker's credential within the token can possibly compromise this info.

All trace messages generated by a broker needs to include the token. Messages received at broker, from a neighboring broker, are discarded if they do not posses this authorization token. A broker will also verify the validity of the token. The broker will check to see if the token was signed by the owner of the trace topic, check to see if the token has expired (Use of NTP timestamp ensures that timestamps are within 30-100 milliseconds of each other). If the validity check fails, the message is discarded and not routed within the network.

5. Security

In this section, we describe the security related aspects of our approach. Our discussion pertains to ensuring the confidentiality of trace messages and coping with denial of service attacks. We do not address (and consider it out of our research scope) cryptographic attacks.

5.1 Ensuring confidentiality

An entity may choose to ensure that its traces are cryptographically secured. This section deals with the case where an entity needs to secure its traces. Here, the entity is first responsible for the generation of a secret symmetric key that will be used for encrypting the traces. The entity then securely routes this secret key, along with information about the encryption algorithm and padding scheme, to the broker that it is connected to.

When the broker issues a gauge interest request, it also sets a flag indicating that the traces will be secured. The broker also needs to include its authorization token within this request. Interested trackers, after confirming the validity of the security token, then respond to this gauge interest request by including their credentials and the topic over which it expects responses. The broker then proceeds to publish a secure payload over the topic contained in the response.

To create this secure payload, the broker first creates a message containing the secret trace key, the encryption algorithm and the padding scheme that will be used. The broker uses a combination of the tracker's credential and a randomly generated secret key to secure the payload (this is described in section 4.3). Only the tracker in possession of the private key associated with its credentials can decipher the contents of the message and retrieve the secret trace key.

All trace messages, published by the broker, are encrypted using the secret trace key. Only the trackers in possession of the trace key can decipher the contents of the trace messages.

5.2 Denial of Service attacks

In some cases, an attacker may wish to spurious trace information about an entity. However, since trace information is published over constrained topics, and since the routing brokers expect these published traces to also include valid authorization tokens, brokers will not route such spurious traces. In the case of multiple bogus attempts by a malicious entity, the broker will terminate communications with such an entity.

In some cases, a malicious entity may wish to launch a denial of service attack directly on a traced entity. Except the broker that a given traced entity is connected to, no other entity within the system is aware of the actual physical location of a given traced entity. All communications with a traced entity are based on communications over topics that include the 128-bit UUID contained in its trace topic. Since discovery of this trace topic is itself restricted to the authorized entities, launching attacks is quite difficult. In the unlikely event that this trace topic was compromised, a trace entity can register another trace topic.

6. Performance Benchmarks

We have measured several aspects of our tracking framework, so that the reader has a precise idea of the costs involved. In all our benchmarks that are reported in this section, all processes executed within version 1.4.2 of Sun's Hotspot™ JVM, and the cryptography package used was BouncyCastle (http://www.bouncycastle.org) v1.3. All machines (4 CPU Xeon, 2.4GHz, 2GB RAM) involved in the benchmarks had Linux as the OS, and were hosted on a 100 Mbps LAN.

6.1 Costs for Tracking with multiple hops

In our benchmarks (topology depicted in Figure 1) we have measured costs involved in tracking entities that are 2, 3 and 4 hops away from the trackers. The intermediate brokers were all hosted on different machines. In all cases, to obviate the need for clock synchronizations, the traced entity and the *measuring* tracker (which reports the results) were hosted on the same machine though they were all connected to different brokers.

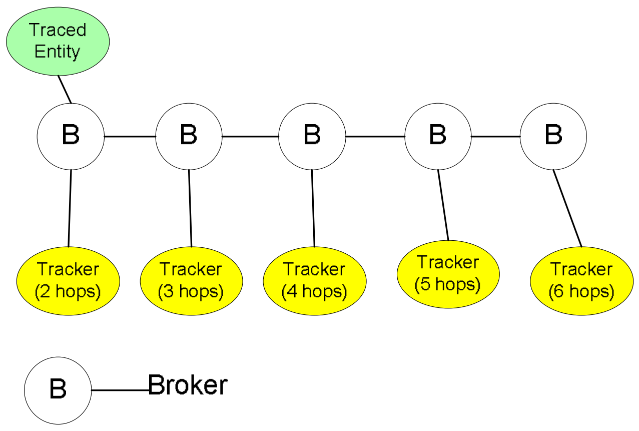

Figure 1: Benchmark Topology

Table 3 summarizes the costs involved in our scheme. We performed our benchmarks under different conditions. First, we did measurements where all communications within the system we based either on TCP or UDP. For each transport, we also measured the costs involved in the tracing scheme when individual traces have messages authorization information (and assorted processing) and cases where the trace messages have authorization information and are also secured. In our experiments for the purposes of signing we used 1024-bit RSA with 160-bit SHA-1 and PKCS#1Padding. For symmetric encryptions and decryptions we used 192-bit AES keys.

Operation	Mean	Standard Deviation	Standard Error
Trace Routing Overhead for different hops (TCP) Authorization Only			
2 hops	72.68	4.14	0.41
3 hops	79.45	4.08	0.41
4 hops	86.4	4.9	0.49
5 hops	93.99	4.33	0.43
6 hops	100.81	4.36	0.44
Trace Routing Overhead for different hops (TCP) Authorization & Security			
2 hops	90.29	4.41	0.44
3 hops	98.12	5.63	0.56
4 hops	105.06	6.17	0.62
5 hops	110.89	7.38	0.74
6 hops	116.21	4.3	0.43
Trace Routing Overhead for different hops (UDP) Authorization Only			
2 hops	70.24	3.45	0.34
3 hops	76.47	3.95	0.4
4 hops	84.02	4	0.4
5 hops	89.78	3.69	0.37
6 hops	96.79	4.61	0.46
Trace Routing Overhead for different hops (UDP) Authorization & Security			
2 hops	88.86	4.52	0.45
3 hops	95.19	5.59	0.56
4 hops	101.76	5.13	0.51
5 hops	107.99	5.81	0.58
6 hops	114.33	4.53	0.45
Security and Authorization Overheads			
Token Generation and Signing	27.19	2.99	0.3
Verifying Authorization Token	2.01	1.04	0.1
Encrypting Trace Message	0.25	0.73	0.07
Decrypting Trace Message	1.15	0.68	0.07
Sign Trace Message	24.51	1.81	0.18
Verify Signature in Trace Message	6.83	1.81	0.18
Sign Encrypted Trace Message	24	1.37	0.14
Verify Signature in Encrypted Trace Message	5.31	1.09	0.11
Key Distribution Overhead			
2-hops	81.53	36.59	8.18
3-hops	114.16	39.29	8.79
4-hops	140.79	40.12	8.97

Table 3: Summary of costs involved in the tracking framework: All results in milliseconds.

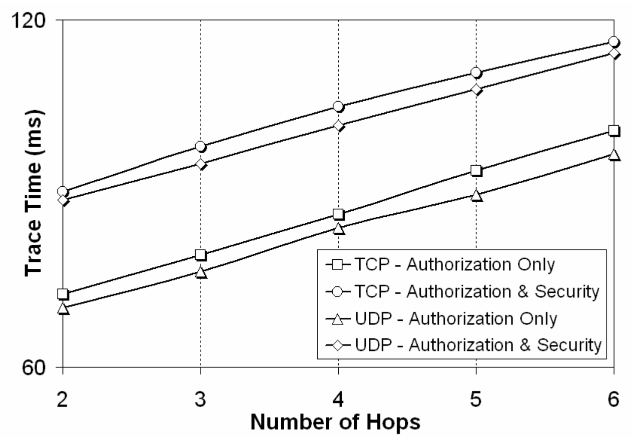

Figure 2: Trace Routing Overhead vs. Hops

Figure 2 depicts the costs involved in the tracing process. Communications over UDP have lower latencies than communications over TCP. Also, when trace message routing based on authorization and security is more expensive than the scheme which involves only authorization since the encryption/decryption costs are not encountered in the latter scheme.

In NaradaBrokering the per-hop communications latency is around 1-2 milliseconds in cluster settings. Additional hops do not significantly increase the routing overhead. Most of the costs for routing of traces are a result of the overheads related to cryptographic operations (also outlined in the table) pertaining to authorization and security related processing.

6.2 Tracing while increasing trackers

We also measured the overheads related to increasing the number of trackers. We did this based on the topology depicted in Figure 3. Here we increased the number of trackers gradually by introducing 10 trackers at a time. The groups of 10 trackers were hosted on different machines.

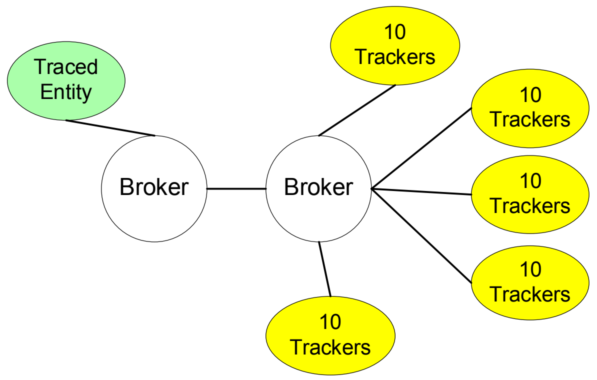

Figure 3: Topology for measuring effect of increasing number of trackers

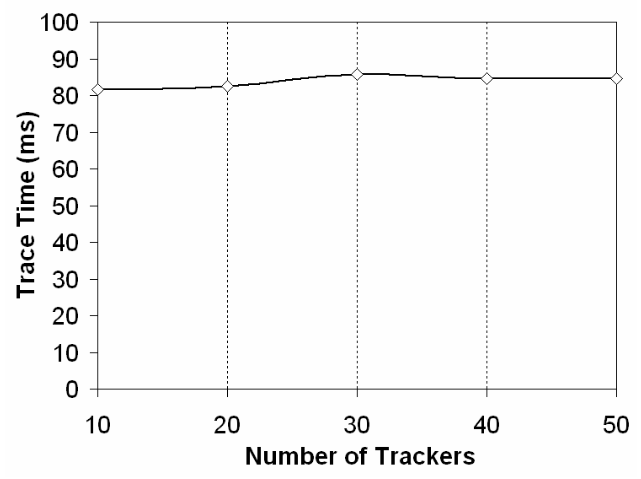

Figure 4: Trace Time vs. Trackers (UDP Based)

Figure 4 summarizes our results; as can be seen the trace time increases very slowly with an increase in the number of trackers. This demonstrates the capability the system to track entities without overloading the brokers.

6.3 Reduction of Signing Costs

In our scheme when a traced entity exchanges messages with its hosting broker, all messages initiated by the traced entity are signed. The broker then constructs the appropriate trace messages with the valid authorization tokens and proceeds to sign the message. To reduce the costs associated with signing of trace messages we introduced an optimization where we eliminate the signing of messages issued by the traced entity to its hosting broker. The traced entity generates a secret symmetric key, and proceeds to securely exchange this key with its host broker. Instead of signing every trace message that it generates, the entity simply encrypts it with its symmetric key. Since only the entity and the broker are in possession of this secret key the broker accepts messages encrypted with this key as having originated by the entity in question. One of the reasons why we did this is that the encryption/decryption costs are cheaper than the corresponding signing/verification cost. Our results in Figure 5 depict the results of using this optimization. As can be seen the authorization enhancement has reduced the tracing costs involved.

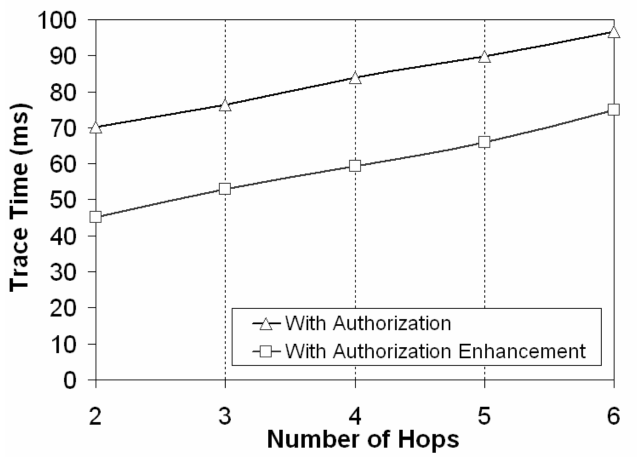

Figure 5: Trace Time vs. Number of Hops with Authorization Enhancement (UDP)

6.4 Increasing the Number of Traced Entities

To measure the overhead introduced by increasing the number of traced entities, we performed an additional experiment. The setup involved 1 broker, and 30 trackers; while keeping the number of trackers constant, we increased the number of traced entities. The results, reported in Table 4, are for the case where there are 10, 20 and 30 entities being traced actively.

To cope with clock skews and to avoid synchronization problems, we had the traced entities and the trackers reside on the same machine. However, this configuration also results in lowering the performance figures since the security operations related to the generation of trace messages are compute intensive. Since these operations are performed by every traced entity for every trace that is generated, this impacted the overall performance in the experimental setup. We expect the performance to be significantly better in practical settings.

Number of Traced Entities	Mean	Standard Deviation	Standard Error
10	75.64	19.79	0.42
20	85.43	30.53	0.44
30	118.77	54.98	0.67

Table 4: Trace routing overhead (milliseconds) by increasing number of traced entities (TCP based)

7. Related Work

The Network Weather System (NWS) [4] collects end-to-end throughput and latency information and uses that information to forecast future performance. Metrics are collected by sensors, which are organized as a hierarchy of sensor sets called cliques in order to prevent contention and also to provide scalability. In addition to network metrics, collected over the TCP/IP transport protocol, NWS also accumulates CPU and available non-paged memory information from various nodes. Remos [5] provides a query based interface for applications to obtain information about their execution environment including network state. Remos maintains both static and dynamically changing information and is based on SNMP measurements on the network router nodes.

Vogels, in Ref [6] provides an excellent overview of the need for failure detection in large distributed systems. Issues related to failure detection and improving the failure detection through the use of process checkpoints and process Upcalls are also outlined.

Renesse, Minsky and Hayden described the first gossip based failure detection service in Ref [7]. In gossip systems, a give node gossips (and passes information) to a set of randomly selected nodes. Gossip system tends to scale well and have no single point of failures. However, systems based on gossip schemes need to address the consistency issue which results from uneven propagation of the gossips. The GEMS (Gossip Enabled Monitoring Service) [8] system provides a scaleable resource monitoring service. Nodes within the GEMS system gossip with each other about information related to resource monitoring. The approach taken here is that of a layered gossip scheme, where nodes are organized into gossip trees. Since gossiping can sometimes lead to uneven spread of failure information, the system relies on consensus: a majority is needed for deeming a failure.

Log-Based Receiver-reliable Multicast (LBRM) [9] protocol describes a scheme to provide scalable and timely dissemination of state updates, that satisfy the needs of multicast sources within Distributed Interactive Simulations. The variable heart-beat scheme in LBRM clusters heartbeat transmissions in the time period after a data-transmission rather than evenly distributing these heartbeats during idle times when data is not being transmitted.

8. Conclusions

A scaleable and secure tracking scheme is important in several loosely-couple distributed systems. In this paper we described our scheme for tracking the availability of entities in distributed systems in a secure and authorized fashion. This work leveraged the publish/subscribe paradigm to achieve this. Our experiments confirm the suitability of this scheme.

Acknowledgements

This research is supported by grants from the National Science Foundation's Division of Earth Sciences project number EAR-0446610, and the National Science Foundation's Information and Intelligent Systems Division project number IIS-0536947.

References

[1] S. Pallickara and G. Fox. NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. Proceedings of the ACM/IFIP/USENIX Middleware Conference Middleware-2003. pp 41-61.

[2] S. Pallickara, G. Fox and H. Gadgil. On the Creation & Discovery of Topics in Distributed Publish/Subscribe Systems. Proc. of IEEE/ACM GRID 2005. Seattle, WA.

[3] S.Pallickara, H. Gadgil and G. Fox. On the Discovery of Brokers in Distributed Messaging Infrastructures. Proceedings of the IEEE Cluster 2005 Conference. Boston.

[4] R. Wolski. Forecasting network performance to support dynamic scheduling using the network weather service. Proceedings of the 6th IEEE Symp. On High Performance Distributed Computing, 1997.

[5] B. Lowecamp et al. A resource query interface for network-aware applications. In Proc. 7th IEEE Symp. On High Performance Distributed Computing, 1998.

[6] Werner Vogels: World wide failures. ACM SIGOPS European Workshop 1996: 115-120

[7] R. Van Renesse, R. Minsky, and M. Hayden, "A Gossip-style Failure Detection Service," Proc. of the IFIP Conference on Distributed Systems Platforms and Open Distributed Processing Middleware, 1998, pp 55-70.

[8] R. Subramaniyan et al, GEMS: Gossip-Enabled Monitoring Service for Scalable Heterogeneous Distributed Systems, Cluster Computing, Vol. 9, No. 1, Jan. 2006, pp. 101-120.

[9] Holbrook, H., Singhai, S. and Cheriton, D., Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation, Proc. of ACM SIGCOMM'95, September 1995