Working Papers Series
INTERNET INTERDISCIPLINARY INSTITUTE

Anonymous communications for JXTA peer-to-peer services:
A multi-protocol approach

Joan Arnedo-Moreno (jarnedo@uoc.edu)
Marc Domingo-Prieto (mdomingopr@uoc.edu)
Noemí Pérez Gilabert (nperezg@uoc.edu)
IN3 – Universitat Oberta de Catalunya

Working Paper

Working Paper Series WP12-003

Research group: K-ryptography and Information Security for Open Networks: KISON
Research group coordinator: David Megías Giménez (UOC)

Submitted in: October 2012
Accepted in: January 2013
Published in: February 2013

Universitat Oberta de Catalunya
IN3 INTERNET INTERDISCIPLINARY INSTITUTE
http://in3.uoc.edu

Internet Interdisciplinary Institute (IN3)
http://www.in3.uoc.edu
Edifici MediaTIC
c/ Roc Boronat, 117
08018 Barcelona
Espanya
Tel. 93 4505200

Universitat Oberta de Catalunya (UOC)
http://www.uoc.edu/
Av. Tibidabo, 39-43
08035 Barcelona
Espanya
Tel. 93 253 23 00

Table of content

Anonymous communications for JXTA peer-to-peer services

A multi-protocol approach

Joan Arnedo-Moreno (jarnedo@uoc.edu)

Marc Domingo-Prieto (mdomingopr@uoc.edu)

Noemí Pérez-Gilabert (nperezg@uoc.edu)

IN3 Working Paper Series is a monograph series promoted by the Internet Interdisciplinary Institute (IN3) of the UOC
IN3 Working Paper Series (2012) | ISSN 2013-8644 | http://in3wps.uoc

Abstract
JXTA is an open peer-to-peer (P2P) protocols specification that, in its about 10 years of history, has slowly evolved to appeal to a broad set of applications. As part of this process, some long awaited security improvements have been included in the latest versions, providing basic message privacy and authentication. However, under some contexts, more advanced security requirements should be met, such as anonymity. Even though several approaches exist to deploy anonymity in P2P networks, no perfect solution exists that is able to cater to every possible scenario. In this work, we propose how to adapt JXTA messaging to allow services to be anonymously accessed in a manner that is completely invisible to the existing protocols, taking advantage of JXTA's idiosyncrasies and capabilities,. To achieve this end, we propose a multi-protocol anonymous layer, so the most suitable one can be chosen depending on the scenario.

Keywords
peer-to-peer, security, anonymity, JXTA, Java

Recommended citation:

ARNEDO-MORENO, Joan; DOMINGO-PRIETO, Marc; PÉREZ-GILABERT, Noemí (2012). "Anonymous communications for JXTA peer-to-peer services: A multi-protocol approach" [online working paper]. (Working Paper Series; WP12-003). IN3 Working Paper Series. IN3 (UOC) [Accessed: dd/mm/yy].
<http://in3wps.uoc.edu/ojs/index.php/in3-working-paper-series/article/view/n12-arnedo/n12-arnedo>

1. Introduction

JXTA (Project Kenai, 2011) is a well-known open protocol specification that enables the deployment of peer-to-peer (P2P) applications, allowing a set of heterogeneous devices to group and collaborate regardless of the underlying network topology. Created by SUN in 2001, JXTA has iterated through successive revisions during its 10 years of history, slowly gaining popularity, with over 2,700,000 downloads and more than 120 active projects. Such projects range from real-time collaboration or web search to remote robot control (Connex, 2007; Matsuo K, 2009; Lele N., 2009).

The latest version at present time, JXTA 2.7, has been made available at the start of 2011. Its main highlights include some long awaited basic security improvements, such as secure peer groups, advertisement signatures and crypto-based identifiers. However, there is still a long way to go on that regard. As P2P systems evolve and are used in new scenarios, more advanced security capabilities become important. A significative example is message anonymity (Hansen M., 2008), for which privacy may be a necessary, but not sufficient, requisite.

Usually, the concept of anonymity in P2P networks is associated with situations where exposure has very strong implications, such as maintaining the right to free speech or circumventing legal responsibilities (Wallace J., 1999). However, there are everyday situations which require anonymity, but that even in the worse case scenario nobody may end up in jail. Some straightforward examples are a corporate suggestion box, a small ballot system or a peer evaluation form.

It is worth remarking that, in each of these scenarios, in order for messages to remain truly anonymous, it must no possible for anybody within the P2P network to discover the message source, no matter what tool is used or its role in the system. It is not enough that the system just hides source message information to the users at the upper protocol layer or such data is managed only by a trusted entity, since it may still be possible to extract the data using external methods or simply due to misuse. Low level protocols must actually make it inaccessible. Even the mere case that users know that a system administrator is still able to expose them may be enough to stymie taking part in transactions. Therefore, should this requirement be fulfilled, actual anonymity provides the added bonus that, by also increasing the users' trust in the system participation is also encouraged.

There are several ways to attain anonymity on a P2P system. Unfortunately, no silver bullet exists. Every different approach has its own strengths and weaknesses and choice entirely depends on the system's requirements and its operating context.

IN3 Working Paper Series is a monograph series promoted by the Internet Interdisciplinary Institute (IN3) of the UOC
IN3 Working Paper Series (2012) | ISSN 2013-8644 | http://in3wps.uoc

Several surveys on anonymity on unstructured P2P networks exist (Bhatti S. 2007; Ren-Yi X., 2008):, each one proposing a different taxonomy to categorize current approaches. Our work is based on the latter, being the most recent one. According to the author, anonymous protocols can be divided into three categories: unimessage, split message and replicated message.

Protocols into the same family tend to share the same strengths and weaknesses. For instance, the unimesage category is the one considered to include the protocols with the highest degree of anonymity. However, protocols under this category also usually share the disadvantage that they put a heavy burden on the source peer, since some precomputation and periodical network probing is necessary prior to sending a message. On the other hand, split message-based approaches sacrifice some amount of anonymity to lessen the computational burden on peers, and specially excel at providing high reliability in dynamic environments. Finally, replicated message-based approaches provide very strong destination anonymity in a really simplistic manner at the cost of a very high bandwidth occupation.

In this paper, we study the feasibility of anonymous communications in JXTA by proposing an anonymity service that relies on a multi-protocol approach. This is achieved by adapting specific protocols from each category to JXTA's architecture. Thus, it is possible to deploy the most suitable protocol given the application's context. Additional contributions of this paper encompass some improvements over some of the original protocols in order to create a service that fully realizes the capabilities that JXTA already provides, minimizing the amount of required changes to an existing system in order to integrate anonymous messaging.

This paper is structured as follows. In Section 2, we review some of the most relevant protocols from each category, as far as this paper's goals are concerned, to anonymity within the context of P2P systems. Section 3 provides a brief overview of JXTA's main architectural highlights, and then, Section 4 describes our proposed anonymity method and how it is adapted to the specifics of JXTA's design and protocols. Finally, Section 4 concludes this paper and outlines further work.

2. Current approaches to anonymity in P2P networks

In this section we describe the gist of main mode of operation in protocols from each category and briefly review some of the most relevant ones. Nevertheless, a much more thorough description can be found in the original sources.

In unimesage based protocols, an anonymous path is pre-constructed by the sender before a message is dispatched towards the actual final destination. The

message will be relayed through every peer in the path before it reaches its destination. However, the message is encrypted in such a manner that, at every hop, each relay peer is only able to learn which the next hop is where the message must be forwarded. No peer, apart from the initial message sender, is able read the whole data path, not even the final destination. Therefore, every time a peer receives a message, it cannot be decided whether it has been received from the original sender or just a former relay. For the same reason, it is not possible to know whether the next hop is the destination either.

Message path encryption is usually performed by adding successive encryption layers to the original message using public key cryptography. The public key of each relay is applied at each layer, so when a relay decrypts the received message with its private key, the only information found is the identity of the next hop and the encrypted data that must be forwarded. Only the final destination peer will be able to obtain the original message after the applying the decryption process. Because of the way the process works, this anonymity approach is also called Onion Routing (Goldsclag D, 1997). This is the most popular approach to anonymity in P2P networks, used by systems such as Tor (Mathewson N., 1998) or APFS (Shields C., 2001). This basic idea is also used in other systems, such as Crowds (Rubin A.D., 2004) or Shortcut (Zhang X., 2003), albeit with some modifications. The path is not predetermined by the sender, but probabilistically decided at each hop. In such approaches, however, only sender anonymity is achieved. Any relay peer knows the destination, since it is transmitted in clear text.

On the other hand, a split message-based approach relies on threshold systems (Desmedt Y.G.., 1989), and it is mainly used for file publication. In these kind of systems, a secret is split into n parts, which are distributed through the network. It is enough that t parts ($t < n$) are gathered to be able to recover the original secret. This idea is directly applied to documents or messages, which take the role of the shared secret, distributed among the peers. The peer which is able to recover the secret parts and reconstruct the original message is the one that finally transmits the data to the final destination, acting on behalf of the original source peer, thus hiding its identity. The main strength of split message-based approaches, in comparison with approaches from other categories, is its high reliability and ability to cope with dynamic environments. However, it comes at the cost of a quite high latency and traffic overhead. Since secret distribution usually relies on heavy use of flooding mechanisms, many messages must be transmitted across the whole network.

Some well-known systems which rely on this approach are the FreeHaven Project (Freedman F.J., 2001) and the Jigsaw Distributed File System (JigDFS) (Bian J., 2009). Both make use of Rabin's information dispersal algorithm (IDA) (Rabin M.O., 1989) to break the transmitted a file into parts, following the guidelines of a threshold system. Nevertheless, this approach is also used beyond document distribution techniques. PUZZLE (Han J., 2008) proposes an anonymous protocol specially suited to Mobile P2P Networks (MOPNET) taking advantage of the broadcast feature of

mobile nodes when distributing split secret parts in order to mitigate the traffic overhead produced by flooding. In this proposal, the IDA algorithm is no longer used to split queries, again focusing on document. The basics laid by PUZZLE are also used in Rumor Riding (RR) (Li Y., 2011), a protocol which, in fact, can be considered its latest. In this proposal, an AES symmetric key is generated and used to encrypt the message. Both the key and the ciphertext are arranged into different packets, called rumors, which are separately forwarded across the network following a random walk algorithm. Any peer which receives both rumors will be the one to actually send the message to the destination, acting as the original source's proxy.

As far as a replicated message approaches are concerned, they rely on using message broadcast or multicast to send the content to everyone, thus hiding which peer within the network is the actual destination. Messages are encrypted so only the destination will be able to read the content, maintaining its privacy. A generic approach which relies on the basics of this category can be traced back as far as (Dolev S., 2000), where multicast based communications are used as a way to anonymously transmit data in generic networks. Nevertheless, the basic ideas have been later adapted to P2P networks.

One of the main contributions to this category can be found in P5 (Sherwood R., 2002), where Peers are divided into a structure of hierarchical channels, according to the result of applying a hash algorithm on each peer's public key. Another proposal that specially focuses on receiver anonymity can be found in (Waters R.B., 2003), since the authors argue that sender anonymity is the most favored scenario in current proposals. This protocol relies on multicast groups to transmit encrypted messages, guaranteeing that only the actual receiver is the only one able to decrypt and process the message.

Finally, some hybrid approaches are also encompassed into this category, such as Hordes (Levine R.B., 2002), which combines the features of path-based and replicated message protocols. Message requests are sent using probabilistic forwarding, whereas responses entirely rely on multicast transmission, trying to reach a compromise between both categories' strengths and weaknesses.

3. JXTA architecture overview

In order to propose an anonymizing service for JXTA messaging which takes advantage of the middleware's basic capabilities, it is important to review the most relevant characteristics of its architecture. From this review, it is possible to create an anonymity layer which nicely integrates with JXTA without the need to define additional protocols or core primitives.

The main idiosyncrasy in JXTA's design, which sets it apart from other P2P frameworks, is introducing the concept of Peer Group, a segmentation of the global JXTA network. All peers publish and consume services within the context of a group, interacting with each other by using JXTA's core services. The most important core services are the Membership Service, the Discovery Service and the Pipe Service.

The **Membership Service** allows joining a peer group and claiming a unique identity within the group's context. Through this service, each group member is provided with a credential, which may be used at any time to authenticate to other group members. Different implementations exist depending on the chosen mechanism to claim such identity and the credential format.

The **Discovery Service** manages group resources publication and discovery. Every resource in a JXTA group is described by an Advertisement, an XML metadata document. A resource cannot be accessed unless its corresponding Advertisement is previously retrieved. It is the Discovery Service's responsibility to manage and distribute Advertisements, since a resource is not considered available unless its advertisement is periodically published.

There are several Advertisement types, but the most important ones are:

- *Peer Advertisement*: Describes a peer and the resources and services it provides, under a special service parameter entry. Each peer is responsible for the publication of its own Peer Advertisement, and is considered online only while it continues to do so.
- *Group Advertisement*: Similar to the previous one, but describing a peer group instead. It is also used as the group's presence mechanism in the network.
- *Pipe Advertisement*: Describes a JXTA Pipe, an abstract communication channel and the main mechanism to exchange data between applications.

Finally, the **Pipe Service** is responsible for managing message exchanges using JXTA Pipes. The simplest pipe in JXTA, the *JxtaUnicast*, provides an asynchronous, unidirectional message transfer mechanism which can be easily established and managed. Nevertheless, there is a higher-level communication abstraction, the *JxtaBiDiPipe*, which provides a bidirectional communication channel. The latter is usually preferred by services, since it allows a straightforward query-response message exchange.

Following the description of JXTA's standard service model using the Pipe Service is explained:

1) Each service provider starts a JXTAServerPipe using the Pipe Service, which makes available and listens to an input pipe in order to process inbound communication requests. This input pipe is defined using a Pipe Advertisement.
2) The service provider publishes the Pipe Advertisement to other group members using the Discovery Service.

IN3 Working Paper Series is a monograph series promoted by the Internet Interdisciplinary Institute (IN3) of the UOC
IN3 Working Paper Series (2012) | ISSN 2013-8644 | http://in3wps.uoc

3) The Advertisement is propagated within the group by Rendezvous Peers, special super-peers who efficiently distribute Advertisements.
4) To consume a service, a peer also uses the Discovery Service to retrieve the Pipe Advertisement. Then, a connection is established via the Pipe Service and the consumer may begin sending messages.
5) Once a message is received at the server side, the results depend on the pipe type, JxtaUnicast or JxtaBiDiPipe. In the former case, messages may be processed, but no response is possible. On the latter case, a linked outbound communication channel is created and two-way exchanges are made possible.

JXTA messages sent through pipe connections follow a predefined structure comprised of a set of name/value pairs labeled under a namespace and organized as an ordered sequence. As a message passes down each JXTA layer, one or more named elements may be added to the message (for example, control data). Their order within the message structure always follows the same order they were added. As a message is processed back up the stack, each layer will remove these elements, until only application data remains.

Message exchanges can be secured in JXTA by using a group based on the PSE (Personal Security Environment) Membership Service implementation. Under this kind of peer group, each peer is provided with a credential based on public key cryptography. This guarantees that each peer has initialized a valid pair of public-private keys and that the public key of each peer is automatically distributed inside its Peer Advertisement, in a special service parameter entry. Currently, other Membership Service implementations are unable to provide any security at all.

4. Anonymity layer integration

In this section we present the multi-protocol anonymity layer for JXTA and the message protocol integration specification. One protocol has been chosen from each of the previously described categories: Onion Routing, Rumor Riding and Hordes. Due to space constraints, this paper will not go into much detail about the original proposal's protocol description, only focusing on its adaptation and integration to JXTA's architecture. Such information, if necessary, can be obtained from the original sources detailed in the literature review in Section 2.

An overview of the anonymizer layer integration within the context of the JXTA general architecture is shown in Figure 1. The bottom of the figure lists the main tasks

that must be solved in order to properly succeed in the integration process. It is enough that a peer group member locally executes the anonymity service to be able to send anonymous requests to other services in the same group.

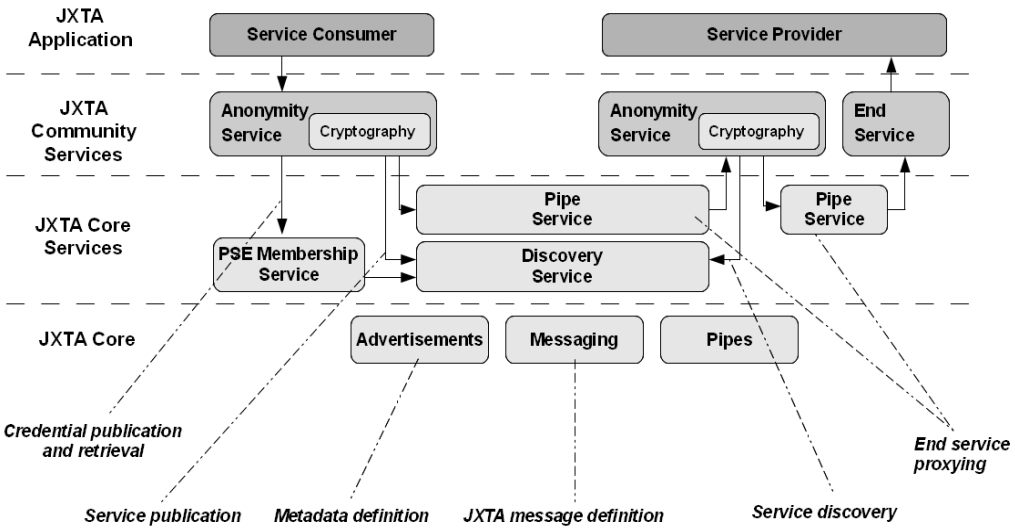


Figure 1. Anonymizer layer within the context of JXTA's architecture

4.1. Anonymity service publication

Just like any JXTA service, the Anonymity Service's Pipe Advertisement must be distributed among other peer members before it may receive incoming requests. Each peer is responsible for the publication of its own service instance's pipe and this procedure must be periodically executed.

In order to maintain the number of advertisements transmitted within the network at a minimum, the service's PipeAdvertisement is piggybacked within each peer's Peer Advertisement, indexed by a hardcoded well-known service identifier. In fact, this is the same method the PSE Membership Service employs to readily distribute public key information. The main advantage of this method is that it is only necessary to manage a single advertisement type to publish or discover all data related to the Anonymity Service (service pipe and peer cryptographic data). Furthermore, the Peer Advertisement's publication and discovery is already part of JXTA's standard procedures, being every peer's presence mechanism. Therefore, Peer Advertisements of group members considered online are always readily available.

A sample Peer Advertisement is shown in Figure 2 (some encoded data has been shortened for the sake of readability). The first grey section (a) denotes the Anonymity Service parameters, which basically consist of only the Pipe Advertisement. The

second grey section (b) denotes the Membership Service parameters. For the case of PSE, that is the peer's public key, encapsulated in a PKIX certificate. Whenever the Anonymity Service is executed, it is enough that a new service parameter entry is created at the Peer Advertisement. When the service is closed, the entry is removed.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jxta:PA>
<jxta:PA xml:space="default" xmlns:jxta="http://jxta.org">
  <PID>urn:jxta:uuid-59616...7B03</PID>
  <GID>urn:jxta:uuid-425A5C703CD5454F9C03938A0D65BD5002</GID>
  <Name>Peer_1</Name>
  <Desc>Created by NetworkManager</Desc>
  <Svc>
    <MCID>urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE0000001105</MCID>
    <Parm>
      <jxta:PipeAdvertisement>
        <Id>urn:jxta:uuid-425A5...4704</Id>
        <Type>JxtaUnicast</Type>
        <Name>ANONYM-PIPE:Peer_1</Name>
      </jxta:PipeAdvertisement>
    </Parm>
  </Svc>                                                    (a)
  <Svc>
    <MCID>urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE0000000805</MCID>
    <Parm>
      <jxta:RA xml:space="preserve" xmlns:jxta="http://jxta.org">
        <DstPID>urn:jxta:uuid-59616...7B03</DstPID>
        <Dst>
          <jxta:APA>
            <EA>jxtatls://uuid-5961...B03</EA>
            <EA>cbjx://uuid-59616...B03</EA>
            <EA>tcp://213.73.36.53:9701</EA>
          </jxta:APA>
        </Dst>
      </jxta:RA>
    </Parm>
  </Svc>
  <Svc>
    <MCID>urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE0000000105</MCID>
    <Parm>
      <RootCert type="jxta:cert">MIIBkT...bV7</RootCert>
    </Parm>
  </Svc>                                                    (b)
</jxta:PA>
```

Figure 2. Sample Peer Advertisement.

4.2. Unimessage protocol integration

The chosen protocol from the unimesage family is Onion Routing, the main reasons being twofold. First of all, it is the one which keeps a better efficiency while maintaining a high anonymity degree. Keeping a good efficiency is especially relevant since message anonymity usually has a very high impact in system performance, even when compared to other security services. Secondly, it is based in an architecture where nodes are completely autonomous and communications are basically unidirectional, which meshes with the principles of JXTA messaging, the Pipe Service.

The protocol is divided in three distinct phases: onion generation, message setup and message processing. The two initial phases are executed at the source peer whereas the last one is executed whenever a peer receives an anonymous message.

IN3 Working Paper Series is a monograph series promoted by the Internet Interdisciplinary Institute (IN3) of the UOC
IN3 Working Paper Series (2012) | ISSN 2013-8644 | http://in3wps.uoc

Services send the response back to the source by using the same procedure, with just some minor modifications.

4.2.1. Onion generation

This step initiates the onion routing process and is performed by the peer, the initator I, who wants to anonymously access an end service deployed at a remote peer, the Responder R, within the same group. In fact, the onion layering procedure is used at several steps in the message setup process.

The layering procedure needs two parameters: *OnionCore*, the JXTA message structure to be sent, which may contain any arbitrary data, and *PeerAdvList* = {Adv_1; …;Adv_n}, an ordered sequence of Peer Advertisements.

For each Peer Advertisement *Adv_i*, from n … 1, the following steps are iteratively executed. *Onion_i* is considered the result of each iteration, being Onion_1 the final result.

1) This iteration's input, *inputData*, is chosen.
   - For the first iteration (i = n) the *OnionCore* structure is considered *inputData*.
   - For the rest of iterations (i = n -1…1), the result from the previous iteration, *Onion_i+1,* is considered *inputData*.
2) The public key *PK_i* is retrieved from *Adv_i*'s Membership Service definition entry (see Figure 2). Under the context of a peer group which implements the PSE Membership Service, it is guaranteed that *PK_i* exists.
3) *inputData* is encrypted using *PK_i* under a wrapped key scheme (Staddon J, 1998), generating *EncInputData*.
4) *Adv_i*'s *PID* field (the peer's unique identifier) is retrieved.
5) A new *OnionLayer* structure is generated by creating a JXTA Message with the following fields:
   - NextHop = *PID*
   - OnionRoute = *EncInputData*
6) The *OnionLayer* structure becomes this iteration's result (*Onion_i*).

4.2.2. Message setup

Once the layering procedure has been established, the message setup process description follows. The description assumes that a query-response cycle will be executed at the end service, and therefore, a response is expected from R.

1) The initiator's (I) corresponding client (*EndClient*) creates a query message (*JXTAQueryMessage*), specifying which is the destination peer (peer R), and a callback structure (*PipeListener*). Such structure is used by JXTA applications to allow asynchronous processing of the incoming response, so they do not need to block until the reply is received.
2) *EndClient* locates the End Service's JXTABidiPipe Advertisement, from now on *EndServicePipe*, using the Discovery Service.

   So far, steps 1-2 describe the required steps to access a generic JXTA service. At this point, *EndClient* would open a connection using *EndServicePipe* and use it to directly send *JXTAQueryMessage*. Instead, it decides to use the anonymity service instead.

3) If an instance of the anonymity service is being executed at the initiator, an anonymous client (*JXTAnonymClient*) is also available. Such client is invoked, receiving *EndServicePipe*, R's identity, *JXTAQueryMessage* and *PipeListener* as parameters. From now on, *JXTAnonymClient* will manage the rest of the message setup process. *EndClient* considers message processing completed as far as it is concerned and just waits for the callback structure to process the eventual response.
4) A random symmetric key (*FinalSymKey*) is generated and used to encrypt both *JXTAQueryMessage* and *EndServicePipe*, obtaining *QueryEncryptedMessage*.
5) A new record is created in a local table *PendingQueries*, containing *FinalSymKey* and *PipeListener*, the former being the primary key.
6) A set of random data (*RndData*) is generated. The length of this data should be between 2411 and 4614 bytes. This size ensures that the message is not vulnerable to some attacks which may try to guess the number of message hops by its size (Syverson P., 2001).
7) The initiator generates a *ResponseCore* structure. This structure is a JXTA message composed of the following name-value pairs:
   - RandomData = *RndData*
   - SymmetricKey = *FinalSymKey*

8) At this point, a number of OnionRouter peers must be chosen in order to create a response path. It is considered that 3 hops is good enough (*Mathewson N., 1998*). This is done by using the Discovery Service to get a set of Peer Advertisements RP = {Adv_1, Adv_2, Adv_3, Adv_4} where is it true that for any Adv_i; (i < 4); Adv_i != Adv_I and Adv_i = Adv_R and Adv4 = Adv_I, and in all of them, the JXTAnonymSvc service parameter field exists (all of them deploy the anonymity service). It is worth remarking that, since S holds the end client waiting for the response, that is the reason why it must be the last peer in the response path.

9) Once the response path is established, an onion structure *ResponseOnion* is created using the layering procedure previously described in section 4.2.1. The input parameters are *ResponseCore* and *RP.*
10) From *ResponseOnion* the NextHop and the OnionRoute fields are extracted. Their values are *ResponseFirstHop* and *ResponseRoute*, respectively.
11) A *QueryCore* structure is created. This structure is a JXTA Message composed of the following name-value pairs:
    - NextHop = *ResponseFirstHop*
    - OnionRoute = *ResponseRoute*
    - SymmetricKey = *FinalSymKey*

12) Now the query path, *QP,* must be generated. This process is identical to step 10, but in this case, Adv_4 = Adv_R instead of Adv_I Ideally, RP != QP.
13) Once the query path is established, an onion structure *QueryOnion* is created using the layering procedure previously described. The input parameters are *QueryCore* and *QP*.
14) From *QueryOnion* the NextHop and the OnionRoute fields are extracted. Their values are *QueryFirstHop* and *QueryRoute* respectively
15) An *OnionMessage* structure is generated. This structure is a JXTA Message composed of the following name-value pairs:
    - OnionRoute = *QueryRoute*
    - EncryptedMessage =*QueryEncryptedMessage*

16) The Peer Advertisement of *QueryFirstHop* is obtained via the DiscoveryService, and its anonymity service Pipe Advertisement extracted. A new connection to this pipe is created and *OnionMessage* sent through it.

4.2.3. Message processing

This step is performed when an anonymous message is received at any peer that has deployed the anonymity service. The process starts when a running instance of the anonymity service receives an incoming message (*OnionMessage*), which contains a EncryptedMessage and OnionRoute fields. Then, the value in the OnionRoute field is extracted and decrypted using the peer's local private key, accessible using JXTA's PSE Membership service. At this point, depending on the decryption process result, three different cases may apply. They are summarized in Figure 3.

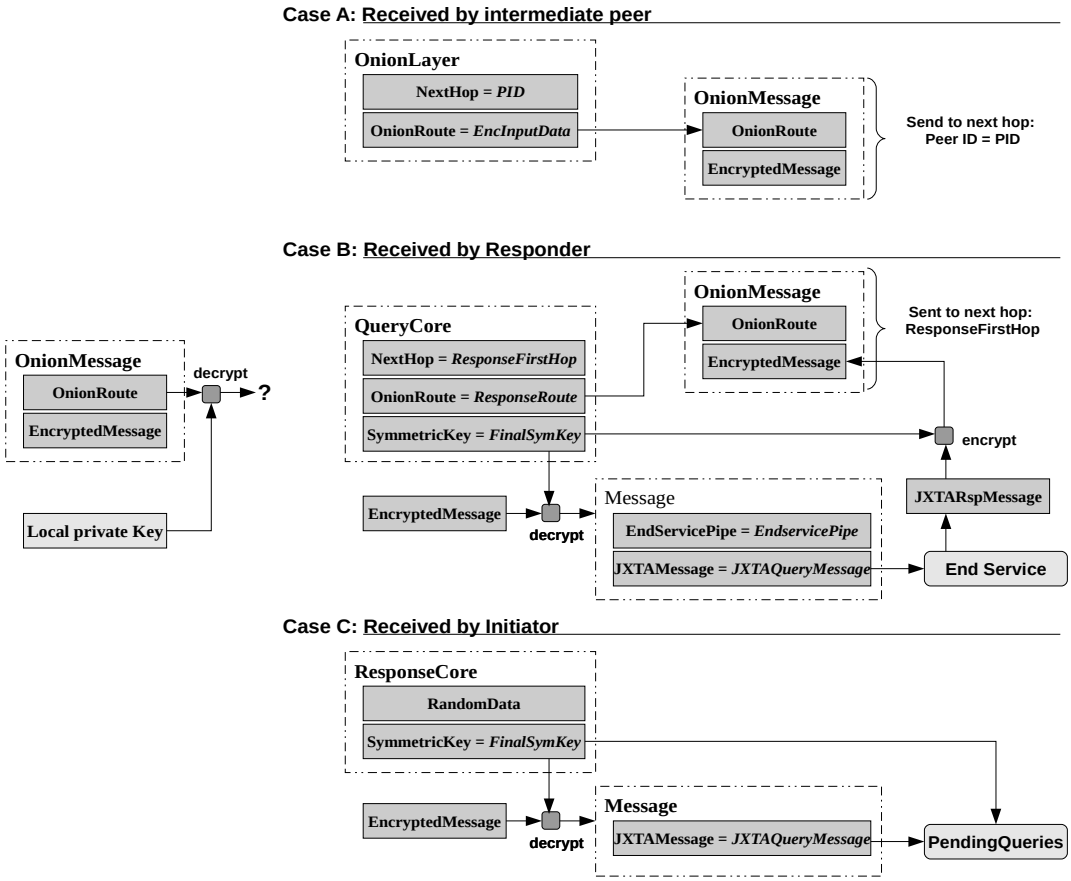


Figure 3. Message processing summary

Should the extracted data become an OnionLayer (Case A) the message must be routed to another peer.

1) The values stored in the NextHop and OnionRoute fields, respectively *PID* and *nextRoute*, are extracted.
2) A new *OnionMessage* structure is generated, where:
   - OnionRoute = *nextRoute*
   - EncryptedMessage =*EncryptedMessage*

3) Using the Discovery Service, *PID*'s Peer Advertisement is located. From this advertisement, the anonymity service Pipe Advertisement is extracted.
4) A new connection to the next hop is established using the previously recovered Pipe Advertisement. The newly generated *OnionMessage* is sent through it.

In the case that the extracted data is a *QueryCore* structure (Case B), the current peer must be the responder, R, executing *EndService*. The query must be processed

by this service. In this scenario, the EncryptedMessage field holds the value *QueryEncryptedMessage*.

1) The values *stored* in the NextHop, OnionRoute and SymmetricKey fields, *ResponseFirstHop*, *ResponseRoute* and *FinalSymKey* respectively, are extracted.
2) *QueryEncryptedMessage* is decrypted using *FinalSymKey*, obtaining the original *JXTAQueryMessage* and *EndServicePipe*.
3) A bidirectional connection is established to the *EndService* using *EndServicePipe* and the client's query, *JXTAQueryMessage*, is sent through it. As far as the *EndService* is concerned, a normal connection has been established. It is oblivious to the anonymity layer.
4) The process waits until a response, *JXTARspMessage*, is received
5) The response is encrypted using *FinalSymKey*, generating *ResponseEncryptedMessage*.
6) A new *OnionMessage* structure is generated, where:
   - OnionRoute = *ResponseRoute*
   - EncryptedMessage = *ResponseEncryptedMessage*
7) *OnionMessage* is sent to the anonymity service's pipe in *ResponseFirstHop*.

Finally, it is also possible that the extracted data is identified as a *ResponseCore* structure (Case C). In this case, the current peer must be the initiator, I, meaning that the query/response the round-trip has come to an end. In this scenario, the *EncryptedMessage* field holds the value *ResponseEncryptedMessage*.

1) The value stored in the SymmetricKey field is extracted (*FinalSymKey*). The RandomData field is ignored.
2) *ResponseEncryptedMessage* is decrypted using *FinalSymKey*, obtaining the original *JXTAResponseMessage*.
3) The *PendingQueries* local table is searched for the record which uses *FinalSymKey* as its primary key.
4) Using the *PipeListener* structure, the message is provided to the original EndClient. As far as the end client is concerned, JXTA has acted just like during any standard service access. The anonymity layer was invisible to it.

4.3. Split-message protocol integration

As far as the split-message protocol category is concerned, we have chosen Rumor Riding (RR) as the foundation for our work, the main reasons being that it is one of the most recent proposals. Even though RR is focused on document retrieval via flooding mechanisms, it can be easily adapted to point-to-point end-service access.

The protocol is divided in four phases, encompassing the whole query-response cycle: query setup, random walk, response processing and backtracking.

In contrast with Onion routing, RR is a stateful protocol which requires that all peers executinga the Anonymity Service track rumors as they travel across the network using a local cache, RumorDB, a table with the following fields:

- *RumorID*: Rumor identifier.
- *CachedRumor*: A cached rumor of any type.
- *Inbound*: Peer ID from which the rumor was received.
- *Outbound*: Peer ID the rumor was forwarded to.
- *TimeStamp*: Time the rumor was routed.

Given the TimeStamp field, all entries expire after some time and are automatically flushed from the cache. This avoids cluttering from stagnant entries related to messages which have been dropped for some reason during transit.

4.3.1. Query setup

This step starts the whole process and is executed at the service consumer at the initiator (I) whenever it wishes to send a query to the end-service, at the responder (R). A rumor pair is initialized, beginning its trip through the group members, towards R, as summarized in Figure 4.

IN3 Working Paper Series is a monograph series promoted by the Internet Interdisciplinary Institute (IN3) of the UOC
IN3 Working Paper Series (2012) | ISSN 2013-8644 | http://in3wps.uoc

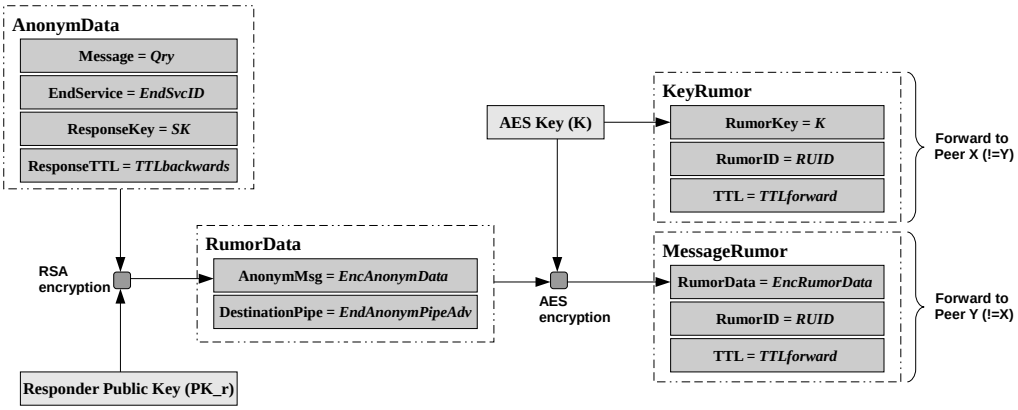


Figure 4. Query setup summary

The detailed process is executed as follows:

1) A local application executing on peer *I* wishes to send a query message *Qry* to an end service executing on peer *R*.
2) Via the Discovery Service, *R*'s Peer Advertisement is retrieved. The following information is looked up in the parameters section (see Figure 2):
   - The end service's Pipe Advertisement, *EndSvcPipeAdv*. Its ID, *EndSvcID*, are extracted.
   - R's public key, *PK_r*.
   - The Anonymity Service's Pipe Advertisement, *EndAnonymPipeAdv*.

3) If *EndSvcPipeAdv* is of the *JxtaBiDiPipe* type, a response will be expected. A secret and secret key *SK* is generated.
4) Two time-to-live values, *TTLforward* and *TTLbackwards*, are established. These values are calculated according to the group size in order to maximize the probability of rumor matching during the random walk phase.
5) An *AnonymData* structure is created as follows:
   - Message = *Qry*
   - EndService = *EndSvcID*
   - ResponseKey = *SK*, if a response is expected. Otherwise, this field is omitted.
   - ResponseTTL = *TTLbackwards*, if a response is expected. Otherwise, this field is omitted.

6) *AnonymData* is encrypted using *PK_r* under a wrapped key scheme, such as in (Staddon J, 1998), generating the bit string *EncAnonymData*.
7) A *RumorData* structure is generated as follows:
   - AnonymMsg = *EncAnonymData*

- DestinationPipe = *EndAnonymPipeAdv*

8) An AES (FIPS, 2001) symmetric key, *K* is dynamically generated.
9) *RumorData* is encrypted using *K*, resulting in the *EncRumorData* bit string.
10) A random JXTA identifier, *RUID* is generated.
11) A message rumor, *MessageRumor*, is created using a JXTA message with the following name-value pairs:
    - RumorData = *EncRumorData*
    - RumorID = *RUID*
    - TTL = *TTLforward*
12) A key rumor, *KeyRumor*, is created using a JXTA message with the following name-value pairs:
    - RumorKey = *K*
    - RumorID = *RUID*
    - TTL = *TTLforward*
13) The Peer Advertisements of two different group members which are executing the Anonymity Service are located. This is achieved by using the Discovery Service to look up peers which include such service in their Peer Advertisement, as previously explained in the Service Publication step.
14) The contained Pipe Advertisements are extracted from each advertisement. *MessageRumor* is forwarded to one pipe and *KeyRumor* to the other.

Once the rumors have been forwarded, if a response is expected, two entries, one for each rumor, are created in *RumorDB*. The RumorID field is assigned *RUID* and the Outbound field is assigned the JXTA ID of the group member used to forward the rumor. In this case, CachedRumor and Inbound are left blank, serving as a special mark indicating that the entries correspond to rumors created at this peer, which is expecting a response. *SK* and a callback mechanism to the end client application are also separately stored.

4.3.2. Random Walk

Whenever the Anonymity Service is executing in a peer, it is always listening to its input pipe, waiting for inbound rumors of any type: message and key. Its basic operation mode is processing inbound rumors received from any peer, Pin, and then immediately forwarding them to a different outbound peer, Pout, acting as a router, immediately updating RumorDB.

The rumor may be processed in two different manners when received, random walk or backtracking, depending on whether an entry already exists in RumorDB for that rumor ID and type, or not. For now, the former case is explained, since it is the way rumors are always processed immediately after query setup. In this scenario, both

rumors are iteratively forwarded through group members until one of them is considered eligible to become the sower, S_i, which will forward the actual query on behalf of I.

1) A rumor is received from peer P_in.
2) The RumorID field value, *RUID*, is extracted from the rumor.
3) The local cache is looked up for a rumor of the same type with a RumorID field matching *RUID*.
4) Random walk must be applied if one of these conditions apply:
   - No enty currently exists for this rumor in the local cache.
   - An entry exists, but the value in the Outbound field is different from P_in's ID.

5) The complementary rumor type, with a RumorID field also matching *RUID*, is looked up in the local cache.
6) If a match exists, the current peer has now a valid message and key rumor pair, *MessageRumor* and *KeyRumor*, and becomes eligible to become a sower. Depending on this fact, an output peer P_out is chosen.
7) The rumor is forwarded to P_out.

If a sower was elected at step 6, the following process is executed in order to decide P_out:

a) The RumorKey field value, K, is extracted from *KeyRumor*.
b) The RumorData field value in *MessageRumor* is decrypted using K. The result is the *RumorData* structure.
c) The DestinationPipe field value, *AnonymSvcPipeAdv*, a pipe advertisement, is extracted from RumorData.
d) The AnonymMsg field value, *EncAnonymData*, is extracted from RumorData.
e) A JXTA message, *EndMessage* is created with the following field name-value pairs:
   - AnonymMsg = *EncAnonymData*
   - RumorID = *RUID*
f) *EndMessage* is forwarded using *EndAnonymPipeAdv*, which points to R's instance of the Anonymity Service. Therefore, P_out = R.

Otherwise, the peer executes the following process in order to decide P_out:

a) The TTL field value is decreased by 1.
b) If TTL = 0, the rumor is discarded. Otherwise it will be forwarded to another peer, P_out.

c) If no entries existed at step 4), a random value different from P_in is chosen for P_out.
d) If other entries already existed, P_out is also chosen at random, but it must be different to every entry's Inbound and Outbound field values.

4.3.3. Response processing

Once the Anonymity Service's instance executing in R receives *EndMessage* from the sower, this step is executed. Given the nature of the random walk process, the possibility exists that multiple peers are eligible to become sowers, and therefore, a few copies of the same request are received. By temporally storing *RUID* for served requests, it is possible to guarantee that, even though several repeated queries are received; only one is actually processed and responded. The rest are discarded.

1) Using the PSE Membership Service, R's secret key, *SK_r* is looked up.
2) The AnonymMsg field value is decrypted using *SK_r*, providing the *AnonymData* = {*Qry, EndSvcID, PKi, TTLbackwardsg*} structure.
3) Using the Discovery Service, R's Peer Advertisement is retrieved. A Pipe Advertisement, *EndSvcPipeAdv*, with an ID equal to *EndSvcID*, is extracted from the service parameters section. This is the end service's advertisement.
4) A connection is opened using *EndSvcPipeAdv* and *Qry* is sent through it.
5) If no response is expected (*EndSvcPipeAdv* is of the *JxtaUnicast* type), the process is over.
6) If a response is expected (*EndSvcPipeAdv* is of the *JxtaBidiPipe* type), the Anonymity Service waits for the response, *Rsp*.
7) At this point, a rumor pair is created, following the same steps 4-13 in Query Setup, with the following changes:
   - The AnonymData structure only has a single field, Message = *Rsp*. It is encrypted using *SK*.
   - The TTL field, in the both the message and key rumor, is initialized to *TTLbackwards*
8) Once the rumors have been created, instead of following two random separate paths, they are both sent back to the sower using the Anonymity Service's pipe

When the sower receives both new rumors, the backtrack process begins.

4.3.4. Backtracking

In the backtrack step, rumors are routed back to the initiator following the inverse path they used to reach the sower, until they arrive to I. Whenever a peer executing the

Anonymity Service receives a rumor, it is able to discern whether a rumor is being backtracked, instead of executing a random walk, in the following manner. When a rumor is received from Pin, it is looked up in the local cache. If an entry already exists and the value in the Outbound field is equal to P_in's ID, backtracking is detected, since this is a circumstance that random walk step ensures will never happen (see step 6). In this scenario, it is worth remarking that it is always guaranteed that an entry exists in a peer's local cache for a given rumor being processed.

1) A rumor of any kind is received from Pin and backtracking is detected.
2) RumorDB is checked for entries with RumorID = *RUID*.
3) The entries Inbound field is checked. If it any of them is blank, then this peer was the initiator.
   a) The rumor is stored in the CachedRumor field.
   b) A rumor of the alternate type but the same ID is looked up. If no match exists, the peer cannot proceed and must wait for it.
   c) If a match exists, the contained response, *Rsp*, is retrieved in a manner similar to the one used by the sower to unbundle the initial rumors, in step 6 of the random walk process. *SK* is used to decrypt *AnonymData*.
   d) The response is pushed to the client application using the stored callback mechanism.

4) Otherwise, the entry with the most recent timestamp is used in the following manner:
   a) The TTL field value in the rumor is decreased by 1. Nevertheless, given then chosen values at the response processing step, it is guaranteed that the rumors will reach the initiator before the TTL field value reaches 0.
   b) The rumor is routed back using the Inbound entry's value. If the peer has gone momentarily offline, the message is held for some time, waiting until it comes back online.
   c) Finally, this rumor's entry is deleted from the local cache, cleaning up.

4.4. Replicated message protocol integration

We have decided to use Hordes as the protocol of choice from the replicated message category, since it provides a nice trade-off between simplicity and efficiency, taking full advantage of multicast communications. Furthermore it is one the few proposals in this category where the authors explicitly quantify the anonymity of their protocol. In fact, from this assessment, it is concluded that the protocol is quite complementary to

IN3 Working Paper Series is a monograph series promoted by the Internet Interdisciplinary Institute (IN3) of the UOC
IN3 Working Paper Series (2012) | ISSN 2013-8644 | http://in3wps.uoc

unimessage protocols, being able to withstand attacks which can be successful on them and vice versa. Therefore, we consider it is an interesting alternative to consider in a JXTA anonymity suite layer. Depending on the kind of attacks a developer is more concerned about, he will be able choose the most convenient anonymity approach. Finally, truth be told, given that the multicast implementation of JXTA is still a bit lacking, we deemed it a bit risky to bet on a protocol which exclusively relies on multicast. Therefore, we considered it would be safer to first work on a hybrid approach.

The protocol is divided in three phases: service initialization, message transmission and message processing and forwarding. Each phase relies on a completely different mechanism to fulfill its intended goal. The transmission phase uses probabilistic forwarding whereas processing and forwarding uses group multicast communications.

4.4.1. Service initialization

Before the anonymity service can be properly used, some previous setup must be executed by the peer group creator. Whenever a Hordes peer group is deployed, a *probability of forwarding* (*pf*) value, that must have a value in the range $0.5 < pf < 1$, is also established by the group creator. This value's utility will be explained at the Message Processing and Forwarding step. Then, the advertisement of this new group, which includes the *pf* parameter, is periodically published using the Discovery Service. Hordes peer groups always rely on the PSE membership Service, to ensure proper cryptographic key initialization.

Peer which join the group, including the creator, must perform two additional steps: initialize a *Forward subset* and subscribe to a multicast group. The Forward subset is a random list of peers used to forward messages to the destination. Each peer has its own Forward subset which is renewed periodically (by default, 24 hours). The Forward subset has to contain only a part of the total horde group members in order to prevent path analysis attacks. The size of this list and the renew time are parameters that can be modified to fine tune the service for different scenarios.

The main characteristic of Hordes is the use of multicast group for response transmission. Our service relies on a premade set of JXTA multicast groups with well-known identifiers and all of these groups have an associated pipe. Peers must subscribe to any of these groups in order to receive responses. It is worth noting that the multicast pipe can be a propagated one, bound to the multicast group, or a JXTA *JxtaMulticastSocket* one, based on multicast socket emulation. In both cases, all the messages sent to this pipe are propagated to all peers that subscribed the pipe group.

The subscribe process to a multicast group is made by choosing it randomly at query transmission time. The peer is unsubscribed from the multicast group as soon as the response is received, or a time out expires. This implies that a peer cannot send

more than one anonymous message at once. Also, hordes recommends to perform the unsubscribe process only after receiving the next message after the one responding the initial anonymous query. This measure avoids false reply attacks, even though it may cause deadlocks if infinite or long timeouts are set. Nevertheless, these limitations can be overcome by implementing a sending/receiving message queue mechanism.

4.4.2. Message transmission

The message transmission phase is executed whenever a peer, the initiator, needs to send an anonymous message to a remote JXTA end-service. Messaging at this stage relies, again, on JXTA's Pipe Service. However, in this particular case, secure pipes are used instead of standard ones, making use of the PSE Membership service' s cryptographic data, such as cipher keys and algorithm type.

The steps to send an anonymous message, *Msg,* using the Hordes approach follows:

1) The initiator locates the destination peer's Peer Advertisement, *DstPeerAdv*. This advertisement also contains the responder's JXTA identifier, *DstId*. The initiator and the responder peer should not be the same.
2) From within *DstPeerAdv*, the chosen end-service Pipe Advertisement, *SvcPipeAdv*, is extracted.
3) A random message identifier, *MsgId*, is generated to uniquely identify the message for its life cycle period. Hordes recommends a length of at least 128 bits in order to avoid collisions.
4) One of the available JXTA multicast channels in the hordes group, with the identifier *MchId*, is randomly chosen. The peer joins to the chosen multicast group.
5) A message, *HordesMsg*, is constructed, following the Hordes protocol parameters, under the JXTA message name-value pair syntax. It contains the following fields:
   - tagDest = *DstId*
   - tagMcast = *MchId*
   - tagId = *MsgId*
   - tagMsg = *Msg*, structured using a syntax as expected by the end-service.
   - tagPipe = *SvcPipeAdv*.
6) The sender randomly chooses a peer from its Forward Subset list of peers (see section 4.4.1), *fwdPeer*.
7) *fwdPeer*'s Peer Advertisement is retrieved. The protocol's Pipe Advertisement is extracted from it.

IN3 Working Paper Series is a monograph series promoted by the Internet Interdisciplinary Institute (IN3) of the UOC
IN3 Working Paper Series (2012) | ISSN 2013-8644 | http://in3wps.uoc

8) Using the Pipe Advertisement, a connection to *fwdPeer*'s anonymous service instance is established and *HordesMsg* is forwarded.
9) A timeout counter is started, and the sender blocks until a message is received from to the multicast group or the timeout ends. The message exchange is only successful in the former case. The received multicast message has the following JXTA message structure:
   - tagMsgId : *MsgId*.
   - tagMsgData : The reply message generated by the end-service.

Whenever a message is received through a subscribed multicast group, the value of its tagMsgId field of any message received via the multicast group is compared with any identifier generated. If a match is produced, it means that the reply has arrived and the operation is over. If the timeout expires and the reply has not arrived, it implies that the message has been a lost. At this point a new query can be sent relying on this multicast group.

4.4.3. Message processing and forwarding

This procedure is activated automatically when the anonymous service receives a forwarded message from any other peer of the horde group (be it the original sender or a peer from the same Forwarding subset). Any peer that has joined to the horde group and belongs to some other peer's Forwarding subset may receive messages and has to process them. The processing mechanism follows:

1) The probability of forwarding (*pf*) local configuration parameter is retrieved.
2) Generate a random number (n) in the range: $0 < n < 1$.
3) The forward condition is checked: $n < pf$.
4) If the forward condition is true the message is forwarded to another peer in the Forward Subset. The peer randomly chooses another peer from its Forward Subset list of peers. and acts like the original sender in the Message Transmission procedure, Steps 6-8.
5) If the forward condition is false the message is not forwarded, but finally sent to the end-service. Extracting the information from inside the hordes message, the peer:
   a) Tries to contact with the destination peer (tagDest) and opens the a channel to the end-service's pipe (tagPipe).
   b) Consumes the service by sending the actual message (tagMsg) to the pipe and waiting for the reply from this pipe.
   c) Subscribes to the multicast group (tagMcast) and creates the multicast message, as expected in the Message Transmission procedure, Step 9.

IN3 Working Paper Series is a monograph series promoted by the Internet Interdisciplinary Institute (IN3) of the UOC
IN3 Working Paper Series (2012) | ISSN 2013-8644 | http://in3wps.uoc

d) The multicast message is sent to the multicast group, awaking all subscribed peers that are waiting for a response using this the multicast group.

One of the main differences between the returning multicast mechanism and the forwarding mechanism is that, according to the Hordes protocol, in the former, the message is sent in plain text, without encrypting it. This means that any peer that is subscribed to the multicast group will get the message identifier and the reply to the message and can understand it. However, knowing this information does not reveal the identity of the source or destination peer, guaranteeing anonymity. This is the actual behavior as intended by the original protocol. Nevertheless it is not difficult to add privacy to the response.

5. Conclusions and further work

We have proposed an anonymity layer for JXTA which relies in a set of protocols, instead of a single one, in order to provide the ability to choose the most suitable one depending on the particular application scenario or the network's properties at each time. The protocol adaptation is tightly integrated to JXTA's architecture, working only within the context of its core services' operation. Thus, it has not been necessary to define new protocols or primitives aside from the ones already available in JXTA. A further advantage of this is that pipe and cryptographic data publication seamlessly meshes within its standard presence mechanism. Thus, the anonymity layer is almost invisible to the actual service being accessed. From the end service provider's standpoint, the original message has been normally received through its published input pipe. In addition, we propose some tweaks and improvements over the original protocol, which we hope increase its overall performance.

At present time, we have almost completed the implementation and in short will be able to study its behavior in real JXTA networks and compare our results with the ones presented in each protocol's original proposals. A quantitative analysis of each protocol operating under a JXTA network will allow us to assess its ideality in different scenarios, as well as fine-tune some of the protocol parameters or design decisions. The results of some of our preliminary work on that regard have been included in Annex I of this document. For example, right now, the use of JXTA secure sockets seems to impact its performance, showing that some ideas that look good on paper, or when simulated, do not behave as expected when implemented, interacting with other systems. Thus, experimental evaluation on real systems becomes important.

IN3 Working Paper Series is a monograph series promoted by the Internet Interdisciplinary Institute (IN3) of the UOC
IN3 Working Paper Series (2012) | ISSN 2013-8644 | http://in3wps.uoc

Finally, such quantitative analysis also becomes valuable in order to establish a valid comparison between the chosen protocols on equal grounds. In the cases where a quantitative analysis exists, the differing methods (simulation based on logs or theoretical models, experiments using filesharing middlewares, instead of service oriented ones, etc.) make it difficult to make a clear comparison between each protocol category. Therefore, the only way to be sure which protocol would better suit each scenario in a JXTA network is by actual experimentation under this particular middleware.

Bibliographic references

Bhatti S. ;Rogers M (2007) "How to disappear completely: A survey of private peer-to-peer networks", in Proceedings of International Workshop on Sustaining Privacy in Autonomous Collaborative Environments.

Bian J., Seker R.(2009), "Jigdfs: A secure distributed file system", in IEEE Symposium on Computational Intelligence in Cyber Security, pp. 76 – 82.

ConneX (2007), "Connex Project Homepage", http://connex.sourceforge.net [Date of query: June 24th, 2012]

Desmedt Y.G.; Frankel Y. (1989) "Threshold cryptosystems", in CRYPTO '89: Proceedings on Advances in cryptology, pp. 307–315.

Dolev S.; Ostrobsky R. (2000) "Xor-trees for efficient anonymous multicast and reception", ACM Trans. Inf. Syst. Secur., vol. 3, pp. 63–84.

FIPS Federal Information Processing Standard, "Advanced encryption standard (AES)", 2001.

Freedman M.J.; Dingledine R.; Molnar D. (2000), "The free haven project: Distributed anonymous storage service In *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*.

Goldsclag D.; Syverson P.; Reed M. (1997) "Anonymous connections and onion routing", Proceeding of the IEEE 18th Annual Symposium on Security and Privacy, pp. 44–54.

Han J.; Liu Y. (2008) "Mutual anonymity for mobile p2p systems", IEEE Transactions on Parallel and Distributed Systems, vol. 19, no. 8, pp. 1009 –1019.

Hansen M.; Tschofenig H. (2011) "Terminology for Talking about Privacy by Data Minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management", http://tools.ietf.org/html/draft-hansen-privacy-terminology-02 [Date of query: June 24th, 2012]

Levine B.N., Shields C. (2002) "Hordes: A multicast based protocol for anonymity", Journal of Computer Security, vol. 10, no. 3, pp. 58–70.

Liu Y.; Han J.; Wang J. (2011) "Rumor riding: Anonymizing unstructured peer-to-peer systems", Transactions on Parallel and Distributed Systems, IEEE, vol. 22, no. 3, pp. 464–475.

Mathewson N.; Dingledine R.; Syverson P. (1998) "Tor: The second generation onion router", Proceeding of the 13th USENIX Security Symposium, pp. 303–320.

Matsuo K., Barolli L., Arnedo-Moreno J., Xhafa F., Koyama A., and Durresi A. (2009), "Experimental results and evaluation of smartbox stimulation device in a P2P e-learning system", in Proceedings of the 12th International Conference on Network-Based Information Systems, pp. 37–44.

Lele N.; Wu L.; Akavipat R.; Menczer F. (2009), "Six-search.org 2.0 peer application for collaborative web search", in Proceedings of the 20th ACM conference on Hypertext and hypermedia, pp. 333–334.

Project Kenai (2011), "JXTA: The Language and Platform Independent Protocol for P2P Networking", http://jxta.kenai.com [Date of query: June 24th, 2012]

Rabin M. O. (1989, "Efficient dispersal of information for security, load balancing, and fault tolerance", Journal of the ACM, vol. 36, pp. 335–348

Ren-Yi X (2008), "Survey on anonymity in unstructured peer-to-peer systems", Journal of Computer Science and Technology, vol. 23, no. 4, pp. 660–671.

Rubin A.D.; Meiter M.K. (2004) "Crowds: Anonymity for web transactions", ACM Transactions on Information and System Security, vol. 1, no. 1, pp. 66–93.

Sherwood R.; Bhattacharjee B.; Srinivasan A., "P5: A protocol for scalable anonymous communication", in PROC. IEEE SYMP. SECURITY AND PRIVACY, pp. 58–70.

Shields C.; Scarlata V.; Levine B.N. (2001) "Responder anonymity and anonymous peer-to-peer file sharing", Proceeding of the ACM CCS, pp. 17–26.

Staddon J.; Kaliski B. (1998) "PKCS1: RSA Cryptography Specifications. Version 2.0", 1998.

Syverson P.Tsudik G.;Reed M.;Landwher C. (2001) "Towards an analysis of onion routing security", Designing Privacy Enhancing Technologies, LNCS, vol, 2009 pp. 96-114.

Wallace J. D. (1999), "Nameless in cyberspace: Anonymity on the internet", http://www.cato.org/pubs/briefs/bp-054es.html [Date of query: June 24th, 2012]

Aters R. B.; Felten E. W.; Sahai A. (2003) "Receiver anonymity via incomparable public keys", in The 2003 ACM Conference on Computer and Communications Security, pp. 112–121.

Zhang X. Xiao L., Xu Z. (2003) "Low-cost and reliable mutual anonymity protocols for peer-to-peer networks", IEEE Transactions on Parallel and Distributed Systems, vol. 14, no. 9, pp. 829–840.

Arnedo-Moreno J., Pérez-Gilabert N., Domingo-Prieto M (2012, "Experimental evaluation of anonymous protocols under the JXTA middleware", in Proceedings of the 15th International Conference on Network-Based Information Systems (NBIS'12), pp. 53-57.

Annex I: Preliminary results

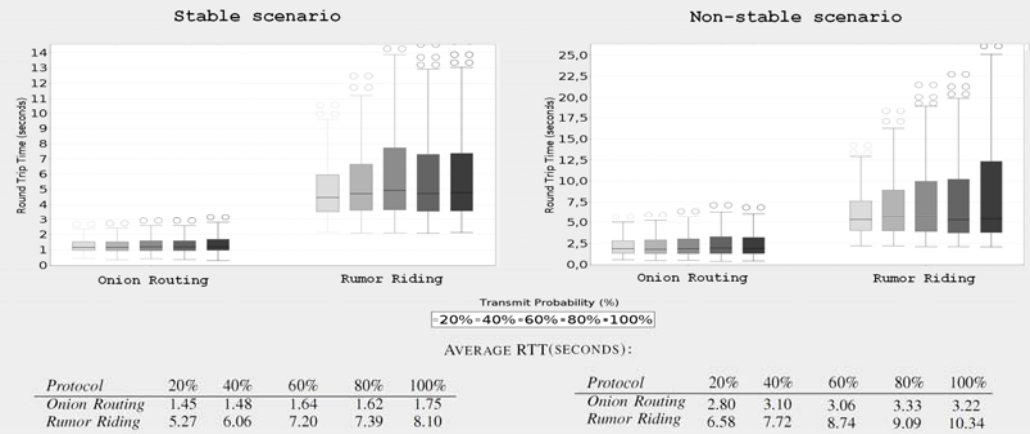
The preliminary results in assessing the protocol performance were published in (Arnedo-Moreno *et al*, 2012). In this work, we used the proposed protocol adaptation to compare the unimessage (onion routing) and split-message (rumor riding) based approaches on an even field: the JXTA middleware. The comparison is based on three metrics: Round Trip Time (RTT), Processing Time and Reliability. For convenience, such results have been included in this work.

An experiment was executed twice for each protocol, each time relying on a different scenario. On one hand, a stable one, where peers rarely disconnect from the network, or when they do, it is in an orderly manner. On the other hand, we evaluated a non-stable, free for all, scenario, where no assumptions can be made about node stability. It must be noted that in this second scenario, the stability of nodes simply cannot be guaranteed, but they are not purposefully unstable.

Each experiment was divided into a set of 10 individual tests. For each test, a JXTA peer group, composed by 32 peers, was created, in such a manner that individual peers were deployed in separate nodes at random locations. Such number of peers was chosen according to the typical maximum value for peer group size in a JXTA network. Using this approach, we could test the behavior of the peer group under different network configurations, using different sets of nodes. Anonymous requests between peer group members were randomly generated according to a parameter p, with a value between 0 and 1. Each individual test in an experiment was divided in 250 time slots, each slot 5 seconds long. During a time slot, peers had a p probability to actually send a request to a random node. By increasing and decreasing the value of p, we could increase and decrease the expected amount of requests in the network at a given time, but still maintaining a random aspect, instead of generating them at a fixed predictable rate. Each test is considered concluded when the last response is actually received (which may happen at some time after the 250th slot). We ran experiments for values of p = 0.2, 0.4, 0.6, 0.8 and 1.0.

Given this experiment configuration, a total of 200 tests were executed (2 protocols * 2 scenarios * 5 transmit probabilities * 10 network layouts). After its execution, the data of about 200000 messages was recollected for each scenario and used to analyze the RTT and reliability metrics. As far as the analysis of processing time is concerned, the amount of data for each case was in the range of about 2-6 million individual measurements, depending on the scenario and protocol.

The Round Trip Time (RTT), Processing Time and Reliability results are shown in Figures 5, 6 and 7, respectively. A more in depth analysis of these results can be found in the original publication.

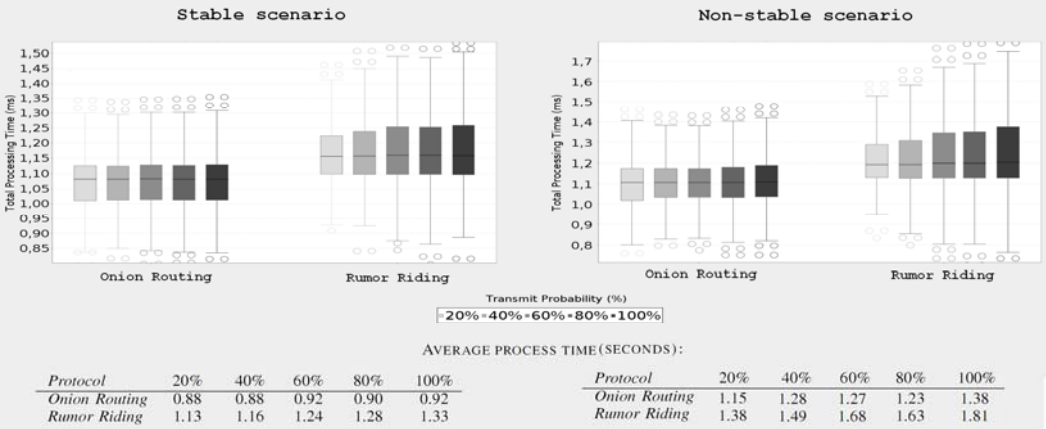


AVERAGE RTT(SECONDS):

Protocol	20%	40%	60%	80%	100%
Onion Routing	1.45	1.48	1.64	1.62	1.75
Rumor Riding	5.27	6.06	7.20	7.39	8.10

Protocol	20%	40%	60%	80%	100%
Onion Routing	2.80	3.10	3.06	3.33	3.22
Rumor Riding	6.58	7.72	8.74	9.09	10.34

Figure 5. Round Trip Time experimental results

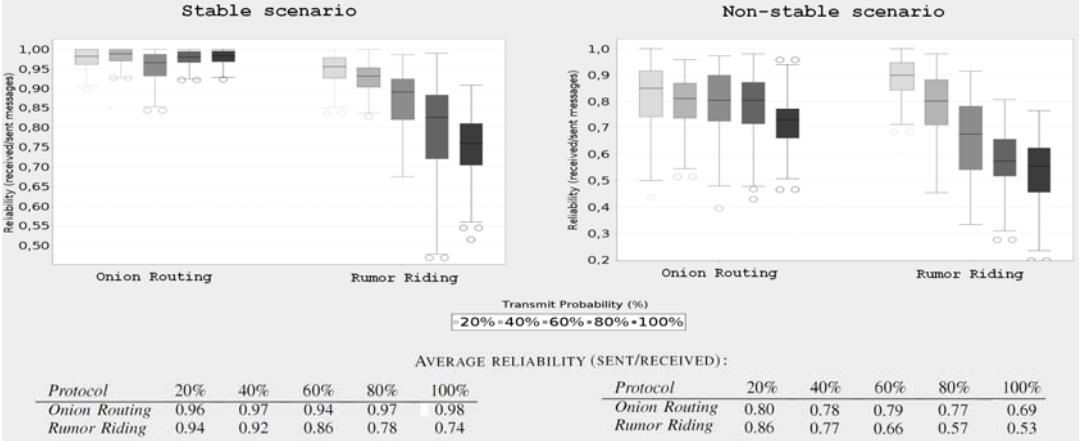


AVERAGE PROCESS TIME(SECONDS):

Protocol	20%	40%	60%	80%	100%
Onion Routing	0.88	0.88	0.92	0.90	0.92
Rumor Riding	1.13	1.16	1.24	1.28	1.33

Protocol	20%	40%	60%	80%	100%
Onion Routing	1.15	1.28	1.27	1.23	1.38
Rumor Riding	1.38	1.49	1.68	1.63	1.81

Figure 6. Processing Time experimental results



AVERAGE RELIABILITY (SENT/RECEIVED):

Protocol	20%	40%	60%	80%	100%
Onion Routing	0.96	0.97	0.94	0.97	0.98
Rumor Riding	0.94	0.92	0.86	0.78	0.74

Protocol	20%	40%	60%	80%	100%
Onion Routing	0.80	0.78	0.79	0.77	0.69
Rumor Riding	0.86	0.77	0.66	0.57	0.53

Figure 7. Reliability experimental results

Resumen

JXTA es una especificación abierta de protocolos peer-to-peer (P2P) que, en sus cerca de 10 años de historia, ha evolucionado lentamente para abarcar una amplia gama de aplicaciones. Como parte de este proceso, algunas mejoras largamente esperadas dentro del ámbito de la seguridad se han incluido en las versiones más recientes. Estas proporcionan un nivel de seguridad en sus protocolos que permiten, como mínimo, alcanzar un grado aceptable de privacidad y autenticación. Sin embargo, en algunos contextos, hay que cumplir requisitos de seguridad más avanzados, como el anonimato. Aunque existen diversas aproximaciones para implementar el anonimato en las redes P2P, no hay una solución perfecta que sea capaz de abarcar a todos los escenarios posibles. En este trabajo, proponemos la forma de adaptar la mensajería de JXTA para que sus servicios puedan acceder de forma anónima, haciendo un máximo uso y aprovechamiento de las particularidades y capacidades de su arquitectura, de manera que sea completamente invisible a los protocolos existentes. Para conseguir este fin, proponemos una capa multi-protocolo en el anonimato, por lo que se puede elegir en cada momento el más adecuado en función del escenario a tratar.

Palabras Clave
peer-to-peer, seguridad, anonimato, JXTA, Java.

Resum

JXTA és una especificació oberta de protocols peer-to-peer (P2P) que, en els seus prop de 10 anys d'història, ha evolucionat lentament per abarcar una àmplia gamma d'aplicacions. Com a part d'aquest procés, algunes millores llargament esperades dins l'àmbit de la seguretat s'han inclòs en les versions més recents. Aquestes proporcionen un nivell de seguretat en els seus protocols que permeten, com a mínim, assolir un grau acceptable de privadesa i autenticació. No obstant això, en alguns contextos, cal complir requisits de seguretat més avançats, com ara l'anonimat. Tot i que existeixen diverses aproximacions per implementar l'anonimat a les xarxes P2P, no hi ha una solució perfecta que sigui capaç d'abastar a tots els escenaris possibles. En aquest treball, proposem la forma d'adaptar la missatgeria de JXTA perquè els seus serveis puguin accedir de forma anònima, fent un màxim ús i aprofitament de les particularitats i capacitats de la seva arquitectura , de manera que sigui completament invisible als protocols existents. Per aconseguir aquest fi, proposem una capa multi-protocol en l'anonimat, de manera que es pot triar en cada moment el més adequat en funció de l'escenari a tractar.

Keywords
Peer-to-peer, seguretat, anonimat, JXTA, Java

Joan Arnedo-Moreno
jarnedo@uoc.edu
Researcher at IN3
Universitat Oberta de Catalunya (Spain)

Joan Arnedo-Moreno is a lecturer at Estudis d'Informàtica, Multimimèdia i Telecomuncicació in the Open University of Catalonia (UOC) and works as a part-time assistant at the Universitat Politènica de Catalunya (UPC). From the latter, he earned his degree in Computer Science in 2002 and his Ph.D. degree in 2009. He has published several papers in international conferences and journals and has been invited as keynote speaker at several conferences. Both his teaching and research interests are related to the fields of networking and security, more specifically in peer-to-peer systems.

Marc Domingo-Prieto
mdomingopr@uoc.edu
Research Collaborator
Universitat Oberta de Catalunya (Spain)

Marc Domingo-Prieto holds the degree of Computer Systems and the master in Computer Architecture, Networks and Systems from Universitat Politècnica de Catalunya (UPC). He is doing his PhD at Worldsensing company about wireless sensor networks (WSN) communications security. Also, he is a research assistant in the K-ryptography and Information Security for Open Networks (KISON) research group in the Open University of Catalonia (UOC). His research interests include scalable distributed algorithms and applications, energy efficient and security in WSN, mobile and peer-to-peer applications.

Noemí Pérez-Gilabert
nperezg@uoc.edu
Research Assistant at IN3
Universitat Oberta de Catalunya (Spain)

Noemí Pérez-Gilabert holds the degree of Computer Systems from University of Catalonia (UOC) and she is a research assistant in the K-ryptography and Information Security for Open Networks (KISON) research group in the Open University of Catalonia (UOC). Also, she works as a freelance programmer with experience in several technologies such as J2EE, PHP and .NET, developing different web-based applications operating on Weblogic, J-Boss and Apache servers.