Project JXTA 2.0 Super-Peer Virtual Network

Bernard Traversat, Ahkil Arora, Mohamed Abdelaziz,
Mike Duigou, Carl Haywood, Jean-Christophe Hugly, Eric Pouyoul, Bill Yeager

Bernard.Traversat@Sun.Com
Project JXTA
Sun Microsystems, Inc.

Abstract

The Project JXTA protocols establish a *virtual network* overlay on top of the Internet, allowing peers to directly interact and self-organize independently of their network connectivity and domain topology (firewalls or NATs). Project JXTA enables application developers, not just network administrators to design network topology that best match their application requirements. Multiple ad hoc virtual networks can be created and dynamically mapped into one physical network unleashing a richer multi-dimensional virtual network world. Project JXTA is looking ahead where billion of network services, all addressable on the network will be able to discover and interact with each other in an ad doc and decentralized manner through the formation of a multitude of virtual networks. The following paper describes the Project JXTA 2.0 implementation that was recently released by the JXTA community. The JXTA 2.0 implementation introduces a number of new features for improving overall functionality, performance, and scalability of the JXTA network. The JXTA 2.0 release is making a stronger differentiation in the way JXTA super-peers (relay and rendezvous) behave and interact with edge peers. The JXTA 2.0 implementation introduces the concept of a rendezvous peer view to connect rendezvous within a peergroup. Resolver queries are no more propagated to edge peers as in JXTA 1.0 reducing overall network traffic. Edge peers use a loosely-coupled distributed hash index to locate advertisements on the rendezvous peer view for efficient query lookups. The JXTA 2.0 implementation provides better resource management (threads and queues), and implements resource usage limits to fairly allocate resource between platform services. The JXTA 2.0 release also introduces new implementations of the TCP/IP and HTTP transports. The TCP/IP transport now takes advantage of bi-directional physical connections to limit the number of connections. Finally, the JXTA 2.0 implementation supports TCP/IP relay for efficient traversal of NATs.

1. 1. Introduction

The open-source Project JXTA [1,2,3] is the industry leading peer-to-peer (P2P) platform originally conceived by Sun Microsystems Inc. and designed with the participation of a small but growing number of experts from academic institutions and industry. The Project JXTA protocols establish a virtual network overlay on top of the Internet and non-IP networks, allowing peers to directly interact and self-organize independently of their network connectivity. Today more than 90 projects are hosted on the JXTA open-source website. More than 1 million downloads of the technology have been made. ISVs are developing and shipping JXTA technology-based applications to a variety of markets including knowledge management, content sharing, and collaborative applications. Enterprises, government agencies, and educational institutions are making plans to adopt the technology as their primary P2P technology solution. Project JXTA standardizes a common set of protocols for building P2P virtual networks. The JXTA protocols defines the minimum required network semantic for peers to form and join a virtual network. The Project JXTA protocols define a generic network substrate usable to build a wide variety of P2P networks. Project JXTA enables application developers, not just network administrators to design network topology that best match their application requirements. Multiple ad hoc virtual networks can be created and dynamically mapped into one single physical network. Project JXTA envisions a world where each peer, independent of software and hardware platform, can benefit and profit from being connected to millions of other peers through the formation of a multitude of virtual networks. Project JXTA is designed to be independent of programming languages (such as C or the Java™ programming language), system platforms (such as the Microsoft Windows and UNIX® operating systems), service definitions (such as RMI and WSDL), and network protocols (such as TCP/IP or Bluetooth). The Project JXTA protocols have been designed to be implementable

on any device with a network heartbeat, including sensors, consumer electronics, PDAs, appliances, network routers, desktop computers, data-center servers, and storage systems. The initial reference implementation of the JXTA 1.0 protocols was released on April 2001, the following paper describes the JXTA 2.0 reference implementation that was recently released by the JXTA community. The Project JXTA 2.0 reference implementation introduces a number of new features to improve overall performance and scalability of the JXTA network. The JXTA 2.0 release is making a stronger differentiation in the way super-peers such as relay and rendezvous behave and interact with edge peers. The JXTA 2.0 release introduces the concept of a *rendezvous peer view* (RPV) to propagate resolver queries, and a *shared resource distributed index* (SRDI) to index advertisements on the rendezvous peer view for efficient advertisement query lookups. The JXTA 2.0 implementation introduces the concept of pluggable *walkers* to propagate queries within the rendezvous RPV network. The JXTA 2.0 implementation is adding better resource management (threads and message queues), and implements resource usage limits to fairly allocate resource between platform services. The JXTA 2.0 implementation is creating thread pools, and thread limits to control thread usage on a per-service based. The JXTA 2.0 implementation is also adding optimizations for reducing extra message buffering and copying when sending messages. The endpoint service minimizes the number of active connection threads required to support large number of peers connecting to a single peer. Peers establish a connection, get their addressed messages, and then remain quiescent until another message is available or the connection is closed. The Project JXTA 2.0 implementation adds support for TCP/IP relays to enable efficient NAT traversals. Finally, the JXTA 2.0 implementation provides dynamic failover from rendezvous and relay super-peers to recover from super-peer failures, or roam for improving connectivity or functionality.

Section 2. gives an overview of Project JXTA virtual network. Section 3. discusses the rendezvous super-peer network, the rendezvous peer view, the shared resource distributed index service, and resolver query propagation. Section 4. covers the relay super-peers, routing, firewall and NAT traversals. Section 5. discusses peergroups, and virtual secure domains. Section 6. describes the JXTA pipe abstraction. Section 7. outlines JXTA security model. Section 8. discusses the JXTA core specification and standard service protocols. Section 9. describes the JXTA 2.0 reference implementation overall architecture. Finally, section 10. concludes on the state of the JXTA 2.0 implementation and future directions.

2. Project JXTA Virtual Network

The Project JXTA protocols create a *virtual network* overlay on top the existing physical network infrastructure. The Project JXTA virtual network allows a peer to exchange messages with any other peers independently of its network location (firewalls, NATs or non-IP networks[6]). Messages are transparently routed, potentially traversing firewalls or NATs, and using different transport/transfer protocols (TCP/IP, HTTP) to reach the receiving peers (see figure 1). The Project JXTA network allows peers to communicate without needing to understand or manage complex and changing physical network topologies allowing mobile peers to move transparently from on location to another. The Project JXTA virtual network standardizes the manner in which peers discover each other, self-organize into peergroups, discover peer resources, and communicate with each other. The Project JXTA 2.0 implementation builds upon the 5 virtual network abstractions introduced in JXTA 1.0 [1]. First, a *logical peer addressing* model that spans the entire JXTA network. Second, *peergroups* that let peers dynamically self-organize into protected virtual domains. Third, *advertisements* to publish peer resources (peer, peergroup, endpoint, service, content). Fourth, a universal binding mechanism, called the *resolver*, to perform all binding operations required in a distributed system. Finally, *pipes* as virtual communication channels enabling applications to communicate between each other. In the following sections, we discuss these abstractions and enhancements made in JXTA 2.0.

2.1 JXTA IDs

The Project JXTA addressing model is based on an uniform and location independent logical addressing model. Every network resource (peer, pipe, data, peergroup, etc.) is assigned a unique *JXTA ID*. JXTA IDs are abstract objects enabling multiple ID representations (IPv6, MAC) to coexist within the same JXTA network. The reference implementation is using 128-bit random UUIDs allowing each peer to self-generate its own IDs. A peer in the JXTA network is uniquely identified by its *peer ID* allowing the peer to be addressed independently of its

physical address (see figure 1). For instance, a laptop booting via DHCP, and therefore having many different IP addresses overtime, will always have the same peer ID. Similarly, a peer supporting multiple network interfaces (Ethernet, Wi-Fi, etc.) will be addressed as a single peer independently on the interface used. The peer ID abstraction allows a peer to encapsulate not just physical transports, but also “logical” transport protocols such as HTTP or TLS. JXTA logical addressing model introduces a fundamental indirection separating the identification of a resource and the location of a resource allowing a variety of virtual mappings to be used to determine the physical location of a resource.

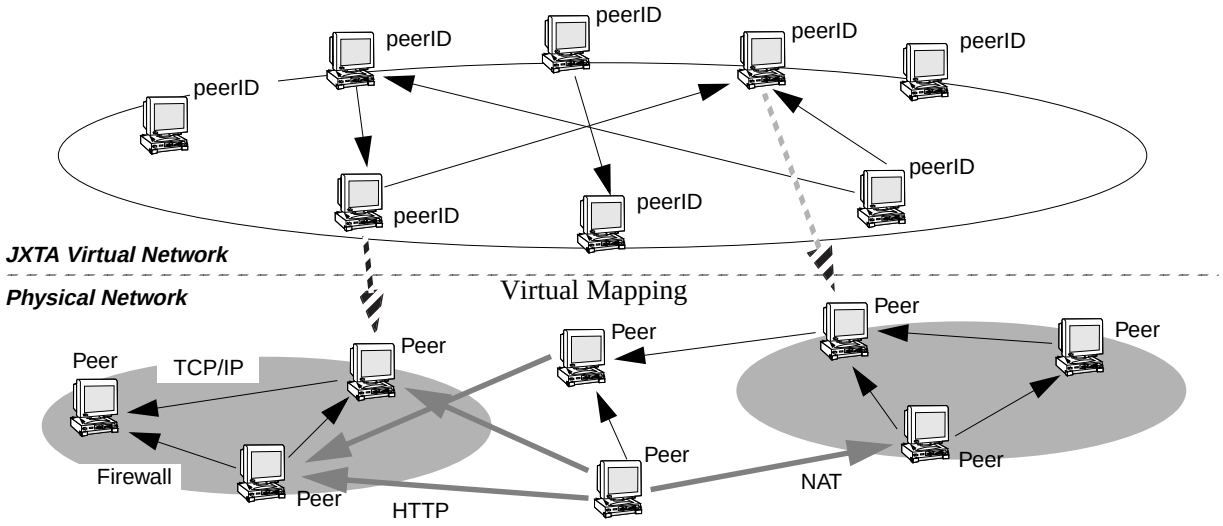


Figure 1: The Project JXTA Virtual Network

A peer *endpoint* is associated with each Peer ID encapsulating all available physical network addresses for a peer. Peer endpoints are very much like business cards, listing the many ways to contact a person (phone, fax, email address, etc.). When receiving a peer endpoint advertisement, another peer can select the most efficient way to communicate with that peer from the list of available peer endpoint addresses (using TCP/IP directly when possible, or HTTP over TCP/IP if one must traverse firewall).

2.2 Advertisements

All network resources in the Project JXTA network, such as peers, peergroups, pipes, and services are represented by *advertisements*. Advertisements are language-neutral metadata structures resource descriptors represented as XML documents. Project JXTA standardizes advertisements for the following core JXTA resource: peer, peergroup, pipe, service, metering, route, content, rendezvous, peer endpoint, transport. Developers can subtype these advertisements to create their own subtypes to add an unlimited amount of additional and richer metadata information to each resource description. For example a Web service advertisement will contain the associated WSDL document associated with the service. Advertisements can be used to virtually describe anything: source code, script, binary, classes, compiled JIT code, Java objects, EJB, J2EE containers.

Figure 2 shows an example of a *PeerGroup Advertisement* (jxta:PGA) that describes a peergroup via a unique peergroup ID (GID), a module specification ID (MSID) that points to an advertisement that describes all of the peergroup services available in this peergroup, a peergroup name (Name), and a description of the peergroup (Desc). Peers cache, publish, and exchange advertisements to discover and find available network resources. Peers discover resources by searching for their associated advertisements. All advertisements are published with

a *lifetime* that specifies the duration of that advertisement in the network. An advertisement can be republished at anytime to extend its lifetime before it expires. Peers exchange advertisements, but maintain their expiration date during exchanges. Lifetimes permit to purge expired advertisements without centralized management.:

```
<?xml version="1.0"?>
<!DOCTYPE jxta:PGA>
<jxta:PGA xmlns:jxta="http://jxta.org">
    <GID> urn:jxta:jxta-NetGroup</GID>
    <MSID>urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE000000010206</MSID>
    <Name>NetPeerGroup</Name>
    <Desc>NetPeerGroup by default</Desc>
</jxta:PGA>
```

Figure 2: Example of a Peergroup Advertisement

3. Rendezvous Super-Peers

The Project JXTA network uses an universal resource binding mechanism called the *resolver* to perform all resolution operations found in traditional distributed systems, such as resolving a peer name into an IP address (DNS), binding a socket to a port, locating a service via a Directory service (LDAP), or searching for content in a distributed filesystem (NFS). In Project JXTA, all resolution operations are unified under the simple discovery of one or more advertisements. The Project JXTA protocols do not specify how the search of advertisements is performed, but provide a generic resolver protocol framework with a default policy that can be overwritten. Developers can tailor their resolver implementations to use a decentralized, centralized, or hybrid approach to match their application domain requirements. The core resolver infrastructure provides the ability to send and propagate queries, and receive responses. The resolver performs authentication and verification of credentials, and drops invalid messages.

The Project JXTA network provides a default resolver policy based on *Rendezvous* super-peers. Rendezvous are peers that have agreed to cache *advertisement indices* (i.e. pointers to edge peers that cache the corresponding advertisement). Rendezvous conceptually corresponds to well known locations used for indexing and locating advertisements. The default rendezvous policy provides the minimum infrastructure for efficiently bootstrapping high-level advertisement search policies.

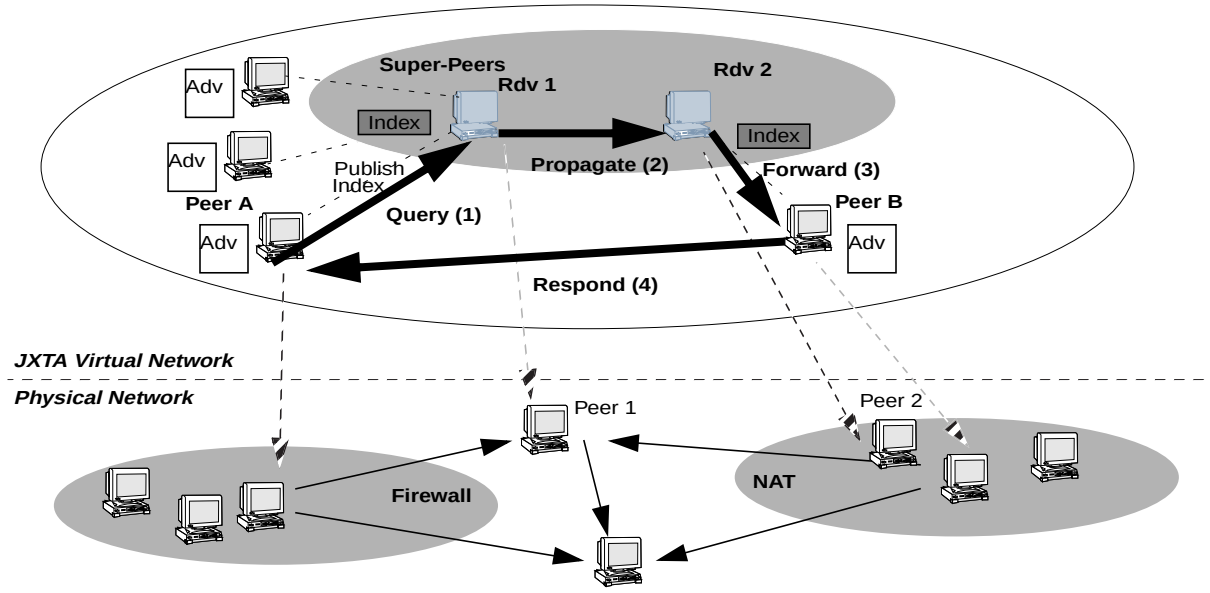


Figure 3: Project JXTA Rendezvous super-peers

For instance, via rendezvous, a peer can find enough peers to bootstrap a more advanced search policy. High-level search services are expected to provide more efficient search mechanisms, because they may have a better knowledge of the content topology distribution (CAN[4] or Chord[5], Semantic Web[7]). The default rendezvous peer infrastructure provides a low-level discovery mechanism to discover advertisements while providing hooks to allow high-level discovery services to participate in the advertisement search process.

Edge peers maintain special relationship with their rendezvous peers. Any peer can become a rendezvous peer assuming it has the right credentials. Secure peergroups may restrict which edge peers can act as rendezvous. A fundamental change made in JXTA 2.0 is rendezvous do no cache edge peer advertisements. Rendezvous just maintain an index of advertisements published by their edge peers. Not caching advertisements make the rendezvous architecture more scalable, and reduce the problem of caching out-of-date advertisements. The *Shared-Resource Distributed Index (SRDI)* service is used by edge peers to index their advertisements on rendezvous. Edge peers use SRDI to push advertisement indices to their rendezvous when publishing new advertisements. Indices can be pushed synchronously when a new advertisement is published, or asynchronously by the SRDI daemon that runs at fixed intervals. For example in figure 3, both Peer A and Peer B are pushing indices of their advertisements to their respective rendezvous Rdv1, and Rdv2. When an advertisement query is issued from Peer A for an advertisement stored on Peer B, the query is sent to Peer A's rendezvous Rdv1 (1), Rdv1 looks if it has an index of that advertisement. If it does not find an index, it propagates the query (2) to the next rendezvous Rdv2. We will discuss shortly the propagation policy used by rendezvous to forward queries between themselves. When the query reaches Rdv2, Rdv2 finds the index for the advertisement and forwards the query to Peer B (3). This is done to ensure that the latest copy of the advertisement on Peer B will be sent to Peer A. When Peer B receives the query, it sends the advertisement to Peer A (4). It is important point out that any peers independently of its physical location can act as a rendezvous. In figure 3, the physical Peer2 is behind NAT and acting as a rendezvous.

Another significant improvement made in JXTA 2.0 is advertisement queries are no more propagated to edge peers. Only rendezvous peers are involved in the propagation of advertisement queries. A query is only forwarded to an edge peer when a matching index has been found. This is significantly reducing network traffic when looking for an advertisement. The next section discusses how queries are propagated between rendezvous super-peers.

3.1 A Loosely-Consistent Rendezvous Network

Rendezvous super-peers organize themselves into a loosely-coupled network. This is done due to the relative high churn rate predicted for ad hoc JXTA networks. In a fluctuating network, it is difficult to maintain a consistent view of all peers in a peergroup without assuming a small peergroup or some tightly-coupled centralized membership structure that maintains the list of all present peers. Maintaining a consistent view becomes even more difficult if the size of the peergroup and the number of peers increases [4] to an extremely large number (1,000,000 peers). While rendezvous in an enterprise P2P networks may show higher stability, rendezvous in consumer P2P networks (PDA, phones) are likely to be less stable and show a higher churn rate. JXTA assumptions differ with common Distributed Hash Table (DHT) approaches such as Brocade[8], Chord[5] or CAN[4] that assume a relatively stable peer infrastructure where a distributed consistent view can be maintained with minimum overhead. Our assumptions are closer to the random walker approach for unstructured networks proposed in [9]. When deploying a P2P network not all entities may be willing to cover the extra costs of deploying dedicated infrastructure peers (super-peers) to support their network. This economical factor and the need to enable self-supporting P2P networks made us thrive toward a fully ad hoc and unstructured approach allowing potentially any peers to become a super-peer. It is important to point out this loosely-coupled unstructured policy is just the default policy. Peergroup creators have the ability to overwrite the default policy and define their own policies.

In a highly fluctuating and unpredictable environment, the cost of maintaining a consistent distributed index is likely to out weight the advantages of having one. We likely will spend most of our time updating indices than performing lookups (i.e index trashing). One can separate the cost of a DHT into index maintenance, and index lookup. DHT approaches provide the most efficient index lookup mechanism log(N), where N is the number of peers. However, there is a maintenance cost which typically grows exponentially as the peer churn rate increases

[9]. On the other hand, not having a DHT means that an expensive exhaustive network crawling needs to be performed leading to even worst network traffic.

Project JXTA 2.0 proposes an hybrid approach that combines the use of a loosely-consistent DHT with a limited-range rendezvous walker to garbage collect out-of-sync indices. Rendezvous peers are not required to maintain a consistent distributed hash index leading to the term loosely-consistent DHT. If the rendezvous churn rate happens to be very low, so the RPV remains stable, the loosely-consistent DHT will be synchronized and achieve optimum lookup performance.

3.2 Rendezvous Peer View (RPV)

Each rendezvous maintains its own rendezvous peer view *(*ordered list of other known rendezvous in the peergroup by their peer IDs). No strong consistency mechanism is used to enforce the consistency of the RPV across all rendezvous. Rendezvous may have temporarily or permanently inconsistent RPVs. A loosely-coupled algorithm is used for converging the RPV across all rendezvous. Rendezvous periodically select a given random number of rendezvous from their local RPV, and send them a random list of their known rendezvous. Rendezvous also send an heartbeat to their neighbor rendezvous (+1 and -1 RPV position). Rendezvous update and purge non-responding rendezvous from their RPV. In addition, rendezvous may retrieve rendezvous info from a predetermined set of bootstrapping *seeding* rendezvous. Each peergroup has the ability to define its own set of seeding rendezvous. Any rendezvous can act as a seeding rendezvous. Seeding rendezvous permit to accelerate the RPV convergence, as all rendezvous are pre-configured with the list of seeding rendezvous. Seeding rendezvous are used as the last resort when a rendezvous cannot find any other rendezvous, and to initially bootstrap rendezvous into the network. This is done to limit dependencies on seeding rendezvous and decentralize the RPV management. Beside the initial seeding, and after a rendezvous has learned about other rendezvous, seeding rendezvous are treated as any other rendezvous. If the rendezvous network is relatively stable, RPVs quickly converge across all rendezvous allowing a consistent index distribution.

3.3 Edge Peer Rendezvous Connection

The following section describes how an edge peer connects to one of the available rendezvous. Edge peers first check in their local cache for any rendezvous advertisements that point to a reachable rendezvous. Rendezvous advertisements persist across edge peers connections. The edge peer orders the candidate rendezvous in batch of five, and try them five at a time until a rendezvous connection is established. Edge peers only maintain one connection. If not rendezvous connection is established after a tunable period (30 sec.), the edge peer will attempt to search for rendezvous via a propagate request on its available transports (IP multicast), or on a peergroup specific propagate pipe if the edge peer has joined a peergroup. Existing rendezvous peers are listening, and will reply to that request. If after an extended period (30 sec.), no rendezvous has been found, the edge peer will query one of the seeding rendezvous. Finally, if none of the seeding rendezvous is reachable after a tunable period (5 minutes), the edge peer will try to become a rendezvous (if it has the right credential). The edge peer can return into an edge peer mode as soon as a rendezvous is found. The peer will continue to search for rendezvous in the background.

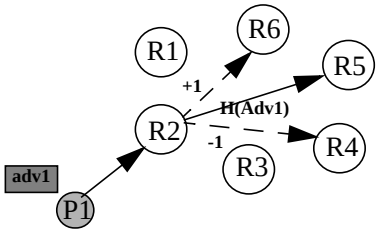
Partitioning of the RPV may occur as set of rendezvous may not be connected. However, RPV partitions will start to merge as soon as each partition reaches one of the seeding rendezvous, or a common rendezvous exists in both partitions. The loosely-consistent RPV mechanism is robust enough to enable partition merging even in the case of failures of seeding peers. Inconsistencies will be resolved overtime as rendezvous continue to randomly exchange information. The above rendezvous policy provides a balance between taking advantage of few predetermined seeding rendezvous while reducing dependencies on these rendezvous. It is expected after an edge peer initially bootstraps itself via the seeding rendezvous, it will learn about enough rendezvous so that it does not have to go back to the seeding rendezvous.

3.4 Rendezvous Propagation

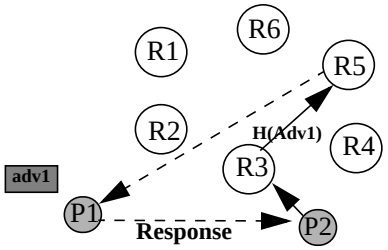
This section discusses how queries are propagated within the rendezvous RPV. Figure 4.1 shows how a newly published advertisement is indexed on the rendezvous RPV. Peer P1 publishes a new advertisement on its rendezvous R2 via the previously mentioned SDRI service. Each advertisement is indexed by SRDI using a pre-

defined number of keys such as the advertisement name, or ID. R2 uses the DHT function (*H(adv1)*) to map the index to a rendezvous in its local RPV. The RPV on R2 contains the R1 to R6 rendezvous. Let's suppose that the DHT function returns R5. R2 will push the index to R5. To increase the probability to retrieve the index in the vicinity of R5, and address the potential disappearance of R5, the index can also be replicated to the RPV neighbors of R5 (+1 and -1 in the RPV ordered list). In our example, the index is replicated on R4 and R6. It is important to note that index proximity in the RPV does not necessarily means physical network proximity. R5 and R4 may be on opposite sides of the hemisphere. Replicating the index around R5 means we are creating a logical region in the RPV where that index can be located. Let's assume that an edge peer P2 is looking for advertisement adv1 (see figure 4.2.a). P2 will issue a resolver query to its rendezvous R3. The SRDI service on R3 will compute the DHT function (H(adv1)) using R3 local RPV. If the RPV on R2 and R3 are the same, the DHT function will return the same rendezvous R5. R3 can forward the request to R5 that will forward it to P1 for responding to P2. This is working as long as the RPV is the same on R2 and R3.

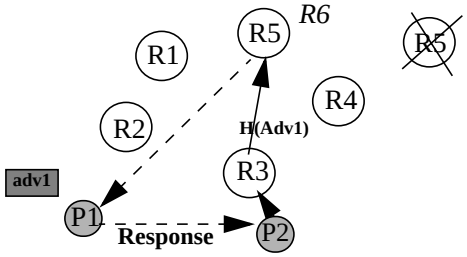
Suppose that R5 goes down, and R3 updated its RPV to reflect the fact that R5 disappeared (see figure 4.2.b) from its RPV. R3 will find out that R5 is down either through a RPV map update, or when it tries to send a message to R5. In this scenario, R3 has a new RPV that contains rendezvous R1 to R5. R5 now points to the rendezvous R6 of our previous RPV map. One missing rendezvous means that the RPV is shifted by one position. Upon receiving the resolver request from P2, R3 will compute the DHT function that will return the new R5 rendezvous. Since we also published the index on R6 as part of our replicated index strategy, we will find the index on the new R5. This example demonstrates, how even with an inconsistent RPV we able to successfully use the DHT. As long as the RPV shift is within the DHT replicated distance (+1,-1), we can guarantee the index will be found. Replicating the index in the vicinity of the initial rendezvous target increased our ability to retrieve the index even with a shifted RPV. It is important to point that no distributed maintenance of the RPV is required here. The replication distance can be increased (+2 or +3) for larger RPV maps.



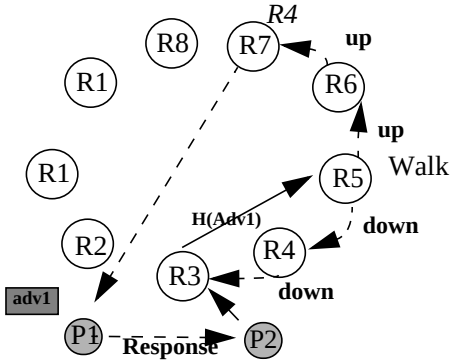
1) Publish Adv1



2.a) Search Adv1 (consistent RPV)



2.b) Search Adv1 (inconsistent RPV)



2.c) Search Adv1 (Limited-Range Walker)

Figure 4: Loosely-Consistent Rendezvous DHT

Let's look at a more chaotic scenario where the RPV is going through massive changes (see figure 4.2.c). The RPV on R3 is now composed of 8 rendezvous (R1-R8). R7 corresponds to the original R4. When R3 receives the query request from P2 it will compute the DHT function that maps the index to R5. Since the RPV has drastically changed, the index will not be found on R5. In this case, an alternative mechanism is used to *walk* the rendezvous peer view to continue searching. A *limited-range walker* is used to walk the rendezvous from the initial DHT rendezvous target. The walker will proceed in both an *up* and *down* direction (see figure 4.2.c). The walk will proceed in the up direction towards R6, and the down direction towards R4. The intent of the limited range walker is to look in the vicinity of the initial target rendezvous for a rendezvous that may have the index. The limited-range walker takes advantage of the increase probability to find the index in that region due to the DHT neighbor replication scheme. In our example, R5 forwards the request to both R4 and R6. A hop count is used to specify the maximum number of times the request can be forwarded. If R4 does not have the index, it will forward the request to the next peer in the down direction R3. Similarly, R6 will forward the request to the next up direction R7. When the index is found on R7, the query is forwarded to P1 and the walk stops in the up direction. The walk on the down direction will continue until the query hop count is reached, or there are no more rendezvous in that direction. Rather than continuing the walk up to the final hop count, a continue walk request may be sent to P2 after a portion of the walk has completed to ask P2 if the walk should continue. P2 may have already get a response from the other side walk. P2 can then decide to either continue or halt the walk.

In order to minimize index skewing while the RPV grows or shrinks, or is temporarily inconsistent, we are using a hash function that attempts to compensate for RPV changes. The hash function we used was originally suggested by Shinya Ishihara, and is dividing the hash space equally by the number of rendezvous in the RPV. Each rendezvous is assigned one segment of the hash space function of its rank in the RPV. If the hash number falls in the middle of the hash space, then the selected rendezvous will be in the middle of the RPV. The hash function allows to scale the hash space down to the RPV size. Even if the RPV changes drastically between the time the index was inserted, and the time it is looked up, chances are the RPV changed as much to the left and to the right of any given rendezvous. The relative position of a rendezvous tends to stay roughly the same as the RPV evolves. Using peer ID to order the RPV helps ensure an uniform change distribution across the RPV. If the RPV shrinks or grows, a rendezvous that was in the middle of the RPV will tend to still be around the middle. Reducing index skewing helps reduce the number of steps required during a walk to find an advertisement.

Walking the RPV in both directions reduces query latency while increasing resiliency. Since the RPV is ordered by peer IDs, each rendezvous is only visited once even if RPVs are inconsistent. The limited-range walker provides a garbage collection mechanism to the DHT initial lookup. If the rendezvous infrastructure is stable we take full advantage of the DHT, and rarely use the more expensive limited-range walker.

4. Relay Super-Peers

The Project JXTA network is an ad-hoc, multi-hop adaptive network. Connections may be transient. Message routing is non-deterministic. Routes may be unidirectional, and may change rapidly. The Project JXTA network introduces the notion of super-peers called *relay peers* for bridging peers that do not have direct physical connectivity (NAT, firewalls). Any peer may become a relay peer assuming it has the right level of credential, and capability (bandwidth, low churn rate and direct connectivity). Relay peers provide the ability to spool messages for unreachable or temporarily unavailable edge peers. For example in figure 5., Peer A wants to send a message to Peer B. Since Peer B is behind a NAT (Peer B address is not reachable from Peer A), Peer A cannot send a message directly to Peer B. Peer B uses the relay Peer D to make itself reachable. Relay peers play the role of *landmarks* facilitating the routing of messages between far away, and non-reachable peers. As part of peer advertisements, peers advertise a set of preferred relays to help route resolution. Peersl always attempt direct connections with other peers before using a relay. Messages go through one or more relays because of firewalls or NAT considerations.

The resolution of routes via relay peers is done transparently and dynamically by the JXTA virtual network. Applications address peers via their Peer IDs. They don't need to be aware of the JXTA network relay super-peer infrastructure. Edge peers maintain a leased-connection to one preferred relay, and retrieve messages from their allocated message queues. Peers can send messages through any available relays, not just its preferred one.

Overtime edge peers roam from relays to relays for optimizing their visibility, or improving connectivity quality to the network. It is important to point out that the relay/edge peer association is transient in nature. Relays only maintain states for their edge peers for an agreed upon lease period. Edge peers may reconnect at anytime to a different relay. Like rendezvous, relay peers maintain a loosely-coupled view between themselves to keep a list of available relays. Seeding relays are used to bootstrap the discovery of available relays. Edge peers cache relay advertisements to remember the list of available relays across reboots. Seeding relays are only used when no other relays are available. In case of a relay failure, edge peers transparently reconnect to another known relay.

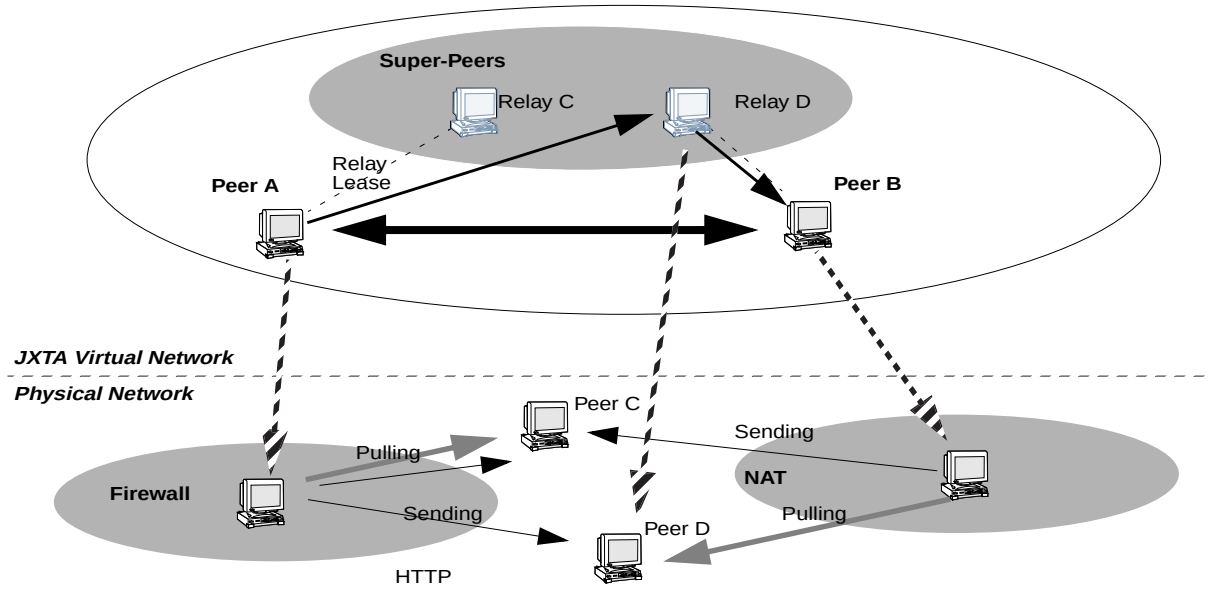


Figure 5: Project JXTA Relay Super-Peers

JXTA uses an *adaptive source-based routing*. Routes are initially constructed by the sender. This is done for decentralizing route management towards the edge peers, and reducing reliance on centralized routing tables. As any other resources, routes are represented by *route advertisement*s. Edge peers are caching route advertisements, and issuing discovery requests to discover a route advertisement for a given destination. A route advertisement describes the known way to reach a peer as an ordered sequence of hops. Each hop is defined by a peer ID with an optional set of endpoint addresses. Endpoint addresses are provided as hints, so the sender does not have to resolve peer IDs to endpoint addresses. Hops are typically relay peers. In an ad hoc mobile network edge peers may also act as hops to route messages due to lack of direct connectivity between peers. In common internet deployments, only one relay peer is needed for two edge peers to talk to each other, and route messages through a NAT or firewall domain. It is important to point out that route definition in a route advertisement is irrelevant of the sender location. The route description in a route advertisements can be used by many different senders. Senders will select the portion of the route that is relevant to them. For example, if the route advertisement for Peer A contains the following list of hops < Peer B, Peer C, Peer D>, then if Peer F knows how to talk to Peer C, then it only will use the <Peer B, Peer C> portion of the route. The intent is for route information to grow, and be coalesced over time as more alternative hops, or shortcuts are found. As any advertisements, route advertisement have a lifetime currently set to 15 minutes.

JXTA messages also contain routing information as part of their payloads. Every time a message goes through a hop, the routing information payload is updated with the current hop information. When a peer receives a message, it can use the message routing information as a hint for routing a reply to the sender. The reverse route

information takes into account uni-directional links, as well as potential shortcuts. For example, a peer behind a firewall can send a message directly to a peer outside a firewall, but the reverse may not be true due to NAT or Firewall considerations. Due to the unreliability of peers, Project JXTA made the decision not to fully rely on relay peers for routing. Allowing every message to carry its own routing information permits each message to be self-sufficient. When a message contains obsolete route information due to a relay failure, a new route is dynamically discovered using other known relay peers. The message routing information is also used for loop detection.

4.1 Project JXTA Messages and Credentials

Messages are the basic unit of data exchange between peers. Peers interact by sending and receiving messages. Project JXTA uses a binary wire format for representing messages and enabling efficient transport of virtually any kinds of payloads. Both XML and binary payloads can be sent. Each JXTA transport can use the most appropriate format for transferring data. A message is an ordered sequence of named and typed contents called *elements*, with the most recently added element appearing at the end of the message. As a message moves down the protocol stack (applications, services, and transports), each level can add one or more named elements to the message. As a message moves back up the stack, the protocol will extract those elements.

The need to support different levels of authentication and resource access in the *ad hoc* Project JXTA network leads to a *role-based* trust model in which a peer will typically act under the authority granted to it by another trusted peer. Peer relationships may change quickly and the policies governing access control need to be flexible in allowing or denying access. The Project JXTA message format allows the addition of a variety of metadata information to a message, such as credentials, digests, certificates, public keys, etc. Every message contains a credential. A *credential* is a token that, when presented in a message body, is used to identify a sender, and can be used to verify the sender's rights to send the message. The credential is an opaque token that is validated by the receiving end. The sending address placed in the message envelope is cross-checked with the sender's identity in the credential. Each credential's implementation is specified as a plug-in configuration, allowing multiple authentication configurations to co-exist on the same network. Message digests guarantee the data integrity of messages. Messages may also be encrypted and signed for confidentiality, integrity, and irrefutability. It is the intent of the JXTA network to be compatible with widely accepted transport-layer security mechanisms for message-based architectures such as Secure Sockets Layer (SSL), Transport Layer Security (TLS), and Internet Protocol Security (IPSec). The JXTA 2.0 implementation minimizes the number of extra message copying and buffering when sending and receiving messages.

5. PeerGroups

Peers in the Project JXTA network self-organize into *peergroups*. A peergroup represents a dynamic set of peers that have a common set of interests, and have agreed upon a common set of policies (membership, content exchange, etc.). Each peergroup is uniquely identified by a unique peergroup ID. Project JXTA does not dictate when, where, or why peergroups are created. Project JXTA only describes how a peergroup is created, published, and discovered. Users, service developers, and network administrators can dynamically create peergroups for scoping interaction between peers, and matching their applications demands. In figure 6, peergroup

A shows a peergroup that is a subset of a physical firewall domain.

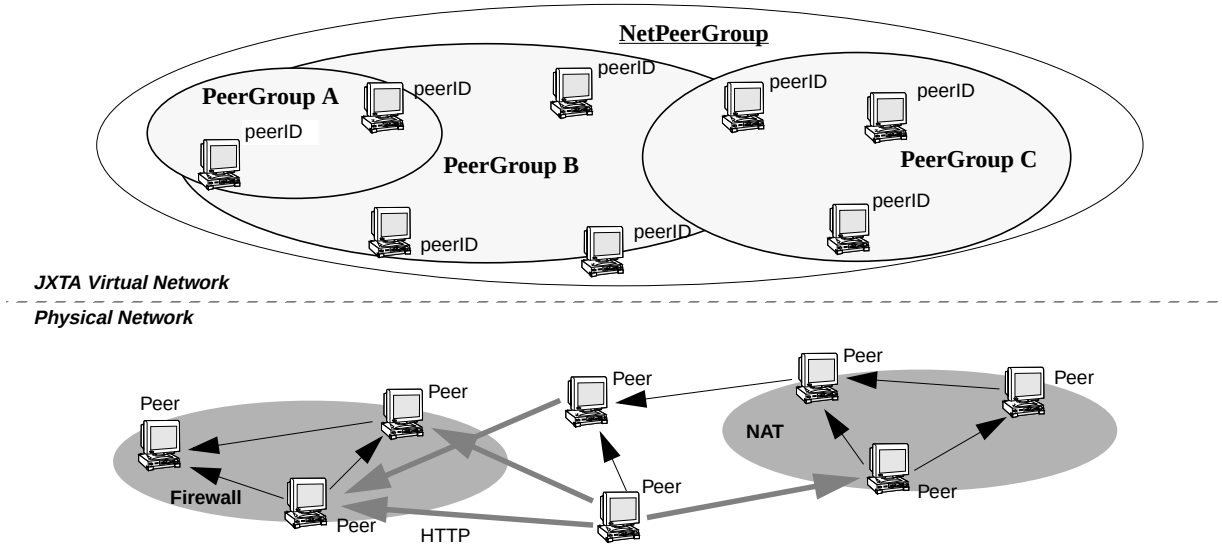


Figure 6: The Project JXTA Virtual Network

PeerGroup B shows a peergroup that is spanning multiple physical domains. Peergroup C shows a peergroup that is exactly mapping the boundary of a NAT domain. A peer can belong to multiple peergroups at the same time.

Project JXTA recognizes three main motivations for creating peergroups:

- To create secure domains for exchanging secure contents. Peergroups form logical regions whose boundaries limit access to non-members. A peergroup does not necessarily reflect the underlying physical network boundaries such as those imposed by routers and firewalls. Peergroups virtualize the notion of routers and firewalls, subdividing the network in secure regions without respect to actual physical network boundaries.

- To create a scoping environment. Peergroups are typically formed and self-organized based upon the mutual interest of peers. No particular rules are imposed on the way peergroups are formed, but peers with the same interests will tend to join the same peergroups. Peergroups serve to subdivide the network into abstract regions, providing an implicit scoping mechanism for restricting the propagation of discovery and search requests.

- To create a monitoring environment. Peergroups allow monitoring (traffic inspection, accounting, tracing) of peers for any purpose.

At boot time, every peer joins the *NetPeerGroup*. The NetPeerGroup acts as a root peergroup every peer initially belongs. Peergroups typically publish a set of services called *peergroup services*. Project JXTA specifies a standard set of core peergroup services with their associated protocols (discovery, resolver, pipe, peer info, and rendezvous). If a core protocol is not adequate for a more demanding peergroup, it can be replaced with a customized protocol. Peergroup advertisements contain the list of all peergroup services which implement the protocols used in a peergroup. Project JXTA core peergroup services provide the basic functionality to support a peergroup's existence (publishing and discovering resources and exchanging messages). Peergroup creators can customize their peergroup services to match their peergroup requirements (centralized versus decentralized, deterministic versus underterministic, etc.). Peergroup services are composed of a collection of instances of the service running on multiple members of the peergroup. Each instance can work autonomously as replicas, or by cooperating with each other. If any one peer fails, the collective peergroup service is not affected, because it is likely that the service is available from another peer member.

6. Pipes

Pipes are virtual communication channels used to send and receive messages between services and applications. Pipes provide a virtual abstraction over the peer endpoints to provide the illusion of virtual in and out mailboxes that are not physically bound to a specific peer location. Pipes can connect one or more peer endpoints. At each endpoint, software to send, receive, or manage message queues or streams is assumed. The pipe ends are referred as the *input pipe* (receiving end), and the *output pipe* (sending end) see figure 7.

Pipes are published and discovered using *pipe advertisements*, and are uniquely identified by a *Pipe ID.* Pipe ends are dynamically bound to a peer endpoint at runtime by the resolver. Using the pipe abstraction, applications and services can transparently failover from one physical peer endpoint to another in order to mask a service or peer failure, or to access a newly published instance of a service. The pipe binding process consists of searching and connecting the two or more ends of a pipe. When a message is sent into a pipe, the message is sent by the local output pipe to the destination pipe input currently listening to this pipe. Pipes offers two modes of communication:

- A *point-to-point pipe* connects exactly two pipe ends with a unidirectional and asynchronous channel, an input pipe end that receives messages sent from the output pipe end. No reply or acknowledgment operation is supported. Additional information in the message payload like a unique ID may be required to thread message sequences. The message payload may also contain a pipe advertisement that can be used to open a new pipe to reply to the sender (send/response).

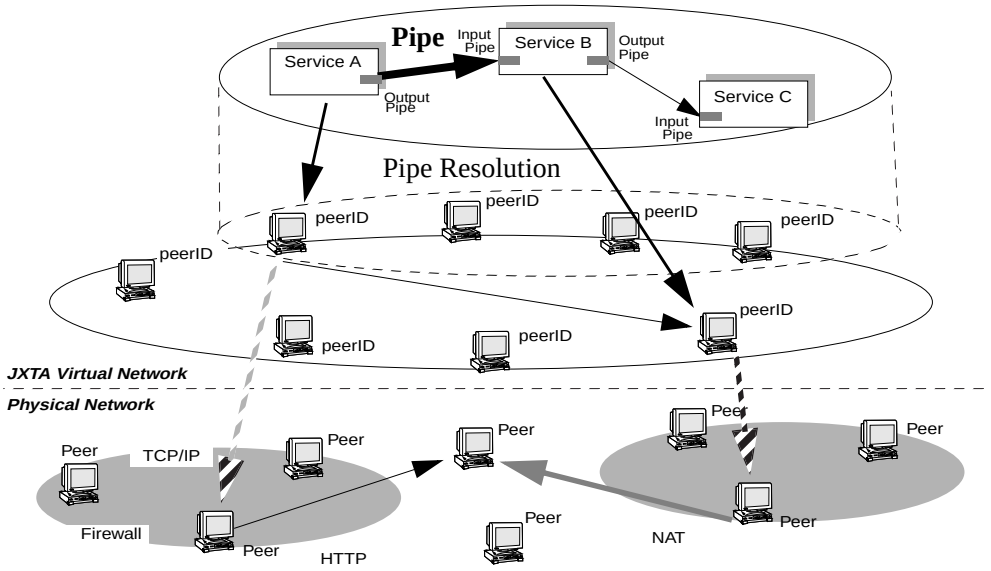


Figure 7: Project JXTA Pipes

- A *Propagate pipe* connects one output pipe to multiple input pipes. Messages flow into the input pipe ends from the output pipe end (propagation source). The propagate message is sent to all listening input pipe ends in the current peergroup context. This process may create multiple copies of the message. On TCP/IP, when the propagate scope maps an underlying physical subnet in a one-to-one fashion, IP multicast may be used as an implementation for propagate pipes. Propagate pipes can be implemented using point-to-point communication on transports that do not provide multicast (e.g. HTTP).

Bi-directional, reliable and secure pipe services have been implemented on top of the core pipe services. Asynchronous point-to-point pipes can have ends that are connected to different peers at different times, or not con-

nected at all. Pipe advertisements can contain unique data-typed XML schema to describe the valid set of messages to be sent or received through a pipe. The JXTA 2.0 implementation introduces a socket API to the pipe service for ease of programming.

7. Project JXTA Security

Certificate authority-based trust models that provide public key infrastructures to secure Web-based transactions are expensive to deploy and manage. Project JXTA provides an entry-level trust model [1] that costs nothing, and which can be easily generalized to support the existing Internet trust models. This trust model is appropriate for chat room's, content sharing, and secure financial transactions. The Project JXTA trust model, permits peers to be their own certificate authorities, or socially accumulated inter-peer interactions. Project JXTA provides strong cipher algorithms to protect principals such as local data (all local data is protected with a pass phrase), data in transit on the Project JXTA virtual network, and remotely stored data. The Internet Engineering Task Force's (IETF) Transport Layer Security (TLS) [11] is the IETF continuation of SSL.V3, and is used for securing communication between computers. Project JXTA has implemented a virtual transport based on TLS to provide secure communication between peers. The default cipher suite is RSA1024 with 3DES and SHA-1.

When a JXTA secure pipe (figure 8) is created, and the associated peer endpoints are resolved, a virtual TLS transport is instantiated. All data moved through secure pipes is then multiplexed over this single instance of a virtual TLS transport. The transport is bi-directionally secured end-to-end with TLS, independently of JXTA relays and the underlying physical transports. Peers can create pipes which will behave like multiple secure channels over a single TLS transport. Project JXTA's TLS implementation minimizes the needed network resources by amortizing one TLS handshake over multiple data pipes and making conservative use of the bandwidth on the physical layer.

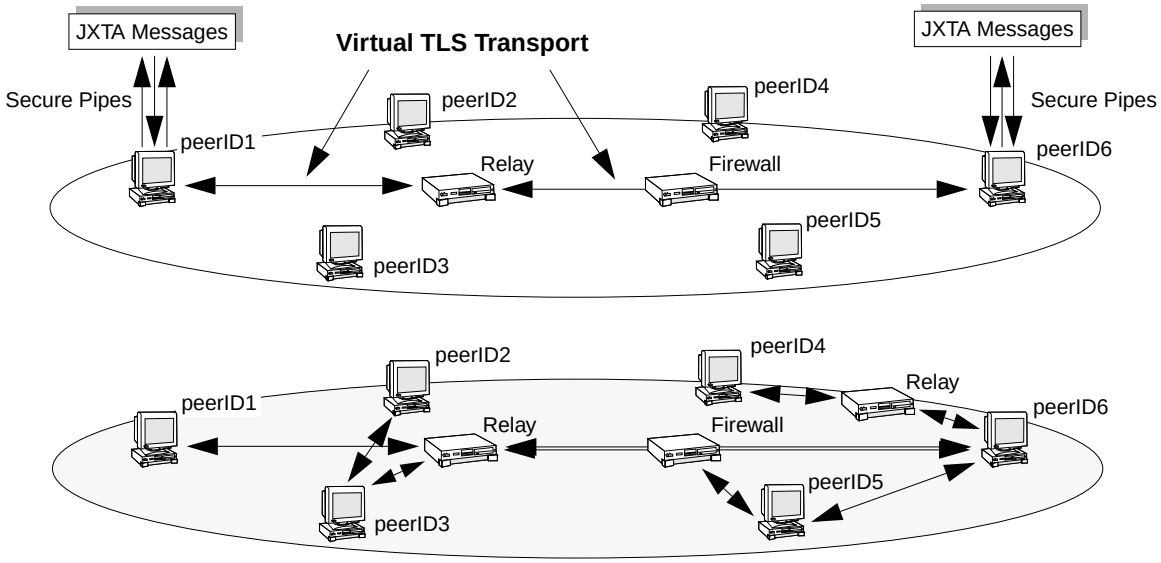


Figure 8: Project JXTA Virtual TLS transport.

Because the TLS virtual transport is bi-directional, both client and server authentication is required because peers may be both clients (sending data) and servers (receiving data). Peers must possess the X509.V3 root certificates of any peer with whom they wish to communicate securely. The root certificates contain the RSA 1024-bit public keys used to verify the RSA private key signatures of peer X509.V3 service certificates. The service certificates are used to authenticate the communicating peer endpoints by TLS after the signatures of the root certificates of their issuers are locally verified.

Finally, it is possible to implement peergroup authentication based on X509.V3 certificates. When one joins a peergroup, they will receive the peergroup creator's root certificate under the protection of a TLS connection, and will use a Certificate Service Request (CSR) to acquire a group membership X509.V3 certificate signed

with the private key of the group creator's root certificate. All of this information is then stored locally on the peer under the protection of its password phrase. When a peer contacts another peergroup member, the former peer can be authenticated by the latter using the TLS handshake's certificate request/response and certificate verification.

8. The Project JXTA Protocols

The Project JXTA protocols are composed of six protocols (figure 9) divided into two categories:

- *Core Specification* Protocols
- *Standard Service* Protocols

8.1 Core Specification protocols

The JXTA protocols have been designed to be implementable on very small systems with only a few required components and behaviors. The functionality required of all implementations is defined by the JXTA Core Specification protocols. Implementations that wish to be JXTA compliant must implement all of the JXTA Core Specification protocols. Implementation of the JXTA Core Specification does not guarantee or even necessarily provide interoperability with other JXTA implementations. There are a number of behaviors which may need to be provided by JXTA implementation which are not part of the JXTA Core Specification. Existing implementations of these components are described separately in the JXTA Standard Services specification (see next section).

The Core Specification defines two protocols:

- The *Endpoint Routing Protocol* (ERP) is the protocol by which a peer can discover a route (sequence of hops) used to send a message to another peer. If a peer A wants to send a message to peer C, and there is no direct route between A and C, then peer A needs to find the intermediary peer(s) to route the message to C. ERP is used to manage and determine the routing information. If the network topology has changed such that the route to C can no longer be used, the peer can use ERP to find routes known by other peers to construct a new route to C.

- The *Peer Resolver Protocol* (PRP) is the protocol by which a peer can send a generic resolver query to one or more peers, and receive a response (or multiple responses) to the query. The PRP protocol permits the dissemination of generic queries to one or more handlers within the group and to match them with responses. Each query is addressed to a specific handler name. This handler name defines the particular semantics of the query and its responses, but is not associated with any specific peer. A given query may be received by any number of peers in the group, possibly all, and processed according to the handler name if such a handler name is defined on that peer.

8.2 Standard Service Protocols

The JXTA Core Specification defines the required components and behaviors for all JXTA implementations. In order to create a complete JXTA implementation there are some additional components which all implementation should provide. The JXTA Standard Services protocols are optional JXTA protocols and behaviors. Implementations are not required to implement these services, but are strongly recommended to do so. Implementing these services will provide greater interoperability with other implementations and broader functionality.

The Standard Services protocols specification defines four protocols:

- The *Rendezvous Protocol* (RVP) is the protocol by which peers can subscribe or be a subscriber to a propagation service. Within a peergroup, peers can be rendezvous peers, or peers that are listening to rendezvous peers. RVP allows messages to be sent to all of the listeners of the service. RVP is used by the Peer Resolver Protocol in order to propagate messages.

- The *Peer Discovery Protocol* (PDP) is the protocol by which a peer publishes its own advertisements, and discovers advertisements from other peers (peer, peergroup, module, pipe and content). PDP uses the Peer Resolver Protocol for sending and propagating discovery advertisement requests.

- The *Peer Information Protocol* (PIP) is the protocol by a which a peer may obtain status information about other peers, such as state, uptime, traffic load, capabilities, etc. PIP uses the PRP for sending and propagating peer information requests.
- The *Pipe Binding Protocol* (PBP) is the protocol by which a peer can establish a virtual communication channel or pipe between one or more peers. The PBP is used by a peer to bind the two or more pipe ends of the connection (input and output pipe) to a physical peer endpoint. PBP uses the PRP for sending and propagating pipe binding requests.

.

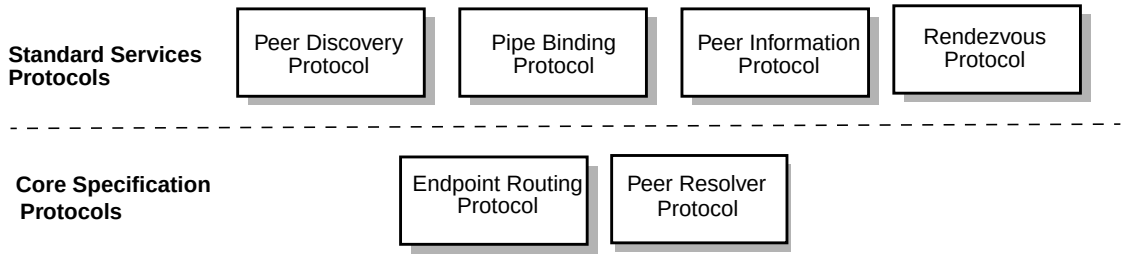


Figure 9: The Project JXTA protocols.

The Project JXTA protocols have been designed to be easily implemented on unidirectional links and asymmetric transports. In particular, many forms of wireless and mobile networking do not provide equal capability for devices to send and receive. Project JXTA permits any unidirectional link to be used when necessary, improving overall network connectivity in the system. The intent is for the Project JXTA protocols to be as pervasive as possible, and easy to implement on any transport. Implementations on reliable and bi-directional transports such as TCP/IP or HTTP should lead to efficient bi-directional communications.

Furthermore, a peer only needs to implement the protocols that it requires. For example, a device may have all its advertisements pre-stored in memory, not requiring to implement the Peer Discovery Protocol. A peer may use a pre-configured set of relays to route all of its messages, hence not requiring to implement the Peer Endpoint protocol, but just sends messages to relays for forwarding. A peer may not need to obtain, or wish to provide, status information to other peers, and therefore does not need to implement the Peer Information Protocol.

The design of the JXTA protocols seeks to create a set of protocols that have very low overhead, make few assumptions about the underlying network transport and impose few requirements on the peer environment, and yet are able to be used to deploy a wide variety of P2P applications and services in a highly unreliable and changing network environment.

9. Project JXTA 2.0 Reference Implementation Architecture

The following section presents an overview of the JXTA 2.0 Java J2SE reference implementation. The reference implementation provides a blueprint for developers to implement the JXTA protocols. The reference implementation implements all six of the JXTA protocols, and a few other standards services. The implementation is decomposed in a number of inter-dependent components see figure 10. Each component defines its own APIs to interact with other components. We attempted to create a modular implementation that enables easy replacement or extension of each component.

The following components provide the necessary infrastructure to manage advertisements:

9.1 ID

Within the JXTA protocols there are a number of entities that need to be uniquely identifiable. These are peers, peergroups, pipes, etc. A JXTA ID uniquely identifies each of these entities and serves as a canonical means of referring to that entity. This component implements JXTA IDs as 128 bits UUIDs. UUIDs are self-generated

locally using a random generator seed. A JXTA ID is a standard URN in the JXTA ID namespace. JXTA ID URNs are identified by the URN namespace identifier *jxta* (ex. urn:jxta:id12345).

9.2 Cache Manager (CM)

The cache manager is used for caching, indexing and storing persistently advertisements. This component enables the efficient storage, indexing and retrieval of advertisements. The JXTA 2.0 implementation uses a scale down version of the Apache open-source Xindice[10] XML database for storing and retrieving JXTA advertisements. The Xpath indexing part of Xindice is not used. The cache manager allows to specify on which fields an advertisement should be indexed. Advertisement Java objects are serialized and deserialized as XML documents. The cache manager provides persistent storage of advertisements across restarts. The cache manager implements an efficient indexing mechanism for retrieving advertisements optimizing retrieval time for the most common core advertisements (peer, peergroup, module, pipe, route, etc.). The cache manager controls the decay of advertisements in the cache. Each advertisement is published in the cache with an associated time-to-live expiration date. Advertisements are deleted from the cache when they have expired.

9.3 XML Parser

The XML parser enables to parse XML documents. The parser enables the serialization and deserialization of Java objects into output and input XML character streams. A lightweight XML parser was implemented that implements basic XML DOM functionality minimizing footprints. The JXTA protocols use a minimal subset of XML (limited namespace and no validation). Small devices do not require a full parser as protocol messages may be pre-generated.

9.4 Advertisements

The advertisement module implements the core advertisements used in the JXTA protocols: peer, peergroup, endpoint, rendezvous, transport, pipe, module and route advertisements. The advertisement module allows a Java object to be serialized and deserialized into an XML document representing an advertisement.

The following components manage the physical transports, and maintain the mapping between virtual messengers and physical connections:

9.5 HTTP Transport

The HTTP transport implements the HTTP transport binding as specified in the JXTA protocols binding specification. The HTTP transport is implemented as servlets using the Jetty embedded server. Jetty is a Java HTTP server and servlet container, so it is not necessary to run a separate web server (like Apache) in order to use servlets. The HTTP transport runs in the same process than the JXTA platform. The HTTP transport provides the ability to initiate a connection between two peers. The connection is first used to determine the logical identities of participating peers. HTTP GET requests are used for determining the logical identity of the HTTP server side, and for polling messages. HTTP POST requests are used for sending and receiving JXTA messages. The HTTP transport supports firewall traversal through proxy.

9.6 TCP/IP Transport

The TCP/IP transport implements a TCP/IP transport binding as specified in the JXTA protocols binding specification. The JXTA 2.0 implementation takes advantage of bi-directional connections, so a single socket connection can be used to send and receive data. Idle connections are recycled to enable handling large number of connections on relay and rendezvous. A limited number (currently 3) of physical connections can be established to a same destination for simultaneous data transfer. The TCP/IP transport may be configured to accept broadcast messages through IP Multicast.

9.7 TLS Virtual Transport

The TLS virtual transport implements a reliable, secure, end-to-end transport over the HTTP and TCP/IP transport. The virtual TLS transport implementation fragments messages into TLS records. A reliable message protocol is used to guarantee that messages containing these records arrive at the destination peer in the same order

in which they were transmitted. The TLS transport performs the exchange of keys, and negociation of a session key to encrypt messages.

9.8 Message

The message service implements the JXTA wire binary message format used when messages are sent in the JXTA virtual network. JXTA uses a binary format for allowing efficient transfer of both binary and XML payloads. All JXTA protocol messages are represented as XML payloads. Messages are formed as an ordered sequence of elements. Each element has a unique name, length and MIME-type. It is important to point that the JXTA wire format representation is agnostic of data representation. Any kinds of data can be sent. Each service when processing a message can add or remove its own message elements. Message elements permit to isolate the different parts of a message allowing greater protection. For instance, the Router service is only allowed to manipulate the Router element message, but does not touch any other elements in the message.

9.9 Virtual Messenger

The virtual messenger service abstracts all JXTA transports thought a common interface for the endpoint service. The virtual messenger normalizes the behavior of synchronous versus asynchronous messenger transports, with well defined behaviors with regard to synchronicity for sending and receiving messages.

9.10 Endpoint Service

The endpoint service implements the JXTA endpoint abstraction, which is used for encapsulating multiple messenger transports into a single virtual endpoint. Endpoints provide the virtual network abstraction used by peers to communicate independently of the underlying network topology (firewalls or NAT), and physical transports. The endpoint service provides uniform de-multiplexing of incoming messages and associated resource management. The endpoint service delegates network propagation, and connectivity establishment to the appropriate messenger transports. The endpoint service also provides buffering of outgoing messages, caching of outgoing messengers, and a unified messenger behavior.

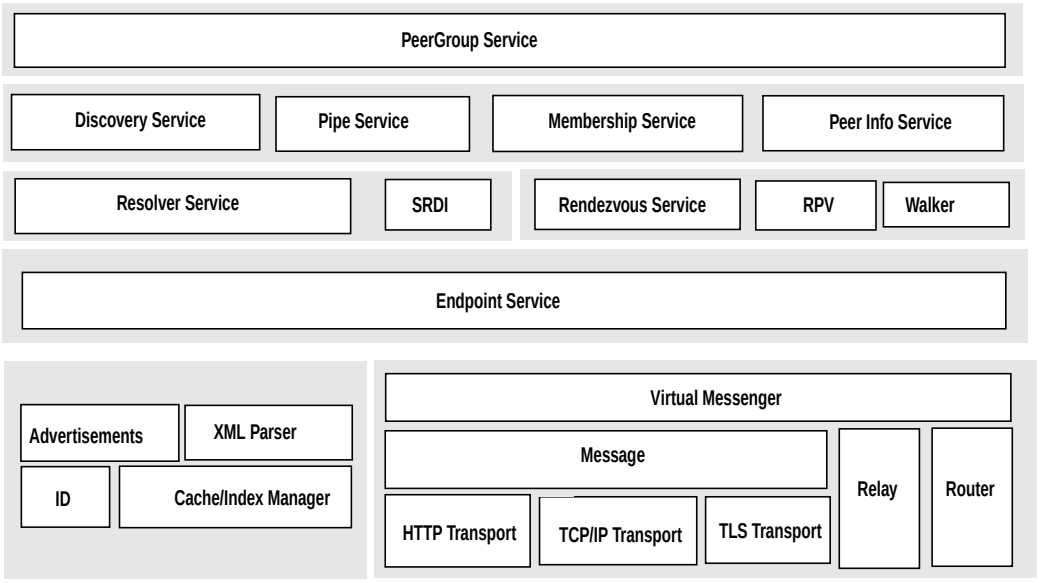


Figure 10: Project JXTA J2SE Reference Implementation Overview

9.11 Router

The router implements the routing service used by a peer endpoint for discovering and maintaining route information to other peers. Some peers may not have a direct connection, so messages need to be routed through one

or more peers to their final destination. The router implements the Endpoint Routing Protocol (ERP), allowing a peer to query and discover route information. The router maintains in-memory routing table of discovered routes. Each message sent and received contains a routing element used for updating route information. When receiving a message, the router payload is examined by the router for optimizing its route information. The router provides route information to the Endpoint service for delivering messages.

9.12 Relay

The relay service provides a mechanism for a peer that is not directly reachable to use a relay peer to hold messages until there can be delivered. The relay service is used for reverse-proxy firewall and NAT traversals. The relay service uses a quota and lease mechanisms to manage message queues for edge peers.

The following components define optional services for supporting JXTA peergroups:

9.13 Rendezvous Service

The rendezvous service is used for propagating messages within the scope of a peergroup. This service implements the RendezVous Protocol (RVP). The rendezvous service allows edge peers to obtain a lease for sending and receiving propagated messages. The rendezvous service uses the endpoint service for propagating messages.

9.14 Rendezvous Peer View (RPV)

The RPV service manages a list of available rendezvous in a peergroup. The RPV maintains a loosely-coupled decentralized view of all rendezvous that converges overtime. Each rendezvous has its own RPV list.

9.15 Walker

The walker service provides a pluggable mechanism to traverse or walk the rendezvous RPV to propagate queries. The walker implements a default policy to walk the rendezvous from the initial SRDI target rendezvous in an up and down direction.

9.16 Resolver Service

The resolver service is used for sending generic query/response (asynchronous RPC) requests within the scope of a peergroup. The resolver service implements the Peer Resolver Protocol (PRP). The resolver uses the endpoint and rendezvous service for unicasting and propagating requests within the scope of a peergroup.

9.17 SRDI

The SRDI service distributes indices of shared resources within the rendezvous network. These indices can be used to forward queries in the direction where the query is most likely to be resolved, or repropagate messages to peers interested in these propagated messages.

9.18 Discovery Service

The discovery service is used for discovering and publishing any type of advertisements (peer, peergroup, pipe, module, etc.) in a peergroup. The discovery service implements the Peer Discovery Protocol (PDP). The peer discovery service uses the resolver service for sending and receiving discovery requests. The discovery service uses the cache manager to cache and store advertisements.

9.19 Pipe Service

The Pipe service is used for creating and binding pipe ends (input and output pipes) to peer endpoints within the scope of a peergroup. Three types of pipes are implemented, unicast (one-to-one), secure, and propagate (one-to-N). The pipe service uses a pipe resolver service for dynamically binding a pipe end to a peer endpoint. The pipe resolver service implements the Pipe Binding Protocol (PBP).

9.20 Membership service

The membership service is used to manage peergroup membership, and issue membership credentials. New

peers need to be authenticated before they can join a peergroup. The membership service provides a pluggable authentication framework to support different authentication mechanisms (JAAS, LDAP).

9.21 PeerInfo service

The PeerInfo service provides a pluggable framework for metering and monitoring peers. Metering monitors can be associated with any peergroup services to collect information about that service. The PeerInfo service provides a remote monitoring capability to collect metering data about a remote peer. The PeerInfo service implements the Peer Information Protocol.

9.22 PeerGroup Service

The peergroup service implements the booting of a peer in the NetPeerGroup and the peergroup navigation. The service exposes all the core NetPeerGroup services (discovery, resolver, pipe, rendezvous, etc.) to applications that have joined the NetPeerGroup. The peergroup service enables a peer to create, advertise, and join new peergroups. Peergroups export a set of peergroup services that are represented as modules. The peergroup service manages the underlying infrastructure for advertising and loading modules. The peergroup service also manages the PlatformConfig configuration file.

10. Status and Future Directions

The Project JXTA protocols establish a virtual network overlay on top of existing networks, hiding their underlying physical topology.Project JXTA enables application developers, not just network administrators to design network topology that best match their application requirements. Multiple ad hoc virtual networks can be created and dynamically mapped into one physical network unleashing a richer multi-dimensional virtual network world. Project JXTA is looking ahead where billion of network services, all addressable on the network will be able to discover and interact with each other in an ad doc and decentralized manner through the formation of a multitude of virtual networks. The JXTA virtual network standardizes the manner in which peers discover each other, self-organize into peergroups, discover network resources, communicate, and monitor one another. The Project JXTA network is built out of five key abstractions — uniform peer ID addressing, peergroups, advertisements, resolver, and pipes — that provide a generic infrastructure to deploy P2P services and applications. By providing the base mechanisms and not dictating policies, Project JXTA enables a diversity of P2P applications to be developed. This paper describes the Project JXTA 2.0 implementation that was recently released by the JXTA community. The JXTA 2.0 implementation introduces a number of new features for improving overall functionality, performance, and scalability of the JXTA network. The JXTA 2.0 release is making a stronger differentiation in the way JXTA super-peers (relay and rendezvous) behave, and interact with edge peers. The JXTA 2.0 implementation introduces the concept of a rendezvous peer view to connect rendezvous within a peergroup. Resolver queries are no more propagated to edge peers as in JXTA 1.0 reducing overall network traffic. Edge peers use a loosely-coupled distributed hash index to index advertisements on the rendezvous peer view for efficient query lookups. The JXTA 2.0 implementation provides better resource management (threads and queues), and implements resource usage limits to fairly allocate resource between platform services. The full reference implementation of the JXTA 2.0 protocols specification has been completed, and is available at *platform.jxta.org*. We are continuing to improve the scalability of the implementation, adding more security infrastructure, and better monitoring tools. We are also looking at providing better integration and support for Web services (SOAP, UDDI, WSDL).

11. References

1. B. Traversat and al, *The Project JXTA Virtual Network*, May 2001, www.jxta.org/docs/JXTAprotocols.pdf,
2. B. Traversat and al., *Project JXTA-C: Enabling a Web of Things*, in proceedings of the HICSS-36 Conference, Jan. 2003.
3. S. Oaks, B.Traversat, L.Gong, *JXTA in a Nutshell*, O'Reilly Press, 0-596-00236-X, Sept. 2002.
4. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, *A Scalable Content Addressable Network*, ACM SIGCOM, 2001
5. F. Dabek, E. Brunskill, M.F. Kaashoek, D. Karger, R. Morris, I. Stoica, and H. Balakrishnan, *Building Peer-to-Peer Systems with Chord, a Distributed Lookup Service*, 2001
6. LongWork Network, *http://www.echelon.com/products/Core/protocol/Default.html*
7. T. Berners-Lee, J. Hendler, and O. Lassila, *The Semantic Web*, *http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html*
8. B. Zhao and al., *Brocade: Landmark Routing on Overlay Networks*, in Proceeding of the First International Workshop on Peer-to-Peer Systems, Mar. 2002.
9. Q. Lv and al., *Search and Replication in Unstructured Peer-to-Peer Network*, www.cs.princeton.edu/~qlv/download searchp2p_full.pdf.
10. Xindice, *xml.apache.org/xindice.*
11. T. Dierks and C. Allen, *The TLS Protocol*, IETF RFC2246. January, 1999.

12. Acknowledgments

We would like to thank the entire JXTA community for their contributions and helping us refine the JXTA protocols.