P2PS (Peer-to-Peer Simplified)

Dr. Ian Wang
Schools of Physics and Computer Science
Cardiff University

Abstract

P2PS (Peer-to-Peer Simplified) is a lightweight infrastructure for peer-to-peer service discovery and pipe-based communication. As its names suggests, P2PS aims to provide a simple platform on which to develop peer-to-peer style applications, hiding the complexity of other similar architectures such as JXTA [6].

As the P2PS infrastructure is based on XML discovery and communication, it is independent of any implementation language or computing hardware. Assuming that suitable P2PS implementations exist, it should be possible to form a peer network that includes everything from super computing peers to PDAs. Furthermore, communication within P2PS is not tied to a single protocol, such as TCP/IP, and messages can be relayed across multiple protocols.

In this paper we present an outline of the P2PS protocols and the current Java implementation. We aim to illustrate both the simplicity of developing peer-to-peer applications using P2PS, and the flexibility of P2PS to incorporate different discovery and communication protocols transparently to the application.

1 Introduction

The common perception of peer-to-peer (P2P) technology concerns illegal sharing of copyrighted material, in particular music and video. The phenomenal but controversial success of Napster [8], Gnutella [3] and other P2P file sharing systems has demonstrated the power and scalability of P2P architectures, but also highlighted the problems in managing and controlling such decentralized systems.

Despite the publicity it has received, file sharing (legal and illegal) is only one of many potential applications for P2P technology. The term peer-to-peer (P2P) generally refers to distributed network architectures in which nodes can share resources (e.g. processing time, storage) and act both as client and server [7][9]. Such architectures have already been applied to areas including parallel computation (e.g. SETI@Home [10]), instant messaging (e.g. AIM [1]) and online games (e.g. Buzzpad [2]), however the majority of these systems are specific a single domain.

P2PS is a lightweight infrastructure for developing peer-to-peer style applications. Its architecture is inspired by that of JXTA [6][11] and its functionality is in many ways a subset of that provided by JXTA. Unlike the previously mentioned systems, P2PS and JXTA are generic P2P infrastructures designed to be applicable to differing applications. Both P2PS and JXTA provide the basic constructs required by P2P applications, namely service discovery and pipe based communication.

The differences between JXTA and P2PS are more in how the P2P network is realized as opposed to the underlying concepts, and also in what P2PS does not concern itself with. For example:

- In JXTA there are separate protocols for publishing advertisements and for sending queries. In P2PS queries are themselves advertisements and therefore published as normal advertisements.
- P2PS provides a more expressive and extendible discovery language than JXTA, allowing peers to query for arbitrary advertisement types rather than restricting them to predefined types.
- Both P2PS and JXTA allow messages to be relayed across multiple protocols. In JXTA this routing is expressed explicitly in an XML message envelope, while in P2PS it is an endpoint resolver implementation issue.
- In P2PS, all services are remote instances contacted using P2PS pipes. JXTA allows remote services but also enables service code to be imported by a peer via module implementations.

Rather than being a problem, the focused functionality of P2PS is advantage in many situations. For

```
public class PeerServer implements MessageListener {

  public PeerServer() throws IOException {
    Peer peer = new PeerImp();
    peer.init();

    adverts = peer.getAdvertisementFactory();
    discovery = peer.getDiscoveryService();
    pipes = peer.getPipeService();

    // initialise server pipe advertisement
    pipead = (PipeAdvertisement) adverts.newAdvertisement
      (PipeAdvertisement.PIPE_ADVERTISEMENT_TYPE);
    pipead.setPipeName("serverPipe");

    // create server pipe and attach listener
    InputPipe inpipe = pipes.createInputPipe(pipead);
    inpipe.addPipeListener(this);

    // publish server pipe advertisement
    discovery.publish(pipead);
  }

  public void messageReceived(MessageReceivedEvent event) {
    // display received messages
  }
}
```

Table 1: Simple Server Peer Example.

example, module specifications and implementations within JXTA cloud the process of creating and discovering simple remote services, while wrapping messages in an XML envelope complicates creating and sending simple messages and introduces performance issues [4]. For most P2P applications, the subset of functionality provided by P2PS is sufficient, with P2PS providing a lighter and cleaner implementation than JXTA. Where additional capabilities are required, the architecture is designed to be easily extended, as we shall discuss in this paper.

In Section 2 we show the application view of P2PS using a simple client/server application example. Section 3 then outlines the key elements of the P2PS architecture, including advertisement and queries (Sections 3.1 and 3.2), discovery and rendezvous (Sections 3.3 and 3.4), endpoint resolvers (Section 3.5), and peer groups (Section 3.6). An overview of the paper and future directions is given in Section 4.

2 Application Example

In this Section we outline a simple client/server example application written using the current Java P2PS implementation. Table 1 lists the server code, in which an input pipe is created using the P2PS pipe service and advertised using the discovery service. As can be seen from the code, the pipe is created from a pipe advertisement. The fields in

```
public class PeerClient
  implements DiscoveryListener, PipeConnectionListener {

  public PeerClient() throws IOException {
    Peer peer = new PeerImp();
    peer.init();

    adverts = peer.getAdvertisementFactory();
    discovery = peer.getDiscoveryService();
    pipes = peer.getPipeService();

    // listen for discovered advertisements
    discovery.addDiscoveryListener(this);
    // listen for pipe connection
    pipes.addPipeConnectionListener(this);

    // create pipe query for server pipe
    pipequery = (PipeQuery)
      adverts.newAdvertisement(PipeQuery.PIPE_QUERY_TYPE);
    pipequery.setQueryPipeName("serverPipe");

    // publish pipe query
    discovery.publish(pipequery);
  }

  public void advertDiscovered(DiscoveryEvent event) {
    // connect output pipe to discovered pipe
    Advertisement advert = event.getAdvertisement();
    pipes.connectOutputPipe((PipeAdvertisement) advert);
  }

  public void outpipeConnected(PipeConnectedEvent event) {
    // send test message when pipe connected
    event.getOutputPipe().send("HELLO".getBytes());
    }
}
```

Table 2: Simple Client Peer Example.

this pipe advertisement determine the attributes of the pipe created by the pipe service. In this example only the name of the pipe is set, but if for example a bidirectional pipe was required then `setPipeType(PipeTypes.BIDIRECTIONAL)` could be used. The `addPipeListener` command adds the server as a listener to the input pipe. This means that when a message is received on the pipe a `messageReceived` event is raised, allowing the server to handle the message.

Table 2 lists the client code. In the constructor of the client a pipe query is created and published in an attempt to discover available server pipes. The `setQueryPipeName` method indicates the client is interested in pipes named `serverPipe`. If the client was interested in bidirectional pipes then `setQueryPipeType` could be used to narrow the search further.

By adding itself as discovery listener, the client receives `advertDiscovered` events when the discovery service receives an advertisement. In the example, when a pipe advertisement is discovered the client attempts to connect an output pipe. As output pipe connection is asynchronous, the client must wait until

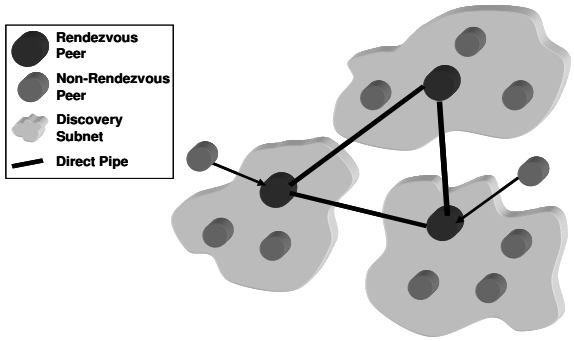

Figure 1: P2PS network containing multiple discovery subnets connected through rendezvous peers.

an `outpipeConnected` event is received before the test message can be sent.

3 P2PS Infrastructure

In order to form a coherent peer-to-peer network it is essential that individual peers be able to discover and communicate with the services (pipes, endpoint resolvers etc.) offered by other peers in the network. For service discovery to take place there first must be some form of language for specifying and querying for available services, which in P2PS takes the form of extendible XML based advertisements and queries, as described further in Sections 3.1 and 3.2.

There also needs to be some form of infrastructure in place to facilitate the discovery process. Some systems use a form of centralized lookup (e.g. Napster [8], Jini [5]), but this introduces scalability and reliability issues. The approach used in P2PS is decentralized, relying on automatic discovery using discovery pipes where possible, and an ad-hoc network of rendezvous peers liking discovery subnets, as illustrated in Figure 1. While this approach is more scalable than a centralized approach it generally means service discovery cannot be guaranteed. A similar approach is used in JXTA. We discuss discovery subnets/pipes and rendezvous peers in more detail in Sections 3.3 and 3.4 respectively.

Underlying all other infrastructure is the ability for communication channels to be opened between peers. In P2PS there are two types of communication channel:

Pipes - Virtual communication channels not bound to specific endpoints or transport protocols.

Endpoint-to-Endpoint - Actual communication channels between specific network locations (endpoints).

P2PS defines an endpoint resolver protocol that allows virtual pipes to be bound to specific endpoints at connection time. Multiple resolver services may be available to handle the endpoint resolution for a single pipe, allowing endpoint-to-endpoint channels to be setup over different transport protocols (e.g. TCP or UDP) or relayed between protocols (e.g. TCP to UDP). Endpoint resolvers are described further in Section 3.5.

The architecture of the current Java implementation of P2PS, as shown in Figure 2, reflects the P2PS infrastructure elements outlined above. Pluggable resolvers allow a peer to communicate using different protocols, and for new resolver implementations to be inserted easily and without affecting the application code. As the discovery service delegates the discovery pipe implementation to the pluggable resolvers, novel discovery pipe implementations can also be easily inserted. Note that the services within the Java implementation should not be confused with peer services (pipes, endpoint resolvers etc.).

3.1 Advertisements

In P2PS, the common mechanism for communicating and querying the capabilities of other peers is through XML based advertisements. The root name of an advertisement denotes its type, and its elements contain the advert id, the originating peer's id, and other information specific to the advertisement type. The following is an example pipe advertisement:

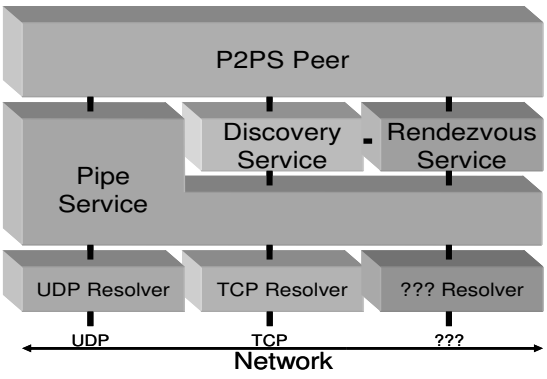

Figure 2: Architecture of a P2PS Peer in the current Java implementation.

```
<?xml version=1.0 encoding=UTF-8?>
<PipeAdvertisement>
  <advertId>advert id</advertId>
  <peerId>originating peer id</peerId>
  <pipeId>pipe id</pipeId>
  <pipeName>pipe name</PipeName>
  <pipeType>pipe type
    (e.g. Standard, Bidirectional, Discovery</pipeType>
</PipeAdvertisement>
```

For each standard capability within P2PS there is a standard advertisement type, these are:

PipeAdvertisement - A pipe is a named virtual communication channel that is only bound to specific endpoints at connection time.

ServiceAdvertisement - A service is named collection of pipes.

EndpointAdvertisement - An endpoint is an address for sending and receiving data using a specified endpoint protocol (e.g. TCP).

EndpointResolverAdvertisement - An endpoint resolver is itself an endpoint that is used to associate actual endpoint points with virtual pipes.

RendezvousAdvertisement - A rendezvous is a specialized peer that provides query forwarding endpoints and advertisement caching.

GroupAdvertisement - A group is a named set of peers used to limit the scope of advertisements. Groups can enforce membership and security requirements.

The XML for all of the standard advertisements can be extended to include additional information if required and completely new advertisement types can be created.

3.2 Queries

Queries are used to locate advertisements that match their query parameters. As queries are themselves advertisements, they are published/discovered in exactly the same way as for other advertisements (see Section 3.3).

An advertisement is denoted as a query through containing a query element, which gives the type of advertisement the query is interested in. Additional query tags can be used to match elements within an advertisement, for example a queryPipeName element in a query is matched against the pipeName element in the original advertisement. Every query should also have either a reply pipe advertisement or a reply endpoint address included so that matching advertisements can be returned directly to the querying peer. The following is an example of a pipe query:

```
<?xml version=1.0 encoding=UTF-8?>
<PipeQuery>
  <advertId>advert id</advertId>
  <query>PipeAdvertisement</peerId>
  <queryPeerId>optional peer id to match</queryPeerId>
  <queryPipeName>optional pipe name to match</queryPipeName>
  <queryPipeType>optional pipe type to match</queryPipeType>
</PipeQuery>
```

As with advertisements, there are standard queries for the standard capabilities, e.g. PipeQuery, ServiceQuery. These standard queries can be extended and new query types created as long as the simple rules described above are followed.

3.3 Discovery Subnets/Pipes

One of the standard P2PS pipe types (see Section 3.5) is the discovery pipe. A discovery pipe is a bidirectional pipe that allows a peer to broadcast advertisements/queries to all other peers within a certain subnet (referred to as the discovery subnet), and to receive the advertisements broadcast by the other peers within the subnet. The implementation of the discovery pipe concept is delegated to individual endpoint resolvers (see Section 3.5), and their implementation will determine the scope of the discovery subnet. For example, a typical discovery pipe implementation would be to use UDP multicast, and in this case the discovery subnet would coincide with the UDP multicast range. Other implementations of the discovery pipe could be using Bluetooth or even JXTA.

Most peers within a discovery subnet are expected to maintain a cache of the advertisements and queries that they published, and to reply directly to queries that they receive. Edge peers such as PDAs and mobile phones that do not have the capacity (processing/storage) to maintain a cache rely exclusively on the presence of rendezvous peers to cache their advertisements.

3.4 Rendezvous Peers

P2PS defines a mechanism for linking discovery subnets based on queries being forwarded between special rendezvous peers. Each rendezvous peer provides at least one endpoint to which other peers (usually from other discovery subnets) can directly forward advertisements/queries. The default behavior in P2PS is that rendezvous peers maintain a cache of all the advertisements they discover/receive but only forward the queries to other rendezvous. This policy

should ensure all queries are fully answered; however other caching/forwarding policies exist and remain a research issue.

The choice of whether to become a rendezvous peer is left to individual peers. In some situations, such as head nodes on a cluster or a PC in a network of low-power edge peers, this choice is obvious. In situations where each peer is equally capable, a peer joining the network could automatically discover whether sufficient rendezvous peers already exists in their subnet, and if not then dynamically become a rendezvous. Similarly, the peer could periodically poll the discovery subnet to check sufficient rendezvous are still alive.

Two other open issues regarding rendezvous peers are 1) how a rendezvous peer joining the network first locates other existing rendezvous peers and 2) how connectivity in the dynamic network of rendezvous peers is maintained. Currently in P2PS the location of an existing rendezvous peer must be expressed explicitly in the peer configuration file, but we are considering implementing rendezvous lookup services located at well known network addresses. Once a rendezvous has joined the network it attempts to maintain connections with three other rendezvous peers it chooses at random. The Java P2PS implementation allows different rendezvous implementations to be plugged in, so different solutions to the open issues outlined can be employed.

3.5 Pipes and Endpoint Resolvers

In P2PS a pipe is a named virtual communication channel that is only bound to specific endpoints at connections time. While multiple pipes can share the same name, each is also identified by a unique id and the id of its originating peer. A pipe also has a specified type; the standard pipe types are:

Standard - Assumed to be unidirectional, unreliable and not secure, but the actual properties depend on the transport protocol.

Bidirectional The same as a standard pipe but with bidirectional communication.

Discovery - Allows messages to be broadcast to and received from other discovery pipes within a certain subnet, as described in Section 3.3.

P2PS allows custom pipe types to be specified, with obvious extensions including *reliable* and *secure* pipes.

The services responsible for establishing the binding between pipe advertisements and specific endpoints are referred to as endpoint resolvers. To resolve a pipe, a peer must first discover at least one endpoint resolver that can handle that pipe. As endpoint resolver advertisements include the peer ids and pipe types handled, this discovery simply involves publishing an endpoint resolver query containing the relevant peer id and type for the pipe (see Section 3.2).

Endpoint resolver advertisements contain one or more endpoint addresses, which when sent an endpoint query, return an actual endpoint address for the pipe specified in the query. It is acceptable to have multiple endpoint resolvers for a peer, each handling different transport protocols. The actual protocol used by a pipe depends on the protocols handled by the connecting peer and the available resolvers.

In Figure 3 we illustrate the steps involved in resolving a pipe. In this example there are two endpoints for the pipe (TCP and UDP) and two endpoint resolvers. However, as the connecting peer (PeerB) only has a TCP connection, the pipe is established only using the TCP resolver. Although in this example the endpoint resolvers are shown as third parties to the peer that created the pipe (PeerA), the more common case is that a resolver for a peer is located at that peer.

3.6 Peer Groups and Security

The concept of peer groups is currently being developed within P2PS. A peer group is a dynamic subnetwork of cooperating peers that can be used to limit the scope of advertisements/queries. In a global-scale network such groupings are essential in reducing the burden on rendezvous peers and increasing the network performance, and also to enforce security between cooperating peers.

As with standard P2PS networks, groups employ rendezvous peers to cache/forward advertisements within the group. However, in groups each edge peer maintains a direct connection to at least one rendezvous peer (as opposed to going via a discovery subnet). In P2PS, groups are handled in the same way as pipes, with group advertisements advertisingthe existence of a virtual peer group and group resolvers binding virtual groups to specific endpoints. In the case of groups, the endpoint address returned by the resolver is that of a group rendezvous peer.

Group resolvers also act as authorization services for the group. When a peer contacts the group resolver (using a secure pipe), it supplies a credential identifying itself to the resolver. If the credential is valid for the group, the resolver can authorize the peer join the group and supply that peer with encryption keys for sending and receiving messages from the group. Alternatively, the application to join the group can be

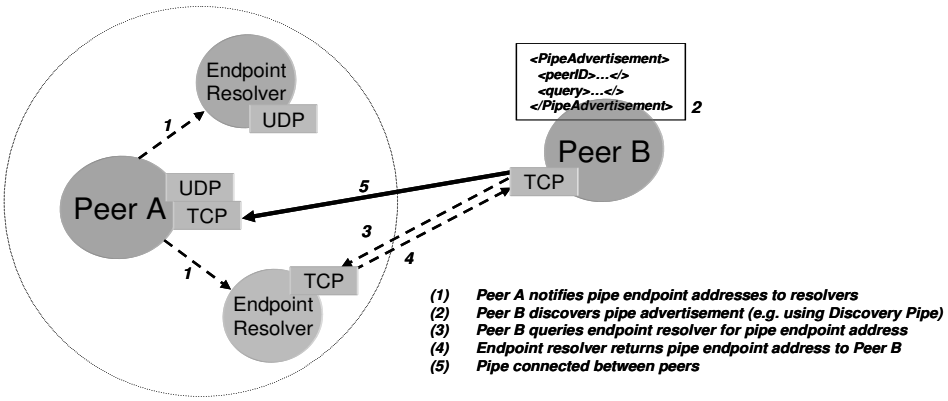

Figure 3: Endpoint resolution example using TCP and UDP resolvers.

rejected. The types of credential (e.g. X.509) accepted by the authorization service are specified in the group advertisement. Also, as a peer can belong to multiple groups and groups can be nested, different levels of authorization/security can be established.

4 Summary and Future Direction

In this paper we introduced P2PS, a lightweight infrastructure for peer-to-peer service discovery and pipe-based communication. An overview was given of the P2PS network architecture and key infrastructure elements, including discovery pipes, rendezvous peers and endpoint resolvers. The application view of P2PS was also discussed, and an example given that illustrated a basic application created using P2PS.

The aim of P2PS is to provide a lightweight infrastructure for developing peer-to-peer applications, while providing the flexibility to transparently incorporate different discovery and communication protocols. As discussed in this paper, pluggable endpoint resolvers allow additional endpoint implementations to be introduced into P2PS without affecting the application code. The default discovery and rendezvous service implementations can also be replaced, allowing different discovery algorithms to be tested.

The next key stage for P2PS is the incorporation of peer groups and the associated security. Once this is complete the focus will shift to testing P2PS in applications environments, principally in the sensor network and task farming applications in which P2PS has already been integrated.

P2PS is an open source project. More information can be found at `www.trianacode.org/p2ps`.

References

[1] AOL Instant Messenger, *http://www.aim.com/*

[2] Buzzpad, *http://www.buzzpad.com/*

[3] Gnutella, *http://www.gnutella com/*

[4] Emir Halepovic and Ralph Deters, JXTA Performance Study, PACRIM'03 Conference, Victoria, BC, Canada, 2003

[5] JINI, *http://www.sun.com/jini*

[6] Project JXTA, *http://www.jxta.org/*

[7] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, Zhichen Xu, Peer-to-Peer Computing, Technical Report HPL-2002-57, HP Lab, 2002

[8] Napster, *http://www.napster.com/*

[9] Rdiger Schollmeier, A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications, In Proceedings of the IEEE 2001 International Conference on Peer-to-Peer Computing (P2P2001), Linkping, Sweden, August 27-29, 2001

[10] SETI@Home, *http://setiathome.ssl.berkeley.edu*

[11] Bernard Traversat, Ahkil Arora, Mohamed Abdelaziz, Mike Duigou, Carl Haywood, Jean-Christophe Hugly, Eric Pouyoul, Bill Yeager, Project JXTA 2.0 Super-Peer Virtual Network, Sun Microsystems Inc, *http://www.jxta.org/white_papers.html*