PSIRP
Publish-Subscribe Internet Routing Paradigm
FP7-INFSO-IST-216173

DELIVERABLE D3.4

Integration and Demonstration Plan

Title of Contract	Publish-Subscribe Internet Routing Paradigm
Acronym	PSIRP
Contract Number	FP7-INFSO-IST 216173
Start date of the project	1.1.2008
Duration	30 months, until 30.6.2010
Document Title:	Integration and Demonstration Plan
Date of preparation	31.08.2009
Author(s)	Dirk Trossen (BT) (editor), Janne Tuononen (NSNF), Jimmy Kjällman (LMF), Borislava Gajic (RWTH), Dmitrij Lagutin (HIIT), Somaya Arianfar (LMF), Walter Wong (Unicamp), Mark Ain (HIIT) (editor)
Responsible of the deliverable	Dirk Trossen (BT) Phone: +44 7918 711695 Email: dirk.trossen@bt.com
Target Dissemination Level:	PU
Status of the Document:	Completed
Version	1.0
Document location	http://www.psirp.org/deliverables/
Project web site	http://www.psirp.org/

Table of Contents

This document has been produced in the context of the PSIRP Project. The PSIRP Project is part of the European Community's Seventh Framework Program for research and is as such funded by the European Commission.

1 Introduction

From the design phase of the project, the prototyping work is a major component in the development of the architecture. While implementation work on different architectural components started early in the project, integration of these components into a coherent system prototype will gradually become the main focus for the remainder of the project.

This deliverable presents the current state of integration at the node level as well as the network architecture. We base our presentation on the current PSIRP architecture that was outlined in D2.3 [Tro2009]. The reader is referred to this deliverable for a deeper understanding of the various components that we will present in the following.

The deliverable starts with the node architecture as outlined in D2.3. The status of work for each of the components within that node architecture is presented followed by an outline of our plans for integration of the various components. The nature of this integration, as opposed to strictness of timing, is of critical emphasis as it takes place across various distributed teams. As for general timing, we expect most of the presented integration work to take place in the second half of 2009, reaching into 2010 – just in time for the final update of the system architecture.

After the node architecture, we move on to the network architecture, based on D2.3, and similarly outline the current status of the various components before presenting the plans for integration. It is important to note that the status of the various network components is less advanced than the node architecture itself, which is due to the ongoing design work in some of the areas, such as for the inter-domain topology formation. Hence, we expect final integration to take place towards the end of the project.

Apart from integration, the aspect of demonstration is also a part of this deliverable. PSIRP clearly stated from the beginning that the application development is not the focus of our work. However, demonstrating the ability of the architecture is still required. Our presented demonstrators focus on this issue rather than the implementation of various applications. For that, we outline a laboratory demonstrator for testing and demonstration purposes and furthermore present an extension towards a campus-wide test bed that allows for demonstrating, e.g., forwarding or rendezvous capabilities, but also enables testing of various components. While the laboratory demonstrators are already available, based on the old as well as the revised node architecture, the campus-wide demonstrator is currently under development with an expectation of a growing functionality towards the end of 2009.

2 Node-level Integration

This first section describes the current and planned integration at the node level, based on the current node architecture as outlined in D2.3 [Tro2009]. We start with a brief overview of the current node architecture before outlining the status of the node components and the plans for further integration.

2.1 Current Node Architecture

The current node architecture, which is described in more detail in deliverable D3.3 [Jok2009b], implements parts of the PSIRP service model and API presented in D2.3 [Tro2009]. It consists of the following pieces:

- A *blackboard* that implements the memory object model from D2.3 inside a node. Conceptually, the blackboard is the place where data items are stored as publications that can be subscribed to.
- An *API* for publishing data items to the blackboard, subscribing to them, and getting notifications when new versions of them are published.
- Applications that can communicate with each other via the blackboard. This includes *helper applications* that implement different functions of the PSIRP *component wheel* (see Figure 2.1).

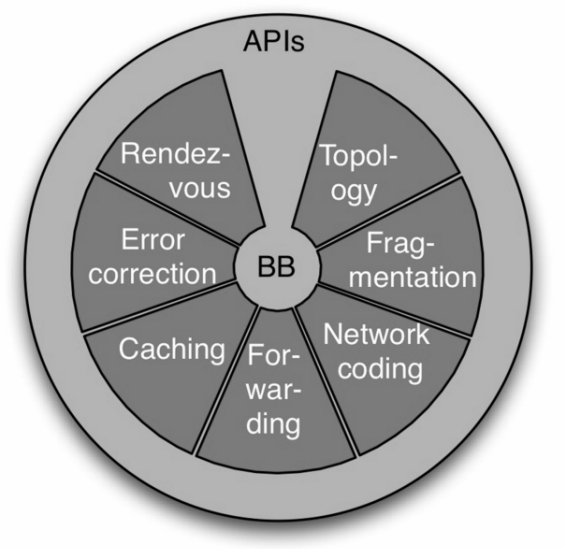

Figure 2.1 – PSIRP Component Wheel

In the Blackhawk prototype for FreeBSD, the blackboard is implemented as a kernel module, as shown in Figure 2.2. It is integrated with the operating system's virtual memory system, i.e., publications in the blackboard correspond to virtual memory objects and pages. The API is implemented as a library that communicates with the kernel module using system calls. It provides functions for creating, publishing, and subscribing to publications. Notifications about publish operations on the blackboard can be acquired via the *kevent* system of FreeBSD. In addition, a file system view to the blackboard is provided which shows the blackboard's contents as a hierarchy of scopes, publications, publication versions, and memory pages (which all have their own RIds).

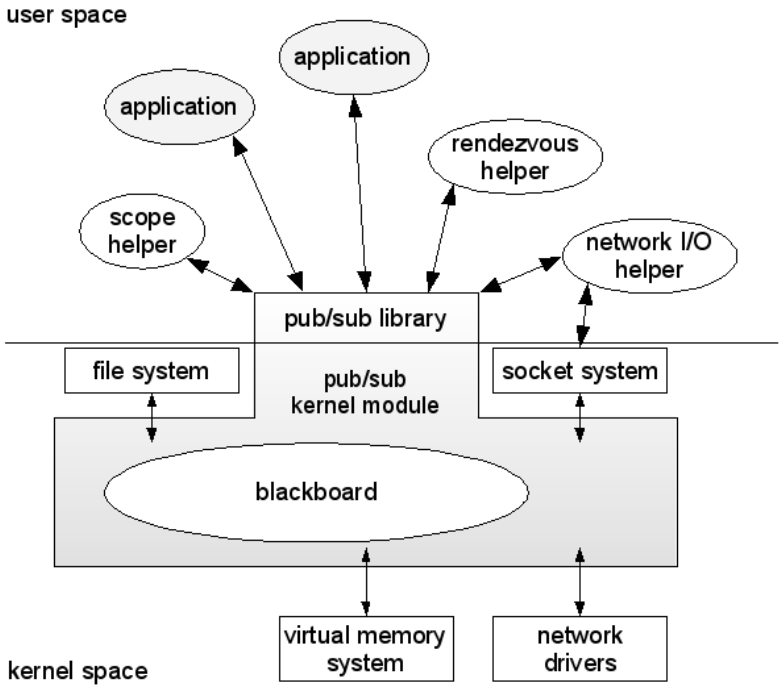

Figure 2.2 – Node Architecture

As mentioned above, user space applications can use pub/sub-based inter-process communication by publishing data to the blackboard, from where other processes can retrieve the publications by subscribing to them. The component wheel functions (e.g., rendezvous and topology management) are also implemented as applications. However, some parts of the component wheel (e.g., forwarding) may also be implemented in the kernel space to achieve better performance. As an example of helpers, we can look at the current version of the Blackhawk prototype (Figure 2.2) that features three helpers:

- A *scope helper* takes care of instantiating, updating, and re-publishing scope publications, i.e. data items that contain collections of RIds.
- A *local-area rendezvous helper* extends the blackboard model into the network It provides local monitoring for publish/subscribe operations, and it advertises the locally published publications to the local-area rendezvous node, used in the first version of the prototype..
- A *network I/O helper* implements packet fragmentation/assembly in addition to forwarding, using sockets for sending and receiving packets over links in the network.
    - The rendezvous helper communicates with the network I/O module when publication metadata or data needs to be sent into the network. In the reverse direction, the network I/O helper dispatches received metadata to the rendezvous helper.

2.2 Status of Components

This section outlines the status of the node components based on the node architecture, presented in Figure 2.2.

2.2.1 Blackboard

As described in Section 2.1, the blackboard, responsible for node-internal publication management, is implemented as a FreeBSD kernel module (named *psfs*). In the blackboard,

publications correspond to memory objects, whose metadata point to version objects, whose metadata then point to memory pages. A rendezvous structure, called a *Publication Index Table* (PIT), maps *Rendezvous identifiers* (Rids) to the memory objects, version objects, and memory pages. This hierarchy is currently exposed by the file system API, which allows applications to access data content on these different levels. Sharing of identical pages between different versions of a publication has been partially implemented.

A pub/sub library (*libpsirp*) allows programmers to create, publish, and subscribe to data on the blackboard. The library is implemented in C and has wrappers for Python and Ruby, however those in C and Python are currently much more developed. Using this API, data in the blackboard can be accessed as arrays after creating a new or subscribing to an existing data object. The subscribed data object can be written to, in which case the affected page is first copied and then modified. The original version of the publication remains intact, and no garbage collection has yet been implemented. The subscribed and modified data can also be re-published, resulting in a new version of the publication. When a publication is updated, a *kevent* is issued In addition to this functionality, the library provides various functions for accessing publication metadata and for practical handling of binary PSIRP identifiers (e.g., hexadecimal format conversions).

A scope helper (*scoped*) in the user space assists the blackboard in creating and updating publications that list the publications in a scope. Sub-scopes can be created under the root scope, but currently not under other scopes.

At the time of writing, the implementation has some known limitations and bugs. The publication size, number of publications in a scope, as well as the number of versions of a publication are currently limited.

2.2.2 Forwarding

Basic zFilter-based forwarding (see [Jok2009a]) has been implemented into the prototype as part of the network I/O helper application, making forwarding decisions by matching the *Forwarding identifiers* (Fids) of outgoing packets with the static *Link identifiers* (Lids) that are read from a configuration file and assigned to the interfaces. The prototype also supports return-path collection into the forwarded packets. The collection operates by ORing the incoming interface Link IDs to the return Fid field in the packet on each forwarding node.

In addition, the network I/O component handles the sending and receiving of packets within broadcasted Ethernet frames through link-layer sockets. It also takes care of fragmentation, that is, it splits large publications into multiple smaller packets to be sent and assembles data from received packets into publications.

The network I/O module can communicate with the local rendezvous module (the status of which is described below) through the node-local blackboard. However, due to the limitations and stability issues of the current blackboard implementation, local socket-based IPC is used as the default communication mechanism in the first prototype release.

2.2.3 Rendezvous

The network rendezvous mechanism integrated with the first prototype release is intended to be used only in small LANs. It is not integrated with the other rendezvous node implementation presented later in this document nor with the topology manager. The implementation language of this rendezvous daemon is Python. In each node, the daemon listens to publications appearing under the root scope and its sub-scopes, to publication updates, and also to subscriptions that take place in the node. Whenever events like these occur, the daemon initiates network pub/sub via the network I/O daemon. The daemon keeps state for pending subscriptions initiated within the node itself and handles rendezvous signals received from the network.

Moreover, nodes have a pre-configured FId pointing to a local rendezvous node. That node runs the same daemon but stores metadata about publications published in other nodes,

responds to metadata subscriptions, and dispatches data subscriptions to the original publishers.

The current version has several limitations; it is only possible to publish and subscribe whole, single-version publications, only the publish-before-subscribe case is supported and only one publication can be received at a time.

2.2.4 Network Attachment

The network attachment daemon was written in Python as an independent software module that has not been integrated with other components in the current prototype. It implements a simple pub/sub-based framework for attachment point discovery, communication bootstrapping, and node configuration. Instead of proper PSIRP SIds, RIds and FIds, this version uses generic labels as message identifiers. It also establishes its own link-layer connections and has an internal blackboard.

2.2.5 PLA

Packet Level Authentication (PLA) functionality has been implemented as a separate library, which was used by the psirpd in the previous prototype to perform PLA operations. This library supports packet verification and signature generation for multiple modes of operation, including cases where the RId and SId are derived from a corresponding public key. Additionally, it contains functionality for certificate creation and management at runtime.

2.2.6 Topology Manager

The Topology Management module maintains local connectivity information in the form of a generic table containing information about an outgoing node's links as row entries. Each link is represented by its ID and type, but the link information can be extended with a set of link related properties, e.g., throughput, delay and ID(s) of the node(s) attached to the other side of the link. This additional link information can be provided by helper functions collecting relevant link data. The node-level functionality of Topology Management can be separated into two main interfaces: updating the table entries for each link the node is connected to and publishing the link information table locally. The interface for updating the table entries requires information about link IDs generated by a responsible entity and distributed during the initialization phase, e.g., the network attachment phase in the form of a configuration file. Moreover, it receives initial link attribute values during the initialization and updates the values upon detecting changes in the link condition. The format of table updates needs to be specified in advance, and in an early stage of the Topology Management implementation prototype, this format can be a simple text-based one. On the other hand, the interface for publishing the link information table subscribes to link table information, parses the contents and publishes the updated version of the table. The publication of link attribute information must occur periodically or on-demand attribute value changes exceed predefined thresholds. In order to optimize performance, a node can publish the table containing only differences between previous and current parameter values.

2.2.7 Applications

The PSIRP Firefox plug-in provides mechanisms for users to subscribe to publications using the PSIRP protocol through their web-browser. The plug-in intercepts all PSIRP protocol calls in the address bar or in a link embedded in a webpage (e.g. psirp://), passing the PSIRP parameters (SId and RId) to the XPCOM component. This component interacts with the PSIRP library by subscribing to the publication identified by the SId and RId pair and, after retrieving it, the component saves it as a local file. Finally, the plugin opens the fetched publication and displays it in the web-browser. Currently, the retrieved publication is saved as a local file, which is later opened by the web browser. As a future improvement, the publications will be displayed directly in the web browser without requiring copies to be saved as local files.

2.3 Integration and Improvement Plan

This section outlines integration and improvement plans for the node architecture implementation.

2.3.1 Blackboard Core Improvements

The blackboard-core and the essential helpers (network I/O and local rendezvous) currently have known limitations that need to be addressed during the upcoming integration phase. The (not exhaustive) list of planned improvements includes:

- Increasing the limit on the maximum number of items in a scope, the maximum size of a publication and the maximum number of versions of a publication.
- Supporting publication versioning in the networking portion.
- Allowing for subscribing to parts of publications and support demand paging over the network.
- Implementing Blackboard garbage collection as currently publications are never freed. This relates to implementing a node-internal caching helper that maintains references to publications.
- Improving access control and security features through the correct use of scopes.
- Reviewing the higher-level view of the blackboard through the pub/sub API, especially for the object-orientated language implementations.
- Adding support for scopes-in-scopes.
- Implementing page sharing between different publications.
- Optimizing various components, including in-kernel rendezvous.
- Implementing configuration information for the forwarding and local rendezvous functions may need configuration information (Ids, link connectivity, etc.) that can be specified by other components. For example, the table of link identifiers should be read from the blackboard instead of a static file.

2.3.2 PLA Integration

There are two possibilities for integrating PLA functionality into the blackboard based prototype, PLA functionality can either be a standalone helper application or it can be included in the existing network I/O helper. We decided to initially implement the latter option, since it is a straightforward solution, while the internal structure of the blackboard prototype will continue to change. The implementation would work as follows: as packets enter or leave the network I/O helper, PLA related functions such as signature verification or addition, will be performed according to the local security policy.

There are plans to eventually create a separate PLA helper application. This is a better fit to the blackboard prototype's design philosophy and allows PLA's cryptographic functionality to also be used for other functions, such as local authentication.

2.3.3 Network Attachment Integration

The network attachment daemon needs to be revised so that it uses the blackboard for communication with other components. For example, signalling identifiers used by the rendezvous and topology functions will probably need to be exchanged. This component can also provide information about link connectivity. In addition, inter-node communication should utilize SIds, RIds and FIds correctly, according to the model outlined in D2.3. Through the network attachment process, nodes joining networks can learn signalling IDs and other necessary configuration information.

2.3.4 Transport and Caching Integration

The current prototype lacks transport functionality, including flow and congestion control as well as reliability. The transport functionality will be tightly coupled with caching.

In routers, caching should be implemented instead of using queuing. A router cache can be used to assist both push- and pull-based transport protocols. In the packet-level push mode, data packets are pushed out by the source. Sinks send (re)transmission requests only when they discover that some packets were lost or when they reconnect after a short disconnection period. Such requests can also come, in a more pull-like mode, from a new node that has just received a new meta-data object. In the packet-level pull mode, where sources do not actively push any data packets, the routers can be more active. Instead of relying on the source to push or the sink to pull the data, a router can actively pull data from the source, adjusting the pull rate to its cache capacity, amongst other metrics.

The details of these transport protocols should also be worked out later on the architecture design side. However as an initial effort, a pull-based flow control scheme that utilizes token buckets has been planned to be implemented.

2.3.5 Application: Firefox Integration

The integration of the Firefox plug-in into the prototype is straightforward since the plug-in is a standalone package using the PSIRP library. First, the user must have a running version of Firefox 3.x and the Blackhawk prototype installed on their machine. For the installation, the user needs to open the plug-in package (*psirp_plugin.xpi*) using the web browser and finally, the PSIRP Firefox plug-in will be ready to use after the web browser has installed the plug-in.

The following succession of four figures shows the PSIRP Firefox plug-in in action. Figure 2.3 shows the plug-in installation, which is straightforward - the user merely needs to open the plug-in package and restart the browser, after which the browser will be PSIRP-enabled.

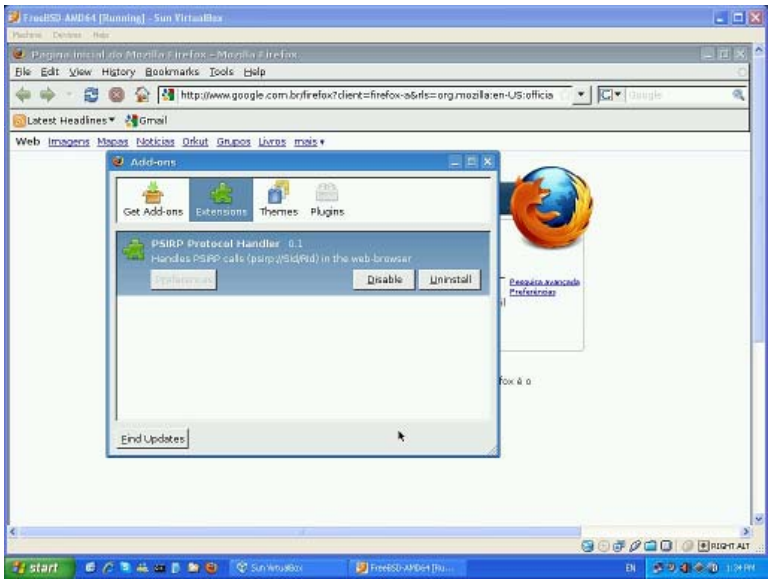

Figure 2.3 – PSIRP Firefox Plugin Installation

Figure 2.4 shows the PSIRP-enabled Firefox subscribing to a publication identified with SId/RId ::/12::34

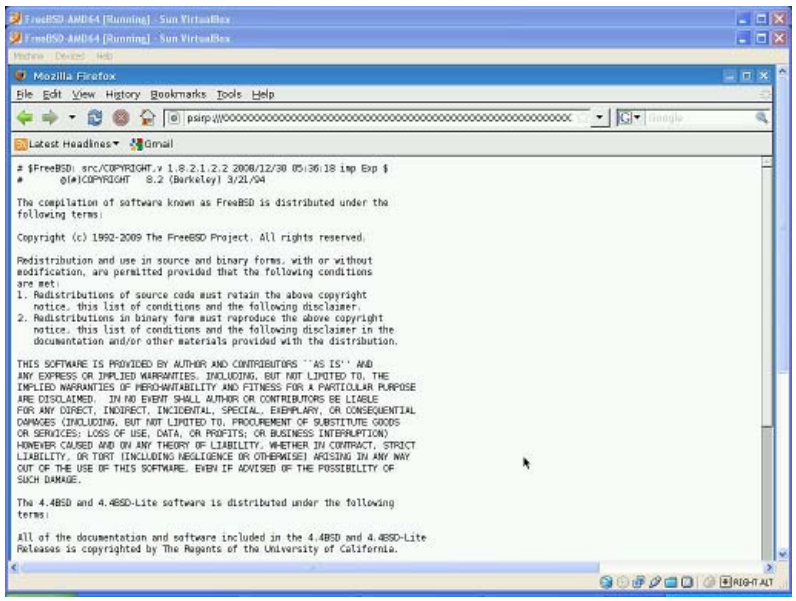

Figure 2.4 – Subscribing to a publication (SId, RId) = ::/12::34

Figure 2.5 illustrates a web-page with an embedded PSIRP link which forwards to a new publication identified by SId/RId ::/::2

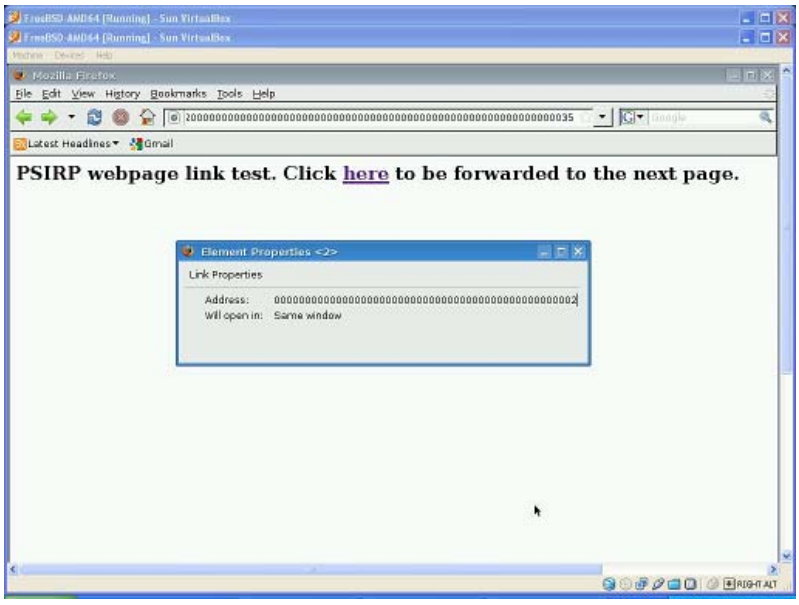

Figure 2.5 – Embedded link with PSIRP identifiers (SId, RId) forwarding to a new publication

Figure 2.6 shows the new publication forwarded from the previous webpage.

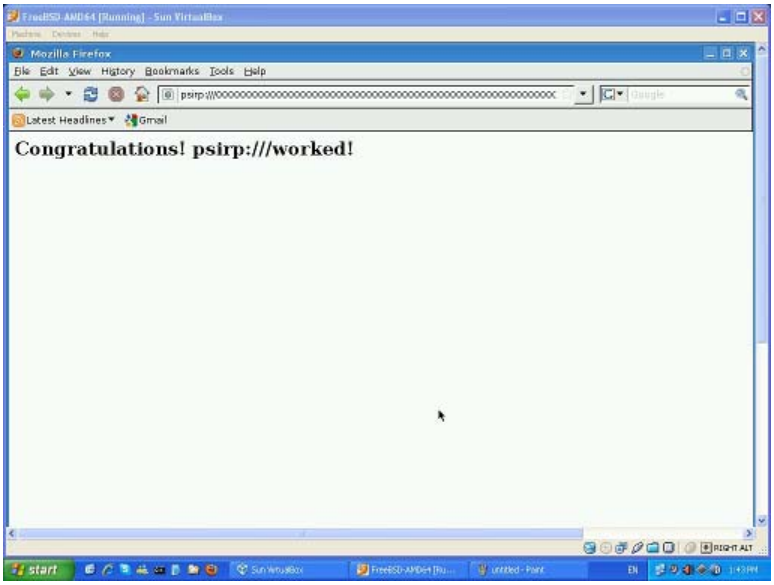

Figure 2.6 – Successful subscription to the new publication

If the publication does not exist, then the browser will return the 'File not Found' error.

3 Network-level Integration

This section outlines the integration of the network-level components, based on the node architecture presented in Section 2. The components in this section are based on the current network architecture [Tro2009], presented in the following Section 3.1.

3.1 Current Network Architecture

The following brief presentation of the PSIRP network architecture is based on D2.3 [Tro2009]. Figure 3.1 shows the current network components and their relationships.

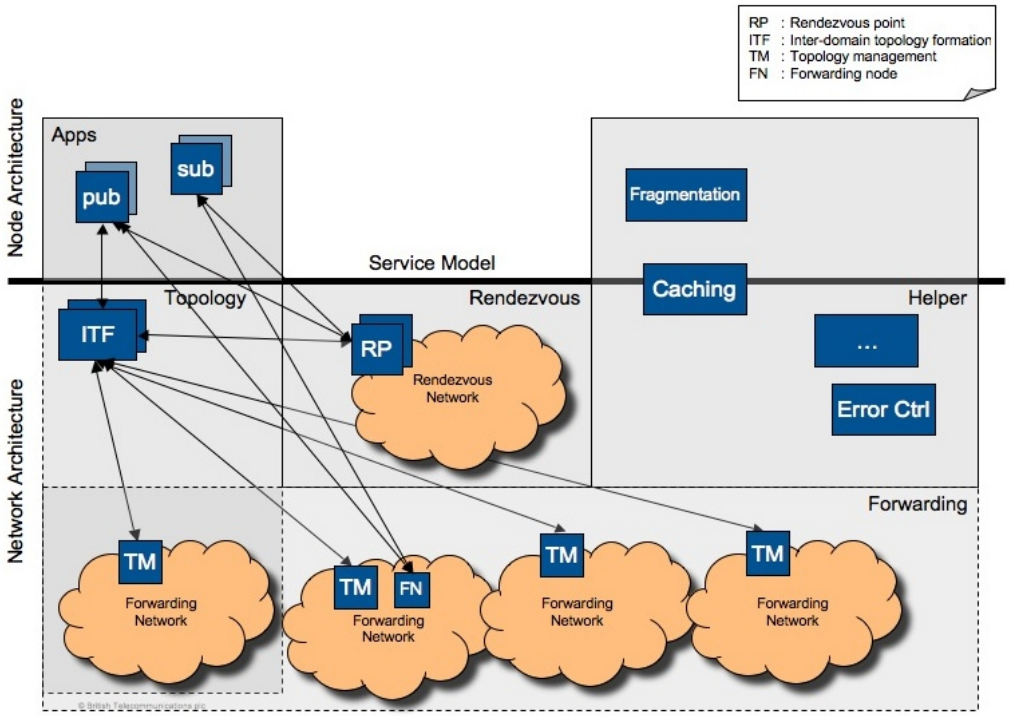

Figure 3.1 – Network Architecture

Forwarding networks represent the physical delivery infrastructure in our architecture. *Forwarding nodes* (FN) provide forwarding functionality from publishers of data to subscribers of data. Rendezvous points (RP) within *rendezvous networks* perform the matching function between publications and subscriptions within particular scopes of publication (see D2.3). After such a match has been found, an inter-domain topology is constructed in dialogue between the RP, the *Inter-domain Topology Formation* (ITF) function and the *Topology Management* (TM) functions of traversing forwarding networks. The topology management function in each forwarding network is responsible for constructing appropriate (intra-domain) forwarding paths within its realm, i.e., the forwarding network. Helper functions of various kinds exist, e.g., for caching, error control, fragmentation, or path discovery.

The network architecture exposes a publish-subscribe service model to applications implementing *pub* and *sub* functionality. These applications reside with nodes that are based on our presented node architecture of Section 2.

A more detailed description of the network architecture can be found in D2.3 [Tro2009].

3.2 Status of Components

The following section outlines the status of implementation, based on the components outlined in Figure 3.1.

3.2.1 Forwarding Node

The project currently has two forwarding node implementations that are compatible with each other:

- The Blackhawk prototype for FreeBSD includes packet forwarding functionalities as part of the network I/O module.
- A hardware-level forwarding node implementation has been implemented on NetFPGA hardware.

As mentioned before, the Blackhawk prototype implements simple zFilter-based forwarding. Currently it is possible to configure one link identifier per interface; support for multiple link identifier tags or identifiers for virtual trees is not yet implemented. Other missing features are TTL support (for loop prevention) and checking the number of 1-bits in FIds (for security and safety). The prototype for NetFPGA, on the other hand, does implement the more advanced features that the Blackhawk implementation is lacking.

3.2.2 Rendezvous Node

In this section, the status of the Rendezvous Node (RN) helper function implementation is described. This text assumes that the reader is familiar with the terms and structure of the PSIRP rendezvous system described in the conceptual architecture [Tro2009] and the contents of the implementation progress report D3.3 [Jok2009b].

The current version of the RN helper function extends support for rendezvous from the single-RN architecture to the multi-RN architecture and thus, it is an initial realization of the inter-domain (inter-RN) rendezvous system defined by the rendezvous design team. The implementation, to a large extent, supports the functionality needed for the creation and management of a rendezvous network, and supports rendezvous for publish/subscribe signalling from the hosts that are within reach of this network. A rendezvous network is a logical network where a group of RN entities have established a policy compliant rendezvous network topology. A minimal size rendezvous network contains only one RN. From the complete inter-domain rendezvous system the implementation excludes support for interconnection overlay, which by design utilizes communication between multitudes of rendezvous networks. However, the implementation can be inherently configured to support communication between two rendezvous networks in the same manner it supports connecting to a rendezvous node via peering. The current rendezvous node implementation is a stand-alone version, being realized using Ruby-1.8.

Figure 3.2 illustrates its overall architecture, outlining three major parts to its implementation.

- The rendezvous node module containing functions for rendezvous signal handling, rendezvous network management, rendezvous point execution, MySQL interfaces and stub function interfaces.
- The MySQL database module (including the Ruby-MySQL-API) containing all rendezvous node and rendezvous point data tables:
    - Scope cache, containing scope identifier (SIds) entries of all scopes known by the rendezvous node.
    - Rendezvous node neighbourhood, containing information about known rendezvous node neighbours, guiding the behaviour of the rendezvous point search function.

- Scope specific publication/subscription tables containing information about active publications and subscriptions.

- Stub functions, such as for forwarding and topology that were implemented to enable the testing of actual rendezvous node functionality in a rendezvous network topology.

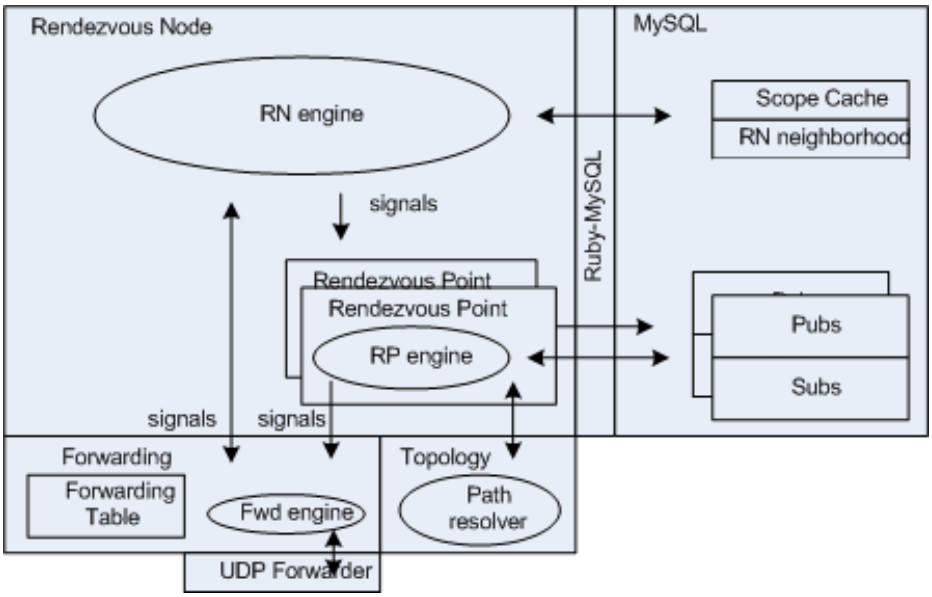

Figure 3.2 – Rendezvous Node Implementation Architecture

While the rendezvous node module and the MySQL database are supposed to be integrated with minor modifications to the complete prototype, the stub function collection is to be replaced by the functionality from the blackboard and topology components.

3.2.3 ITF Node

There is currently no implementation of the ITF node, as the protocol and mechanism design for the ITF function is still ongoing. The first design choice implementations are expected towards the end of 2009.

3.2.4 Topology Management

The network level Topology Management module subscribes to link table publications and to Rendezvous-related events at the same time. As it collects link information in the form of tables, it can store the topology information for the overall network together with the per-link annotations in a flexible graph description format (such as GML). Having all necessary information, this module is responsible for generating the shortest paths within its assigned domain. The notion of a *domain* (area) associated to a particular Topology Management entity can be implemented through predefined scopes. Thus, all communication related to topology information collection and dissemination is done under a particular scope assigned to the topology management. After receiving the publication/subscription matching notification from the rendezvous point, the Topology Management function publishes the shortest path information in the form of a zFilter, taking into account the current link parameters and application requirements. Additional application demands can be contained in the metadata of signalling messages coming from the rendezvous system.

A crucial aspect of this architecture lies in the optimal data distribution. Publishing topology information under a predefined SId/RId pair is one possibility. For this, each Topology Management area is associated with one predefined SId, under which each node is represented as an assigned RId. Therefore, publishing the link state tables will contain

SId/RId pairs unique for a given domain and a given node, while every topology information update will rewrite the existing topology information since it is considered a new version of the same publication. Publishing the zFilter information as the Topology Management function's response to the rendezvous system query can be done using algorithmic identifiers, generated using the Topology Management area SId and RIds of publishing and subscribing nodes. This is illustrated in Figure 3.3.

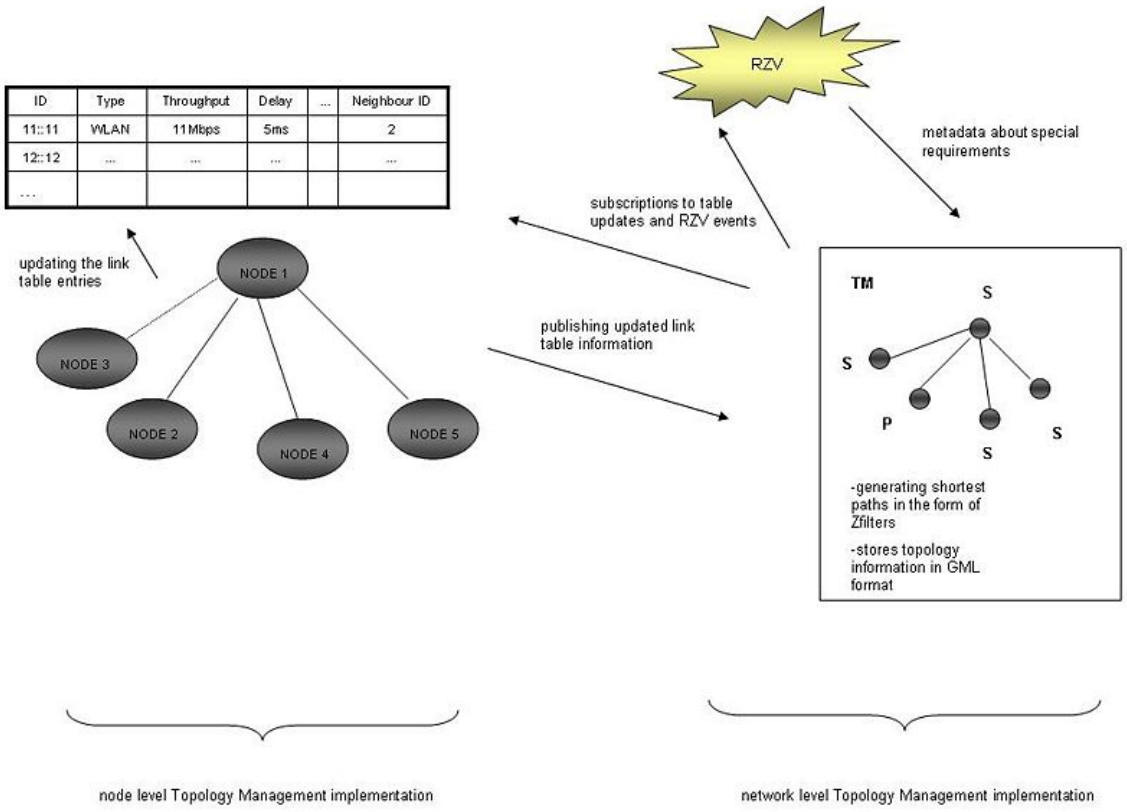

Figure 3.3 – Topology Management implementation

At present, a simple prototype for local and networked cases exists. It is currently being rewritten and will be harmonized with the other components, e.g., network attachment and rendezvous. In order to test the performance of the current prototype, we aim to create larger networks using a set of virtual machines and the ns-3 simulator's emulation environments.

3.3 Integration Plan

The integration plan for the network architecture focuses on two major elements, namely the rendezvous node and the forwarding node. The plans are outlined below.

3.3.1 Rendezvous Node Integration

The integration of the rendezvous node implementation with the blackboard and topology manager components will happen in phases, illustrated in Figure 3.4.

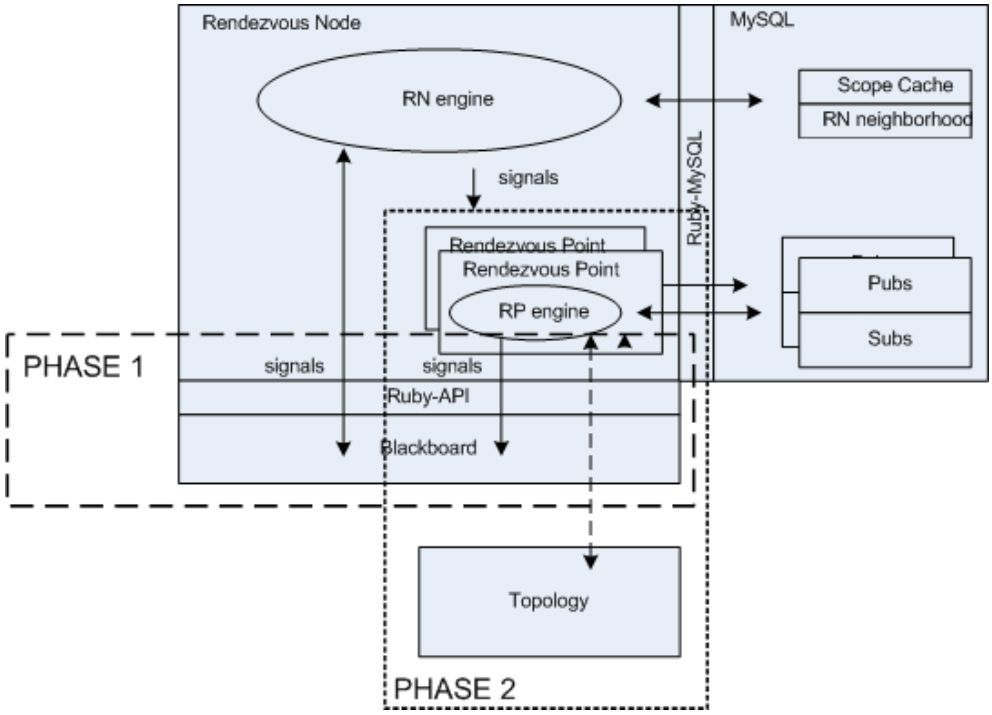

Figure 3.4 – Rendezvous Node Integration Plan

The rendezvous node is a helper application that, at least in theory, can be located at any node in the PSIRP network. The first integration step will be to connect the rendezvous node with the rest of the PSIRP system modules and to reproduce the existing rendezvous features in the stand-alone implementation using the tools provided by the blackboard. In practice, this means replacing the existing forwarding stub with the ruby-based API provided by the blackboard, and synchronization of the used signal formats, if needed. It is assumed that during the first phase, RN-to-RN rendezvous signal formats will remain largely the same because rendezvous signals are publications under the scope identifier subscribed to by the receiving RN.

During the second phase, we will focus on removing the topology stub and establishing the required co-operation with the actual topology manager. Even though the co-operation with the topology function is not critical for the rendezvous node, it is critical for the topology function because the rendezvous system is assumed to be the key provider of pub/sub topology information. It is therefore assumed that the topology-rendezvous co-operation will exploit similar pub/sub based communication as is used in the RN-to-RN communication i.e., the RP provides the required parameters to the topology function as a publication that the topology manager can subscribe to. Currently, it is still open what these parameters will be and what the procedure will be following publication, e.g., who will provide the path to the publisher and in which format.

After these two phases, the majority of the rendezvous integration will have been finished, enabling further implementation work to continue. As the progress of the two phases could be adversely affected by unforeseen issues, a detailed schedule cannot be released. However, it is expected to be completed before December 2009.

3.3.2 Forwarding Node Integration

As the packet format of the prototype is likely to change (during autumn 2009), it could affect both forwarding node implementations in addition to the local rendezvous function. However, as the other components generally deal with higher-level data (which has been processed by other components in the component wheel instead of the raw packet data), they should remain unaffected.

4 Demonstrators

The PSIRP project implements various demonstrators, based on the presented prototypes. In the following, we outline the current laboratory demonstrator as well as the efforts towards a larger networked test bed with one of our external partners, Essex University.

4.1 Current Laboratory Demonstrator

In this section, four different demonstrators are discussed:

- File-based pub/sub demonstrations with the earlier, unpublished, FUSE-based prototype
- zFilter-multicast demonstrations with a UDP/zFilter-proxy
- Stand-alone rendezvous network demonstrator
- Planned demonstrations with the released Blackhawk prototype

4.1.1 Demonstrator with the FUSE-based prototype

Deliverable D2.3 [Tro2009] presented a demonstrator based on an early development version of the prototype for FreeBSD. It enables audiovisual demonstrations suitable for introducing and concretizing the implemented system and pub/sub concepts. Several public demonstrations have been carried out with a setup consisting of three computers connected to each other by separate links.

The architecture of the earlier implementation (the so-called FUSE-based prototype) was described in D3.2 [Jok2008], and it had two important features that the demonstration relied upon. Firstly, metadata was broadcast to all nodes in the network when something was published. Consequently, end-nodes could immediately initialize "empty data containers" for the corresponding publications in their virtual memory systems. They also created entries in their local filesystems for those publications (e.g., /pubsub/0000.../12ab.../data). Secondly, the prototype supported demand paging over the network. When an application started to read a file of the aforementioned type at some location, the corresponding memory pages were subscribed to and fetched from the network. Thus, unmodified legacy applications, such as image viewers and music players, could be used in demonstrations, and the system could even emulate streaming. Moreover, packet authentication with PLA was also part of the system. Figure 4.1 below shows what the protocol looks like in a demo scenario that utilizes these concepts.

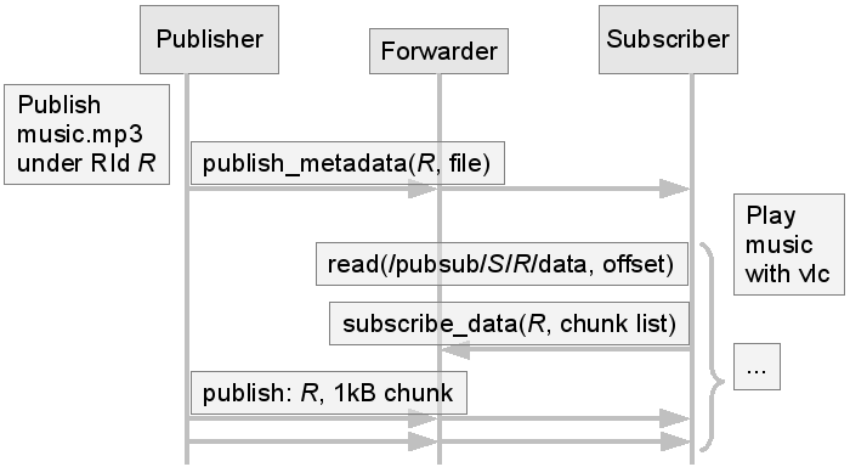

Figure 4.1 – Basic Demonstration Scenario from D2.3

During spring 2009, an evolved demo setup was also created. It included a utility, written in Python, for publishing lists of files in the following way:

Publisher

1. Traverse a directory (or a list of filenames) and publish all files in it.
2. Publish a list of RId-to-path mappings of all the files published in the previous step.

Subscriber

1. Receive metadata about the published files and the list.
2. Subscribe to the content of the list of files.
3. Create a local directory structure with symbolic links from the filenames to the publications under /pubsub/.
4. When those files are accessed, their content is fetched on demand.

These steps are illustrated in Figure 4.2.

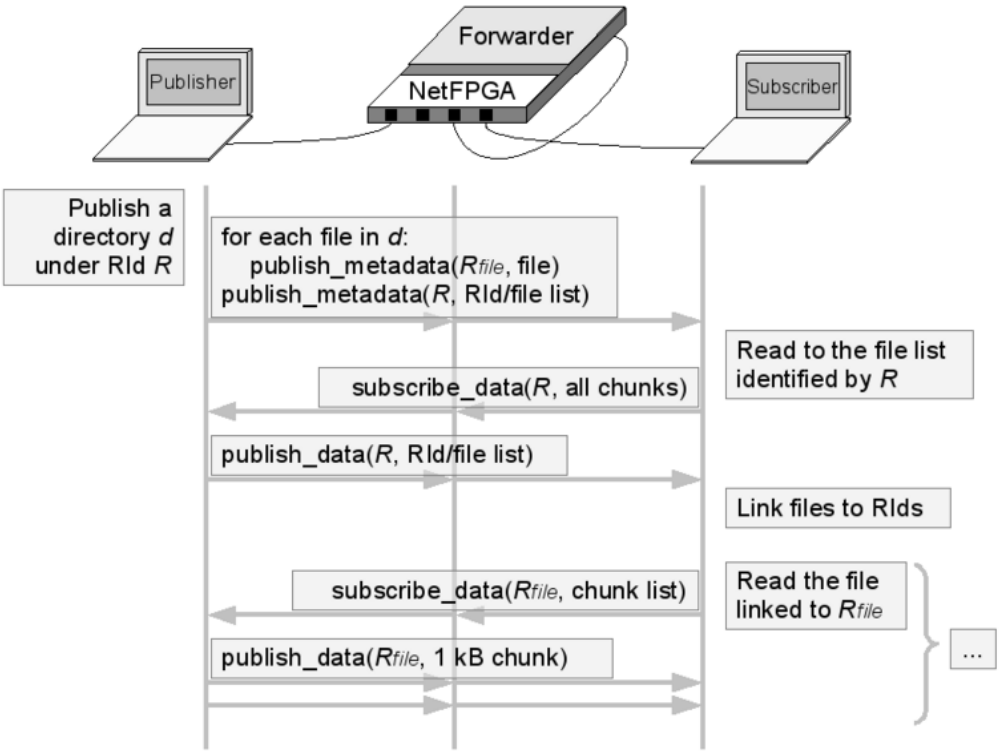

Figure 4.2 – Publishing Directory Contents

By adding this functionality on top of the underlying pub/sub system, the following concrete demo scenario became possible:

1. A computer game's executable binary, e.g. pinball, and its resource file directory (containing sounds, images, etc.) are published and loaded in the publisher's memory. At this point, only the metadata of these files get transferred to the subscriber.
2. The subscriber parses the received list of files and creates links in the same directories where they were found on the publishing computer (e.g., /usr/local/share/pinball).

3. The application is started by executing the link to the binary, found by its original name on the subscribing machine. The contents of the binary, or the needed parts of it, are fetched from the publisher. Some additional data gets transferred as well, as resource files are accessed (via the links that point to /pubsub/.../.../data files).
4. When different levels are loaded in the game, more data is fetched from the publisher.

4.1.2 NetFPGA demonstrator: UDP streaming over zFilter-based multicast

In order to demonstrate the multicast features of zFilters [Jok2009] and the forwarding node implementation for NetFPGA [Kei2009], a stand-alone pub/sub proxy that performs a simple translation of UDP data to PSIRP packets was written. The proxy was implemented in Python for Linux. On the publisher side, it re-packetizes the payload of UDP datagrams into Ethernet frames with a forwarding header that includes a zFilter-FId. Subscribers do the inverse operation to received frames. In this system, subscribers of data need to know a FId with a path to a publisher, who can then OR the collected return path FIds to a FId that is used for data delivery.

The demo network can look like the one in Figure 4.3 below, or it could be much more complex, having multiple forwarding nodes. Each end-node can run several instances of the proxy (with different UDP and FId configurations), and can act as both a publisher and subscriber for several data streams (audio, video, etc.) at the same time.

Of course, this demonstrator was only a temporary solution -- created for the NetFPGA Developers Workshop in Stanford in August 2009 -- before integrating the same kind of functionality into the Blackhawk prototype (see below).

4.1.3 Standalone Rendezvous Network Demonstrator

During spring 2009, the rendezvous node implementation was expanded to support rendezvous network functionality. This functionality was successfully tested in a rendezvous network demonstrator, which is illustrated in the Figure 4.3.

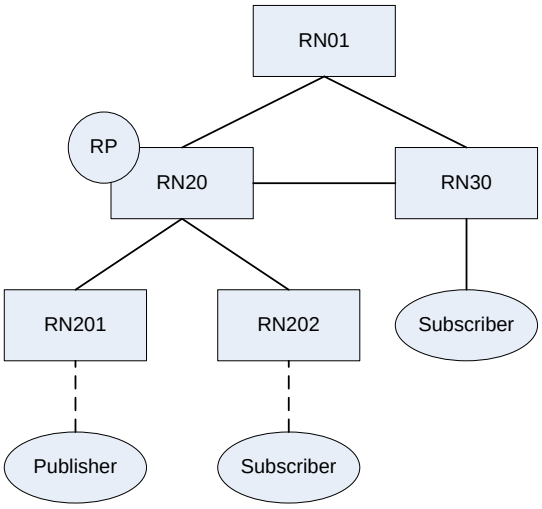

Figure 4.3 – Rendezvous Network Demonstrator Example Topology

In the demonstrator, a group of rendezvous nodes (RNs) establishes a rendezvous network with a policy compliant topology by using a simple rendezvous node discovery protocol. When the rendezvous network topology is established and each RN knows their active RN neighbourhood, test clients (publishers and subscribers) could start communicating with it through publications and subscriptions. The rendezvous network provides rendezvous and publication reachability. In addition, before the clients can start subscribing and publishing publications, they need to determine their policy-compliant inter-RN upgraphs by subscribing to the well-known “upgraph probe” publication. This probe is forwarded upwards in the

rendezvous network based on the topology and, while moving upwards and horizontally, the upgraph is updated hop-by-hop in each rendezvous node. Finally, the completed parts of the upgraph are returned to the test client as a publication. The reasons for having these upgraphs, in the first place is that publisher and subscriber upgraphs are needed to be submitted from the RP to the *topology stub* in the case of a publication rendezvous. From the upgraphs, based on given policies, the topology stub can build a policy compliant forwarding path for the publication data delivery – this path being returned to the RP, which in turn delivers it to the publisher. When receiving the notification from the RP, the publisher sends the publication data to the subscriber using the provided path information. It is important to note that this demonstrator does not implement multicast forwarding. Instead, data is delivered separately to multiple subscribers.

The demonstrator runs in a single physical node where the rendezvous network is locally established as a logical network. Communication between the rendezvous entities is implemented using UDP. The demonstrator supports multiple test clients in single or dual role. Figure 4.3 shows a typical example topology of the rendezvous network. RP creation is done dynamically based on the trust relation between the test client and its rendezvous node.

4.1.4 Planned demonstrations with Blackhawk v0.1

At the time of writing, no public demonstrations have yet been held with the released Blackhawk prototype. The architecture of the released version has changed considerably from the FUSE-based prototype, as explained in D3.3 [Jok2009a]. In fact, the same demonstrations as described above cannot be performed with the current version, because it does not support demand paging, and because metadata is sent only to a rendezvous node instead of "pushing" it to all nodes in the network. Furthermore, the prototype has limitations regarding the publication size, number of publication versions, etc., and it does not support multicast data delivery.

The current setup used for testing the prototype is depicted in Figure 4.4.

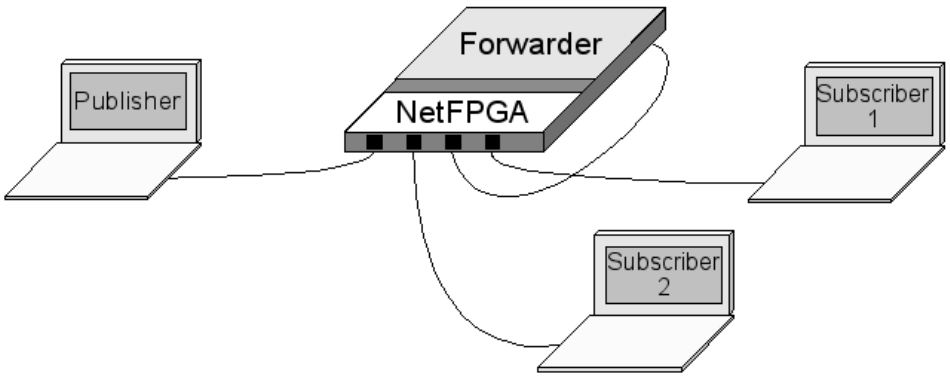

Figure 4.4 – Simple Test Setup for the Current Prototype

Instead of the NetFPGA-forwarder, a FreeBSD node can be used to perform the same role. In this system, metadata about publications is sent to and stored at the local rendezvous node. Applications using *libpsirp* can issue *asynchronous* subscriptions to data, that is, they can receive whole publications by first getting metadata from the rendezvous node, and then data from the publisher.

Once the required functionality has been implemented in future releases, similar demonstrations as with the FUSE-based prototype and the UDP/zFilter-proxy could become possible. In addition, the demonstration could present, e.g. network attachment, rendezvous, topology management, transport, and caching functionalities once these get fully integrated into the implementation. Using pure pub/sub-based applications as demonstrators has also been envisioned.

4.2 Planned Demonstrator at Essex University

Efforts have been undertaken to establish a campus-scale infrastructure to demonstrate and test the integration of various components, to demonstrate applications, and to provide an infrastructure for external partner collaboration. The infrastructure is supposed to be gradually expanded for our purposes. The following section outlines this infrastructure and the current status of its realization.

4.2.1 Infrastructure

The infrastructure is based on a heterogeneous IP infrastructure that was built under the recently finished UK TSB (Technology Strategy Board) funded project HIPnet [HIP2009]. It provides a variety of access technologies in the wireless and wireline domain, e.g., WiMax, WiFi, and all-optical infrastructure. The infrastructure spans the local campus at Essex University, located at the edge of Colchester (UK). The university's facilities hold about 2500 students in their dorms, with access to their infrastructure. The wireless coverage has recently, in June 2009, been extended to full campus coverage with a single SSID. The WiMax coverage spans most of the campus. The physical connectivity at the optical level extends to the BT facilities at Adastral Park (UK). Figure 4.5 outlines the current facilities.

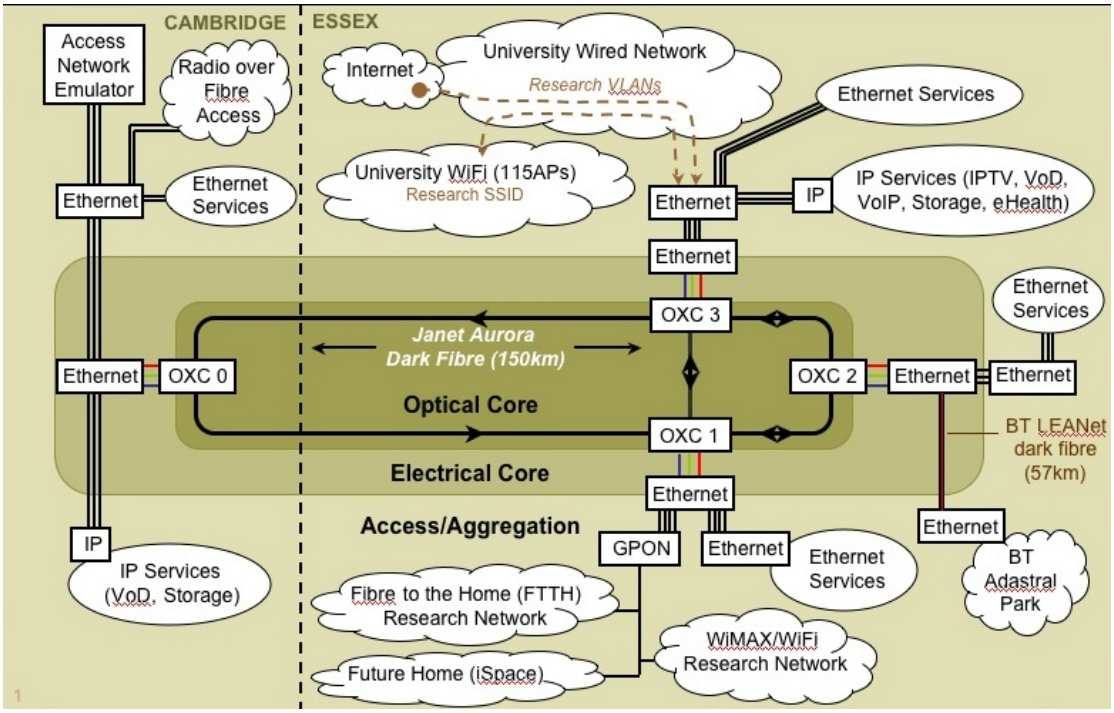

Figure 4.5 – Testbed between BT and Essex University (UK)

As can be seen, various service facilities like storage and video services are available. These could be used for a variety of PSIRP-based services later on.

4.2.2 Current Integration with PSIRP

Currently, three machines are being installed for a simple PSIRP network setup. Two of these machines are a publisher and a subscriber, respectively, currently running the latest release of the PSIRP node architecture BlackHawk [Jok2009b]. The third node is a forwarding node, utilizing the current implementation of the Bloom filter approach developed in PSIRP [Jok2009a].

Work has started to install a third end node, serving as a second subscriber in a simple scenario. Furthermore, a second forwarding node will be installed soon for demonstrating the intra-domain forwarding capability. The forwarding elements will be co-located with the

Ethernet switches at Essex University (see Figure 4.5). The publisher will reside at Essex University together with one subscriber. The second subscriber will reside in the BT premises at Adastral Park. We expect the test bed and its PSIRP nodes to be gradually expanded towards a set of some 10 nodes within a growing campus infrastructure.

4.2.3 Potential Usage

Apart from the apparent demonstration appeal of such a networked setup, serving our dissemination and demonstration to potentially interested stakeholders, the test bed will also serve evaluation purposes as outlined in [Rii2009]. For instance, real network load performance experiments can be conducted for evaluating (a) end node architecture performance and (b) forwarding node performance.

But we also plan to utilize the test bed for extensions at the technological level. One such extension is the development of a topology management module (see [Tro2009]), which will demonstrate the applicability of the PSIRP information concepts for optimizing resource utilization on the optical level. For this, we will utilize the existing optical infrastructure in the testbed. This work will be conducted in partnership with Essex University and is a very good example of the engagement with external partners.

Furthermore, the integrative demonstrator will be used as the basis for a UK-funded project between BT, Ericsson (UK), Essex University and Cambridge University in the area of lifestyle management [PAL2009]. This project targets novel services in the user-centric health area through self-monitoring and information processing. This is an area where we expect novel input from an underlying information-centric architecture like PSIRP.

5 Conclusions

The integration of implementation work into a single coherent prototype is progressing, as we outlined in this deliverable. As a result, not only is the architecture work converging, but also the various implementation efforts on these architectural components are starting to converge into a coherent and running system. In particular core components like the node architecture, forwarding and rendezvous node are coming together, allowing for envisioning a first networked setup of a PSIRP network. Further integration efforts are needed for components like topology management, node attachment or the ITF node. In the latter cases, the existing collaboration with the architecture efforts is the foundation to provide direct input into the design of node implementations.

We recognize that the demonstration of the capabilities of our system is crucial. For this reason, a first demonstrator in our laboratories existed relatively early in the project and is constantly being updated, based on the implementation work. In addition, efforts are underway to extend the laboratory demonstrator towards a fully networked demonstrator within a local test bed. This will not only enable demonstrations but also provide a testing ground for the implementation itself.

6 References

[HIP2009] "Heterogeneous IP networks: HIPnet", available at http://gow.epsrc.ac.uk/ViewGrant.aspx?GrantRef=EP/E002382/1, August 2009

[Jok2008] P. Jokela (ed), "Implementation Plan based on Concept Architecture", PSIRP deliverable D3.2, September 2008

[Jok2009a] P. Jokela, A. Zahemszky, S. Arianfar, P. Nikander, C. Esteve, "LIPSIN: Line speed Publish/Subscribe Inter-Networking", Proc. of ACM SIGCOMM, 2009

[Jok2009b] P. Jokela (ed), "Progress Report and Evaluation of Implemented Upper and Lower Layer Function", PSIRP deliverable D3.3, June 2009

[Kei2009] J. Keinänen, P. Jokela, K. Slavov, "Implementing zFilter based forwarding node on a NetFPGA", Proc. of NetFPGA Developers Workshop, August 2009

[PAL2009] "PAL: Personal and Social Communication Services for Health and Lifestyle Monitoring", available at http://pal.dalore.net, August 2009

[Tro2009] D. Trossen (ed), "Architecture Definition, Components Descriptions and Requirements", PSIRP deliverable D2.3, February 2009

[Rii2009] J. Riihijarvi (ed.), "Description of validation and simulation tools in PSIRP context", PSIRP deliverable D4.4, July 2009