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

DELIVERABLE D3.1

Prototype Platform and Applications Plan and Definition

Title of Contract	Publish-Subscribe Internet Routing Paradigm
Acronym	PSIRP
Contract Number	FP7-INFSO-ICT 216173
Start date of the project	1.1.2008
Duration	30 months, until 30.6.2010
Document Title:	Prototype Platform and Applications Plan and Definition
Date of preparation	29.4.2008
Author(s)	Petri Jokela (LMF), Teemu Rinta-aho (LMF), Dirk Trossen (BT), George Xylomenos (AUEB-RC), Janne Tuononen (NSN)
Responsible of the deliverable	Petri Jokela Phone: +358442992413 Fax: Email: petri.jokela@ericsson.com
Reviewed by	Dirk Trossen (BT), Janne Riihijärvi (RWTH Aachen)
Target Dissemination Level:	Public
Status of the Document:	Completed
Version	1.0
Document location	http://www.psirp.org/Deliverables/D3.1
Project web site	http://www.psirp.org/

Table of Contents

Executive summary

The PSIRP prototyping work package is responsible for generating prototypes of the PSIRP architecture designed in the WP2. During the project this WP will create few iterations of the prototype, providing feedback on the architecture, as well as providing demonstrations for illustrating the publish/subscribe network operations.

In this deliverable, the initially three-layer structure and the interfaces between the layers are briefly described. The lower and upper layer implementations (LoLI and UpLI, respectively) are described in more details in the deliverable 3.2, and this deliverable 3.1 gives more detailed information about the application development process as well as the platform and tools that will be used during the project. In addition, the licensing terms that have been approved in the project are briefly described and guidance for software developers is given.

1 Introduction

The current Internet has properties that are nowadays considered problems. One of the problems is the sender-centric way of thinking: the network serves the sender of data and delivers the data always to the destination that the sender has specified. This doesn't give the receiver any control and it has led to problems e.g. in the form of Denial of Service attacks. The publish/subscribe paradigm (pub/sub) has been proposed as a solution to such problems in the Internet. In the PSIRP project the Internet architecture will be redesigned from the publish/subscribe point of view.

Designing a new architecture requires also active work on testing the proposed architectural decisions, giving feedback from tests to the architecture designers, and redesigning the architecture if there are problems when running in a real environment. In PSIRP, the WP3 has the responsibility of creating prototype based on the pub/sub architecture created in WP2, architecture work package.

The system has been initially divided into three layers; lower layer, upper layer and applications. One of the purposes of this document is to give a more high level description of these three different areas and interfaces between them. In addition, the application work is described in more detail. The lower layer implementation (LoLI) and upper layer implementation (UpLI) will be described in more detail in the deliverable 3.2 and in this document only a rough description of their functionality is given. The application development activity includes an innovation process that gives guidelines to initiate and perform co-operation with external partners, e.g. universities to develop applications on top of the implemented UpLI and LoLI.

In addition, an essential topic in this deliverable is the description of the tools used for prototype development. This document specifies the operating systems that will be used in the project as well as the required tools for prototype development and version management.

In the project, a licensing scheme has been agreed on and this document gives guidelines as to how the prototype developers need to take this into account when writing implementations and what needs to be considered when the prototype code is distributed between partners or outside the project.

1.1 Abbreviations

LoLI	Lower Layer Implementation	UDP	User Datagram Protocol
PSIRP	Publish Subscribe Internet Routing Paradigm	UpLI	Upper Layer Implementation
TCP	Transmission Control Protocol	WiFi	Wireless Local Area Network Certified by the WiFi Alliance

2 Environment and Tools

In this section the prototype code development tools and management system are described. Also the licensing terms that have been agreed in the project as well as how they are applied to the code are introduced.

2.1 Code Management

2.1.1 Licensing

The project has decided that a dual-licensing model is used in the project. All code shared between partners or distributed externally under the same license. The license text that must be included in every code file written in the project can be found from the project web pages.

The license text that is included in all code written in the project states that the user of the code can select either BSD or GPLv2 license, which ever suits best the user needs.

As an exception, if pieces of code that are originally under other licenses are used in the project, they have to be clearly marked and listed in the project web pages in the relevant section. The distribution of such code needs to follow the conditions of the original license. For example, if Linux kernel code is modified to support pub/sub code, the original license in Linux mandates the usage of GPL for the pub/sub code.

2.1.2 Code Availability

An internal approval process will decide when code can be published outside the project. The published code needs to be stable (enough). Before publishing, there has to be a review period during which the code will be checked to avoid publishing anything that is not meant for publishing. Published versions have to be marked with *reviewed* and *ready for publication* tags.

The development repository is not directly visible outside the project. However, it may be made available during the latter stages of the project, when the architecture and the implementation are somewhat mature.

For both internal and external use, it is desirable that the software is also made available as easily installable binary packages (like Debian .deb or FreeBSD .tgz) so that people who want to try the software will not need to compile it. For internal use, these should be made available early in the project, e.g. as daily builds.

When the project decides to publish a public version of the code, each partner has to have at least one week time to review the published code before the publishing date. When a new version of the code is made available, it will be announced on the project web pages together with required instructions on how to download and install the code.

2.1.3 Code Repository and Version Management

The code repository is maintained at HIIT and software version management is handled using Subversion. Detailed information about the repository as well as rules of usage will be available at the project web pages.

The code repository is accessible by the project members. In case of collaboration (Section 3.2), full or limited access can be granted to external partners developing software on top of PSIRP platform.

2.1.4 Community Process

Involvement of the external developer community is desired and one of the main reasons to open source the software developed in this project. In order to harvest this collective intelligence, community processes, such as bug reports, forums, wish lists, and future

requirements, will be established on the public website. This allows for an active dialogue with the external developer community.

2.2 Operating System

While the target of the project is to design an architecture that will be possible to implement on any computing platform, we have to note that it is not feasible during a research project to implement the prototypes on every operating system, especially considering that the lower layers of the implementation will have to reside close to the hardware, at least for performance reasons.

However, one limiting factor that has to be taken into account when selecting tools and environments for development is that in the project the portability of the code to other operating systems has to be maintained to the extent possible. The project will develop a single demonstrator where all different parts of the implementation are merged. The development has been divided into three main parts: applications, upper layer implementation and lower layer implementation (see Section 3).

For the lower layer implementation (Section 3.5) the FreeBSD 7.x operating system has been chosen as the main implementation platform. The decision is largely based on technical aspects: the BSD networking code is well documented, and this is especially important considering the fact that parts of the code will have to be implemented in the kernel for performance reasons. From the dissemination point of view, BSD is used also in other networking research communities. In a similar way as the innovation process for applications is defined in Section 3.2, future development in different operating systems is also possible through external partners. See Table 1 for a comparison between different operating systems.

The upper layer implementation (Section 3.4) as well as application development (Section 3.3) will use an operating system neutral platform for development so as to enable easy portability, even though the FreeBSD 7.x platform will be the only environment where the lower layer implementation will initially be available. Implemented parts need to maintain the portability to FreeBSD 7.x platform so that the different implementations can be merged into one single demonstrator.

While these software parts are implemented in user space, porting them to FreeBSD should be straightforward as long as they do not use features that are available on some OSes only. These kinds of features have to be avoided.

The interface between the lower and upper layers will have to be designed to provide low coupling between the layers, allowing the independent evolution of each layer and preventing the need for considerable integration or porting efforts when both layers are to be used together at the same node.

The programming language that will be used in the lower layer implementation is C, the language of choice for the kernel of most operating systems. The upper layer implementation and applications may use other languages, with preference to portable languages such as Java or Python.

It is desirable that the source code development environment is setup so that it supports implementations on multiple platforms already from the beginning. This means that tools such as *autoconf* and *automake* will be used for configuring the code so that the user will not have to manually configure the distribution for compilation in different environments.

Table 1 OS Comparison

OS	Kernel level development	Used in research	Cost	Users	Licensing
Linux	Yes	widely	Free	Mostly skilled people	GPL
***BSD (FreeBSD)**	Yes	widely	Free	Mostly skilled people	BSD
Windows	No	Not widely in network protocol research	Not free	Widely used	Proprietary
MAC	Yes	not widely	Not free	Professional / Business people, key persons in network research area	BSD

3 Prototype Plan

In this section the main parts of the implementation are described. First, we introduce the initial separation to different implementation modules, and after that describe the application development process. Finally, the initial plans for the upper and lower layer implementations are described.

3.1 Overview of Implementation Modules

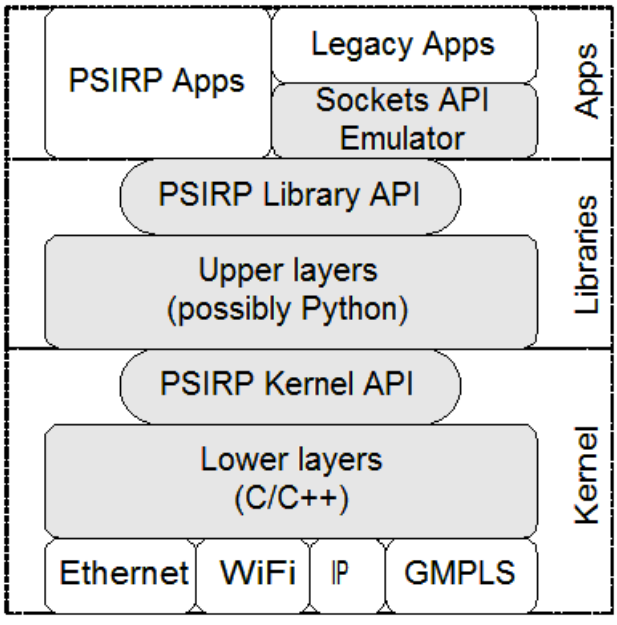

Figure 1 Implementation modules and APIs

The PSIRP prototype will be implemented in a modular manner, the outline of the foreseen structure being given in Figure 1**.** The core PSIRP prototype will be implemented in two layers:

1. The lower layer will mainly, if not exclusively, reside in kernel space for performance reasons, providing whatever functionality is deemed to be critical enough to justify its inclusion into the kernel. Examples of such functionality include interfacing with

network devices, data forwarding and data manipulation for forwarding purposes. The lower layer will be the hardest part to port since it may be quite kernel specific, especially if it is fine tuned for performance. Nevertheless, every effort will be made to write as portable code as possible. Furthermore, the lower layer will be the only common part in all nodes participating in a PSIRP based future Internet.

2. The upper layer(s) will reside exclusively in user space for flexibility reasons, providing whatever functionality is deemed necessary to support writing PSIRP applications, supplementing the functionality provided by the lower layer. Examples of such functionality include providing primitives for publishing data and subscribing to publications. The upper layer(s) should be portable to any open source operating system providing an implementation of the lower layer. It will be possible for a PSIRP network to support multiple types or multiple instances of the upper layer functionality. For example, a single PSIRP node may support both reliable and unreliable data publication via two different upper layer implementations.

3.2 APIs

The PSIRP prototype will incorporate implementations of two interfaces:

1. The lower layer will implement the PSIRP kernel API for the benefit of the upper layer(s). While the exact functionality provided by the PSIRP kernel API will depend on the actual division of work between the upper and lower PSIRP layers, it is expected that this API will be implemented as a set of new system calls, wrapped in a set of thin library functions, for example publish() and subscribe() instead of write() and read(). It is also expected that the PSIRP kernel API will be able to support multiple upper layers in the same node.

2. The upper layer(s) will implement the PSIRP library API for the benefit of the applications. While the exact functionality provided by the PSIRP library API will depend on the actual functionality offered by PSIRP as a whole, it is expected that this API will be implemented as a set of new library calls or extensions to existing C library calls. It is possible that different instances of the upper layer(s) may support different parts of the PSIRP library API, for example, reliable and unreliable data publication.

The lower layer of the PSIRP prototype will run directly on top of the network device drivers, for example the Ethernet and Wi-Fi drivers. In order to be compatible with the existing IP based Internet, the lower layer will also run on top of IP by treating IP as just another link layer technology, albeit with more capabilities than other link layers. While it is certainly possible to exploit IP/UDP/TCP in order to allow the PSIRP upper layer(s) to run as an Internet overlay, this would require the upper layer(s) to interface with two very different underlying layers (the PSIRP lower layer implementation and IP/UDP/TCP) something that would not only complicate upper layer design but would also create a tension between the two interfaces in areas such as optimization. By treating IP as just another link layer technology, the lower layer implementation will make fewer demands on it (similar to those made on Ethernet and Wi-Fi, for example, only local node reachability will be assumed) making it possible to completely avoid using UDP and TCP. This will simplify migration by first making UDP and TCP redundant, and then allowing the elimination of IP itself. Migration is further simplified by treating IP as a link layer technology, as this allows IP to be gradually replaced by PSIRP in individual local networks: when all hosts in a local network have converted to PSIRP, support for IP can be simply dropped.

In order to simplify migration from the current Internet to a PSIRP based future Internet, the PSIRP prototype will incorporate at least a Sockets API emulator running on top of the PSIRP library API, thus allowing legacy applications, such as web browsers, to operate on top of the PSIRP upper layer(s). It will also be possible to provide custom higher level emulators for legacy applications so as to improve their performance. For example, web browsing can be supported either via the Sockets API emulator which will treat the HTTP requests and replies

simply as a set of TCP sessions, or via a custom and more efficient HTTP emulator that will translate HTTP requests and replies directly to PSIRP library API calls.

3.3 Applications

Applications are necessary to showcase the functionality enabled by the new paradigm and the technologies developed in PSIRP. These applications will give a glimpse on the possibilities, some of which are outlined in our vision and use cases for a PSIRP-enabled world [2].

However, application development is costly in time and resources. It can easily consume a substantial (even the major) part of the development effort, driven by issues such as UI development, user tests, stability tests, support, and documentation, which are well beyond the scope of PSIRP. We will first outline in the following a process to attract external partners to our environment, before presenting in more detail the directions in which we intend to steer some of these activities.

3.3.1 Innovation Process and External Partners

PSIRP does not have the resources necessary to showcase the wide applicability of what the architecture and its paradigm shift potentially can bring. As a result, we intend to establish an innovation process in the application space that will allow for attracting the applications we need in order to showcase what we developed. The efforts within PSIRP will then be limited to facilitate and enable this innovation process, directly work with partners, feed potential requirements into the project and so on.

Hence: the innovation process intends to ensure that external partners will innovate on top of the PSIRP stack to build applications that showcase the possibilities outlined in our vision.

For this, the following steps are taken:

- Select a group of external partners with which PSIRP will directly collaborate in application development and development support.
- Collaborate with external initiatives and projects, such as the FIRE (Future Internet Research) initiative or the Onelab2 project, to facilitate experimentally-driven research on top of PSIRP technology.
- Develop a set of applications with these external partners that will demonstrate the appeal of the PSIRP platform.
- Hold frequent workshops with external partners to facilitate application development and disseminate the implementation results.
- Establish a community development process with bug reports, requirements input, wish list and so on, that is directly fed into the implementation efforts for the upper and lower layer implementations.

It is worth mentioning that a similar innovation process will be applied for the adaptation of the PSIRP lower layer implementation onto a variety of link layer technologies, as this is an equally resource intensive task that cannot be completed within the lifetime of the project.

3.3.2 Driving the Innovation Process

The previous section outlines the general process used to stimulate innovation within the PSIRP environment. Generally, this process is intended to create a community of developers for applications but also lower network adaptations, which will enrich the PSIRP environment beyond the core partners and therefore showcase the technology we will develop.

While this general process is not intended to restrict but rather foster the innovativeness of the involved external partners, we do intend to maintain specific relations that are steered towards developing certain solutions, in particular in the application space that will emphasize the

value of the solutions developed in PSIRP. With this more steered form of collaboration, we intend to create a dedicated pool of applications for our environment that best demonstrates its value.

For this, we intend to closely work with a selected set of partners on applications of special interest. It must be clear however that we cannot guarantee these applications as firm deliverables, being in line with project work plan. But we can commit to use our limited resources to prioritize certain communal development towards a set of most interesting use cases from a project perspective.

These special interest applications are envisioned to be located in the following areas:

- New forms of file sharing: Natural forms of information to be shared within PSIRP are files as known from today's computer systems. A demonstrator in this space could showcase the possibilities stemming from the possibility to distribute files over a large set of providers, similar to the known BitTorrent and other systems. Such system will also help to investigate new forms of DRM in sharing such information.
- New forms of participatory sensing: wireless sensing is a potential for future PSIRP systems (see also the vision and use cases [2]). Hence, special interest is placed on new forms of participatory sensing systems, potentially based on mobile device platforms that will take advantage of unique features of our PSIRP system. For instance, the available open source NORS (Nokia Remote Sensing) platform [3] can be used as a basis for such development.
- New forms of media consumption: A PSIRP system seems to be a natural candidate for an evolution of media consumption towards a highly distributed, multipoint-supporting form of media system. A demonstrator in this area is expected to show the strengths of a PSIRP platform in this area.
- New forms of context awareness: the information-rich scenarios for context-aware applications seem to be a natural fit for an information-centric system like PSIRP. Also, the concepts of scoping information according to context ownership map well onto concepts of information scopes in PSIRP. Hence, a potential demonstrator in this space shall shed some light on the potential that such alignment of higher layer information concepts with lower layer information-centric routing could bring for future applications in this space.

It is important to note that the innovation will happen within the realm of the expected external partners. The efforts in PSIRP will be steered to generally enable this innovation. But we will also work closely with the partners in the above outlined areas to define system requirements for the demonstrators, provide substantial support and also coordinate dissemination efforts, i.e., showcase the value of PSIRP with the help of these demonstrators.

3.4 Upper Layer Implementation (UpLI)

The definition of the Upper Layer Implementation (UpLI) is highly dependent on the contributions of the architecture work in WP2. The first of these architecture contributions, the conceptual architecture, is scheduled to be available in M6. Therefore, in the present deliverable only a very rough vision for the upper layer implementation is presented and the actual first iteration of the UpLI description will be part of the deliverable 3.2, which is scheduled to become available in M9.

As shown in the layer model in Section 3.1 the current understanding is that UpLI is the middle layer between the applications and kernel part of the PSIRP prototype implementation, also called the Lower Layer Implementation (LoLI). UpLI is assumed to be implemented as a collection of libraries, where the main implementation entity is the PSIRP library API towards the application layer. The API is supposed to provide pub/sub for both PSIRP aware and legacy applications. Another interface related to the UpLI will be the PSIRP kernel API, which

UpLI uses to communicate with the LoLI. In addition to the APIs, UpLI will also implement at least parts of the rendezvous mechanism, which is actually likely to be a complex set of different rendezvous mechanisms, responsible for locating and providing the contents in the pub/sub system. UpLI is also the natural location for the name resolution function, if such is defined in the architecture. Whether the UpLI will encompass some other functionality is for further study.

The UpLI will be implemented in user space. Because porting the libraries from a Unix system to another is supposedly an easy task, the implementation team has not set any strict rules for the preferred implementation platform. Important issues to be taken into account is that the implementation must be easily portable to the FreeBSD 7.X platform, due to the possible future integrated prototype, which naturally suggests avoiding operating system specific coding practices in the UpLI. The implementation work is planned to have several iteration rounds tightly connected to the architecture work. Therefore the coding should be done with programming languages that enable fast prototype cycles. Typically these programming languages are higher level languages than C or C++ and the implementation team suggests use of Python or Ruby. However, if it is found feasible and/or appropriate, implementers of the UpLI may use also other languages.

3.5 Lower Layer Implementation (LoLI)

As is the case for the Upper Layer Implementation, the Lower Layer Implementation is dependent on the architecture work; therefore a more detailed description will be given in deliverable 3.2.

As shown in section 3.1, the LoLI will work both directly on top of different network technologies, such as Ethernet and WiFi, as well as on top of IP. The support for IP networks is important especially for the transition phase when different types of networks will simultaneously operate over the same physical network.

The functions that the LoLI is implementing are roughly network attachment, forwarding and parts of routing and rendezvous. The forwarding function is responsible for efficient data forwarding from one interface to another, from the interface to the local host (UpLI or other LoLI function), or from the local host to the interface. The forwarding table is set up by the routing function. The routing function is also responsible for verifying the link status.

The implementation should support both passive and active modes, i.e. the participant attaching to a network (or to another node) may either listen to network or it may send a specific solicitation message. The details of the process will be refined during the architecture design process and they will be described in more details in future deliverables.

The LoLI will be implemented partly in the kernel, when high performance is needed, and partly as user space daemons that take care of actions that are not time critical. In LoLI, the target is to avoid copying of data between different memory locations to avoid unnecessary delays in data processing. The algorithms, protocols and the software architecture should be designed so that at least the minimum LoLI (and required parts of UpLI) functionality can in the future be implemented directly in hardware, such as ASIC chips, to enable deployment in fast routers as well as e.g. wireless sensor networks.

The assumption related to the hardware to be used while implementing LoLI is that standard off-the-shelf hardware is used. Due to the reasons specified in Section 2.2 the FreeBSD 7.X operating system was chosen as the platform for LoLI, thus the hardware must be supported in FreeBSD 7.X. This is, however, not a significant limitation while most of the common hardware is supported in FreeBSD 7.X.

Similar to the applications (see Section 3.3); the limited resources of PSIRP will not allow adapting the lower layer implementation to all varieties of available link layer technologies. During the project ethernet, Wi-Fi, and IP will be used as the underlying technologies. For other technologies, we envision a similar innovation process for extending the PSIRP lower layers, that is, cooperation with selected external partners on adapting the PSIRP lower layer implementation for different link layers, thus widening the applicability of the PSIRP stack.

4 Conclusions

This document specified the technology environment for the prototyping work in the PSIRP project. The licensing terms and their usage in the project were discussed. The separation of the actual publish/subscribe prototype into two different parts was described creating a separation between functions that are closely hardware and operating system related and functions that are more generic in terms of the underlying physical and OS technology. In addition, used tools for creating a development environment for the actual source code were specified.

For the applications that are running on top of the actual publish/subscribe implementation, an innovation process was presented. The target for the innovation process is to get a wider variety of applications from external partners that utilize the features of publish/subscribe networking. In addition, a similar innovation process can be adopted for the actual publish/subscribe implementation parts to get support for other operating systems that those used inside the project.

5 References

[1] http://wiki.psirp.org

[2] PSIRP Vision Document (http://www.psirp.org)

[3] Nokia Remote Sensing (NORS, http://sourceforge.com/projects/nors)