International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 5(2013) pp. 050–059
© MIR Labs, www.mirlabs.net/ijcisim/index.html

Towards the Evolution of Publish/Subscribe Internetworking Mechanisms with PSIRP

Augusto Morales Dominguez[1], **Oscar Novo**[2], **Walter Wong**[3] **and Ramon Alcarria**[1]

[1]Technical University of Madrid
Madrid, Spain
amorales@dit.upm.es
ralcarria@dit.upm.es

[2]Ericsson Research
Jorvas, Finland
oscar.novo@ericsson.com

[3]University of Campinas
Campinas, Brazil
wong@dca.fee.unicamp.br

Abstract:
Despite its enormous success, the current Internet's architecture has not evolved in a scalable manner. There are many issues concerning simple services that need complex network systems in order to work. Future Internet Architectures are candidates to solve some of the problems related to the current Internet. This paper describes and evaluates a mechanism to adapt and migrate the current Internet services to future network architectures based on the Publish/Subscribe paradigm. This paper also contributes to a soft transition mechanism for seamless interoperability of the PubSubHubbub protocol to a Future Internet Architecture.

Keywords: Publish/Subscribe, PuSH, Future Internet, PubSubHubbub, internetworking, PSIRP

I. Introduction

The original idea of Internet was based on enabling communication between two machines following the *end-to-end principle* [15]. This functionality did not require many complex communication layers, as long as it could transport data from the sender to the receiver. However, the Internet's evolution has turned it into a mixture of different protocols, models and architectures that brings an unnecessary complexity [6] in the way nodes access to information. Taking this into consideration, there are several research topics that propose a new information-centric approach [7, 16] where Internet is used to transfer information .

These research topics are focused on the concept that standard users or information consumers do not have to be concerned about the source of information, as long as they can receive their data. In this case the network itself has not only to provide the information, but also a mechanism to locate it. This is a different approach from the current host-centric network which Internet is based on.

Nonetheless, this information-centric approach [7, 16] has risen as an opportunity to improve the architectural limitations of current networks. One of the trends in information-centric networks is to base the communication, between network elements that are interested in information, on the Publish/Subscribe paradigm [5]. This approach replaces the traditional end-to-end communication paradigm and its limitations concerning routing, privacy, security and mobility. In this paradigm, there is an important decoupling between location and information. In addition to that, publishers and subscribers do not need to be synchronized, which improves the overall network by simplifying data access mechanisms, and therefore, saving network resources.

The design of information-centric networks [1, 3] should take into consideration several requirements, including addressability, compatibility, and interoperability with existing systems as well as compatibility with resource locators. These requirements are directly related to new applications that will be supported and use this new communication paradigm. However, one of the most important factors is the compatibility with existing Web technologies and Internet. The importance of Web technologies is based on the fact that it has been the core of the current communication models and one of the reasons for the Internet's success. Consequently, these requirements alter the multitude of new applications that will be supported and use this new communication paradigm.

This paper proposes and tests an alternative for integrating information-centric networks with existing host centric networks, using Publish/Subscribe mechanism as the key concept. There are many protocols that can be used for this aim. However, we have focused on the *PubSubHub-*

bub (PuSH) protocol [1], which uses publish and subscribe concepts in an application level and support real-time communication as well. The integration has discovered several issues and opportunities. One of these opportunities is data identification, link, and conversion through different networks. Following sections describe these opportunities in more detail.

The organization of this paper is as follows. Section II describes the background information for our work. Section III presents the design of our host-to-information-centric networking proxy, outlining the main features. Section IV presents the implementation and Section V evaluates our proposed architecture. Finally, Section VI summarizes the paper.

II. Background

In this section, we briefly describe the background information of our work: the PubSubHubbub Protocol and the PSIRP Architecture.

A. The PubSubHubbub Protocol

PubSubHubbub is an open, server-to-server protocol for distributed communication over the Internet using Atom [10] and RSS [2] as a source format. It was created as a mechanism to advance the state of the discussion in the Publish/Subscribe space.

A decentralized Publish/Subscribe layer is missing from the Internet architecture and its existence would enable new features to the Internet, such as decentralized social networking.

In PubSubHubbub (PuSH), all content exchange between publishers and subscribers runs through servers called hubs. The hubs receive subscription requests from subscribers, post updates to subscribers, and provides an endpoint URL to which the publishers can post their feeds and updates.

B. High-Level PuSH Protocol Flow

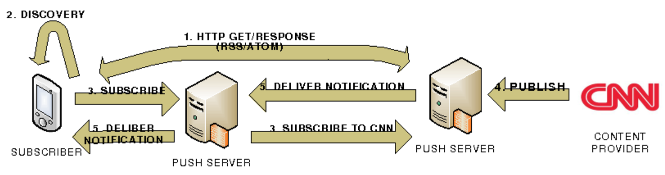


Figure. 1: High-level PuSH Protocol Flow

In Figure 1, a potential subscriber initiates a HTTP transaction requesting (1) the feed to which it wants to subscribe. Once the feed is received, the subscriber discovers (2) the publisher hub URL inside the feed. This information is declared in the Atom [10] or RSS feed document via the $< link\ rel = "hub"\ href = "URL" >$ element. The href attribute contains the publisher hub's endpoint URL.
If the Atom [10] or RSS feed document defines a publisher hub URL, the subscriber registers (3) to the publisher hub URL through its own hub making an HTTP POST request to the hub URL. Additionally, the subscriber subscribes to the topic that he is interested at. A topic identifies the feeds to which the clients can subscribe to changes. Finally, when the publisher publishes (4) new content, the publisher hub fetches the publisher web feed and delivers (5) the new content to all registered subscribers by multicast.

C. The PSIRP Architecture

Research on information centric networks [13] [11] is one of the many alternatives for designing the future Internet Architecture. The *EU FP7 Publish and Subscribe Internet Routing Paradigm* (PSIRP)[3] (and its continuation PURSUIT) project has proposed one of these alternative architectures. The PSIRP architecture [14] proposes a clean slate approach for the Internet, rethinking the basics of the Internet model. The main idea is to substitute the *send/receive* primitives from the current Internet with *publish/susbcribe* down to the stack, i.e., rather than using the IP address to send and receive data, PSIRP proposes to publish data into the network and subscribe the data from the network. Such approach has some advantages, for instance, publishers and subscribers do not need to know about the location of each one (there is not prior knowledge of IP addresses) and subscribers can asynchronously receive data from the network whenever it is ready (thus, clients do not need to poll the servers frequently). Additionally, features such as mobility, security and network-level caching is natively supported in the publish/subscribe model.

PSIRP focuses on information rather than connections, thus, clients focus on content identifiers rather than server's IP address where the content is located. In order to achieve that, each piece of data has two types of identifiers associated, the *rendezvous identifier* (Rid) and *scope identifier* (Sid). The Rid is associated to each publication, where a publication is the smallest piece of data identifiable, e.g., a Web-page, a news feed or even a movie. Note that the identification issue regards to the semantic meaning of the publication and not its size. Therefore, Rids can be used to identify a broad range of content in the network. The Rid is generated using a strong cryptographic hash function over the publication itself, for example, a SHA-1 cryptographic hash function over the content. Thus, the generated identifier is strong and collision resistant and able to be used as unique and permanent content identifier. Any tampering attempt against the publication will result in a different Rid.

The Sid represents one scope in the Internet, resembling a virtual private network in the current Internet. One scope represents an information domain but it is not limited to a location, so it can be related to other kind of logical data, for example, favorite videos and feeds, dislike images, etc. In this way a scope allows to organize the information and set a

[1]PubSubHubbub Project Homepage - http://code.google.com/p/pubsubhubbub/

[2]RSS 2.0 Specification - http://www.rssboard.org/rss-specification

[3]PSIRP Project Homepage - http://www.psirp.org

meaning to it. The Sid is randomly generated using a strong cryptographic hash function, embedding security within the identifiers. This way of information organized via flat and independent labeling is effective since the architecture can provide all its features: routing, security and identification, using the information itself.

The PSIRP architecture is based on the following basic [17] functions: Rendezvous, Topology Formation and Forwarding Fuction, which are depicted in the Figure 2. The Rendezvous function is primarily concerned with associating subscriber and publishers under a specific information scope, so that it can provide a control function for both roles. Subscribers publish their interests into the network and the interest packets are forwarded and stored in local rendezvous. Whenever there is a publication, there is a matching of interests between publishers and subscribers and the content is delivered to the subscribers through Topology Formation and Forwarding Function.

The Topology Formation is responsible for creating forwarding paths from the publishers to the subscribers. Whenever requested by the Rendezvous, the Topology Formation computes a set of paths interconnecting the publisher to all subscribers. Note there that there can be more than one subscribers, through it may require some multicast communication abstraction. The Forwarding function is responsible for forwarding each publication at each hop to the correct destination. Rather than regular IP-based forwarding, PSIRP use Bloom filter-based forwarding to deliver the data [12].

The complete picture works as follows: subscribers subscribe to publications and send a publication containing the interest to the network. The network knows how to deliver to the local rendezvous server through initial configuration (e.g. bootstrapping process) and the interest is placed in the server. Upon receiving a publication matching the interest, it triggers the Topology Formation to find the path between publisher and subscribers and asks for a set of bloom filters for content delivery to the interested subscribers, which is delivered to the publisher. Then, the publisher publishes the data with these identifiers and the network perform a check-and-forward at each hop until the data is delivered to the subscribers.

Whenever publishers require a publication issue, they have to use two identifiers: Rid and Sid. Since Rid and Sid are decoupled from location, these identifiers can be generated using random or hashing functions. On the other hand, when subscribers want to access specific data, they only need to know the Rid and Sid, and the network will take care of routing, access control and data forwarding.

III. Architectural proposal

One of the issues in the current Internet is the asynchronous characteristic of several applications and services, for instance, web feed distribution. Web feed distribution can be done using different techniques. One of the most common techniques is polling. Polling can affect the overall network

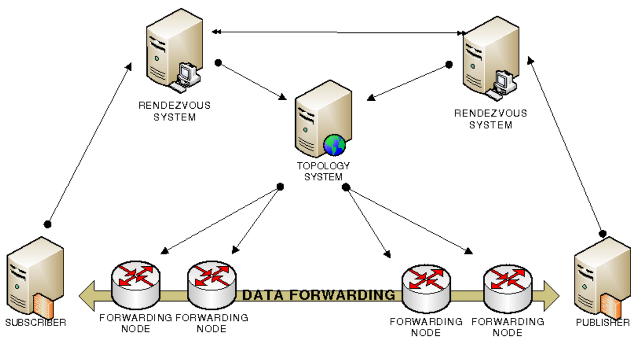


Figure. 2: PSIRP Architecture

performance, particularly; it can be significant when the core network does not have enough capacity for managing traffic. Consequently, there is resource wasting that could be transferred to other applications that need more priority and bandwidth. If we analyze this simplistic content distribution scenario and take it to a larger scale, we can conclude that Internet is capable of carrying out these simple services but it is not being utilized efficiently to its full potential.

There are other approaches that help to improve the performance. However, they also overuse resources through continuous synchronizing. In conclusion, even if a technique tries to improve the overall network using asynchronous communication, the Internet architecture limits the number of opportunities to improve the whole network.

The architecture described in this paper tries to deal with this problem. Based on the Publish/Subscribe paradigm, the proposed architecture describes mechanisms required to bridge the host-centric and information-centric model together. The PSIRP architecture works with an alternative communication scheme which is not compatible with any of the current host centric networks and applications. Therefore, an interoperability mechanism is needed between both architectures. In addition, we have concentrated on the *PubSubHubbub* (PuSH) protocol, as the reference application protocol, because of its similarity with concepts used by information centric network.

A. *Network Architecture Overview*

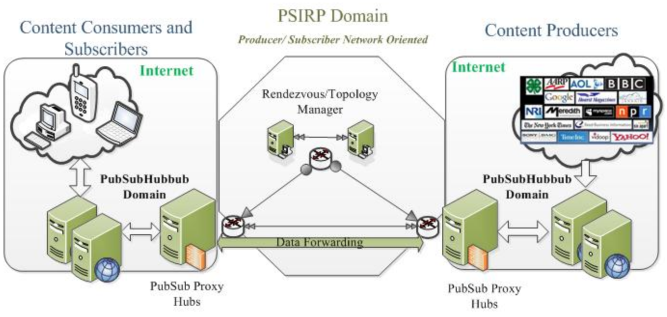


Figure. 3: PSIRP/PuSH architecture

As shown in Figure 3, the architecture accomplishes seamless interoperability between host-centric and information-centric domains. As a first requirement, the host-centric domain has to be agnostic around an existent lower layer where data are transmitted, routed and managed. For this task, the architecture is mainly based on network proxies called *PubSub Proxy Hubs* (PSPH), which carry out all the processes of translating, routing and adapting the information between the different domains. One of the key architecture's challenges is to translate the information, and its inherited attached location, to a native data network. In a host centric network, the data meaning is given by the location, for example, in the PuSH protocol the feed content has the topic URL. So, it shows who is the data owner as well as where it could be found. This concept does not exist in the information-centric network because network elements do not need it. Thus, the PSPHs keep this relationship across different domains in order to transfer data back and forth, without losing the information's meaning. PSPH are in charge of creating new information scopes which contains the content of the feeds. In our proposed architecture the PSPHs are in charge of the protocol communication. In the asynchronous communication case, in the PSIRP side, the architecture relies the message forwarding process through the Rendezvous point existing in the PSIRP. This task is executed using mechanisms provided by the PSIRP architecture [17]. However, in the host centric side, the PSPH requires to alter the end-to-end connection in order to ensure compatibility with applications. So the PSPH manages the TCP sessions at expenses of the performance.

On the border of each host centric domain a PSPH implements the transforming mechanisms for carrying data through TCP/IP and the information-centric network. As the PuSH protocol needs some elements in order to deliver information, there are PuSH hubs which manage the subscription and publishing information related to the parties interested on the feeds. These hubs communicate with the PSPH following the PuSH standard using HTTP messages. A PSPH also integrates some functionalities of the PuSH hub, because it has to process the data flow and filter the information.

Basically, when a client subscribes to a specific URL topic, which represents the information the client wants, this request goes to the PuSH Hubs and then it communicates with the PSPH using the PuSH protocol. The PSPH transforms the requested information (the URL topic) into an information-centric compatible request, previously generating the scope identifiers <Sid,Rid>, and then expressing its interest on this kind of information through the PSIRP network. In the publisher side, when the content producer publishes data, it goes to the PuSH server using PuSH messaging who informs the PSPH regarding this new information. Thus, the PSPH publishes the information in the PSIRP network using scope identifiers, which were generated following a process we will explain later. In order to deliver information from the PSIRP publisher side to the subscriber, both identifiers Sid and Rid must match.

B. Application Programming Interfaces

In the following section, we present the layer topology and APIs used in the different parts of the architecture. It is interesting to note that our proposed architecture does not have any specific method in the terminal side. The communication between the terminal and the PuSH hubs can be done using different communication techniques. That communication is not part of the architecture proposed in this paper.

1) PubSub Proxy Hub

The *PubSub Proxy Hub* (PSPH) follows the topology reference of the figure 4.

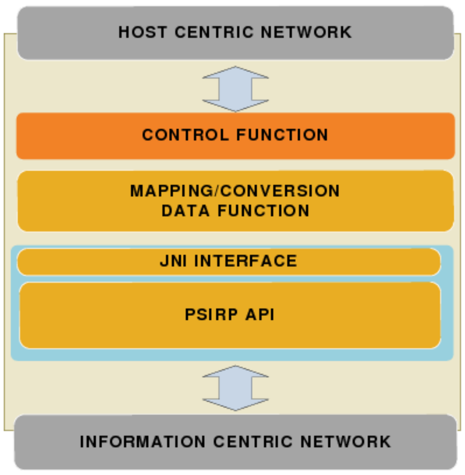


Figure. 4: PuSH/PSIRP Layer Topology

The first layer is called *Control Function* and it is in charge of maintaining the subscription status of every PubSub Proxy Hub. The subscription information is generated using the correspondences between the callback URL, (of the specific Hub) and the URL topic. Therefore, all the subscription statuses are stored using Bloom Filters [2]. This function allows a seamless interaction between the PuSH Hubs and the PSPH without modifying any part of the PubSubHubbub network.

The PSPH has to deal with two different naming systems: the Sid/Rid and the URLs. In this way, it is necessary to guarantee a stable relationship between them in order to keep data consistency, as well as to maintain compatibility. The URL is provided by the hierarchical model of the Internet which prevents a naming duplication; as a result, the PSPH must offer the same functionality in the PSIRP side. Therefore, the *Mapping/Conversion data* function is in charge of creating, manipulating and adapting identifiers. This function also keeps the relationships ordered in a hierarchical manner, which helps to easily look for correspondences. Since the Rid/Sid are 256 bits identifiers, the new PSIRP publications would overlap highly unlikely, so this function allows the PSPH to handle two different kinds of identifiers that point out the same information.

The PSIRP API performs as the bridge with the PSIRP Network, which publishes the information into the PSIRP domain using Sid and Rid. The PSIRP API calls the publish and subscribe primitives that later are received by the Rendezvous nodes. Considering that the native Publish/Subscribe API provided by the Blackadder[4] is programmed in C, and the other Proxy's functions are Java based, we employed an intermediate *Java Native Interface* (JNI). This interface fulfills the bi-directional calling requirements for the PSPH implementation, and abstracts other internal identifiers (e.g. LIPSIN). Hence, PSPHs are designed to perform as standard PSIRP nodes, but could also instantiate the Rendezvous and Topology Manager functions without affecting their core function.

2) PSIRP Network API

The PSIRP API makes use of the Blackadder v0.2.0 as the PSIRP network platform. The current Blackadder implementation offers two Java APIs: a SWIG-based and a Java JNI library. We integrated the JNI library which - at expenses of simplicity - offers better performance.

The first version of our prototype [4] used the Blackhawk v0.3.0[5] implementation instead which, regardless of the version number, is an older API version and less efficient.

C. Designed Architecture Features

There are different data mechanisms in the PubSubHubbub and PSIRP domains. The PubSubHubbub domain uses standard URL in order to define where information is located, as well as its meaning. This URL, called topic, identifies the feeds to which the standard clients can subscribe to changes. In this way, in the host centric domain, the client has only to know the URL in order to obtain the information a publisher is delivering using standard TCP/IP communication. On the other hand, in the PSIRP network there are not TCP/IP packets. The information is distributed using publications identified with a <Sid,Rid>pair. When a publisher requests to publish a new piece of information, it gathers up a Rid and pushes it to the system. In the same way, when a subscriber wants to get this piece of information, it acquires the Rid and asks the Rendezvous system to arrange the data to be received. Once the Rendezvous system has identified a publication that has a publisher and subscribers, the network requests the topology system to build a forwarding tree from the current data location to the subscribers. In this context, the architecture has mechanisms for ensuring compatibility between Sid/Rid and URL through the PSPH, which is in charge of keeping relationships between the two different naming systems involved.

1) Application Extensibility

The architecture allows flexibility in the case where other applications need to communicate through the PSIRP network. The Control Function is completely detached from the PuSH implementation, as well as the scope identifiers (Sid and Rid) are not limited by the information's location. Therefore, in order to support a different protocol the PSPH needs to implement it, by adding an additional software module and offering an interface to the Control Function which will take care of the bridging mechanisms. This interface could be for example an abstract JAVA interface (which was employed in the validation scenario), or an XML-RPC based one. By following this interface-based scheme new protocols can be easily plugged, without affecting other PSPH' tasks, as well as, maintain flexibility from its location in the network.

2) Application Addressability

In the architecture, PSPHs are the only elements that interact with all the domains. In our architecture the application addressability cannot be completely managed by the PSPH without interfering the network flow. In a standard scenario, the PSPH does not control the naming resolution system used by every PuSH hub or terminal. So, it should communicate with the DNS system and update a specific SRV record. An example of this SRV record [9] might be: *_psirp._tcp.ericsson.com. 86400 IN SRV 0 5 1025 psph.ericsson.com*. Therefore, any application could be easily addressable using an already present and simple process. This is one of the available alternatives for not modifying applications and allows them access to information located in other domains.

3) Server Selection

Our proxy-based architecture can introduce disadvantages concerning reliability; as well as a single failure point could affect the whole network and be a bottleneck. However, the architecture allows introducing several PSPHs as the entrance point of the PSIRP network. In this case, network elements can discover current available or preferred PSPHs, in order to send the information. In the particular case of the PuSH protocol, the presence of several Hubs allows to have some sort of redundancy. In the same way, the publisher exactly defines which PuSH hub will be publishing its information. So a subscriber will ask directly for this data. However in complex scenarios with multiple PSPHs and PuSH Hubs per domain, there may be information that needs to be filtered before it reaches subscriber's terminal. So the existence of several PSPHs could leverage into data duplication. However, the PSIRP API performs data versioning tasks, so every new generated publication unit is delivered as a controlled event and the Rendezvous function keeps track of it. Consequently, the proxy selection process in the host-centric domain does not alter the PSIRP capabilities of carrying data, so most-sophisticated reliability mechanisms or even load balancing techniques could be implemented.

IV. Implementation

The prototype is focused on two specific scenarios. In the first scenario, the native Publish/Subscribe network (PSIRP) carries feeds between the PubSubHubbub elements. Feeds are published in the publisher domain, and then, the subscriber gets them using a standard web browser. The second scenario implements a synchronous chat application.

[4]Blackadder API v0.2.0 - https://github.com/fp7-pursuit/blackadder

[5]Blackhawk API v0.3.0 - code.psirp.org

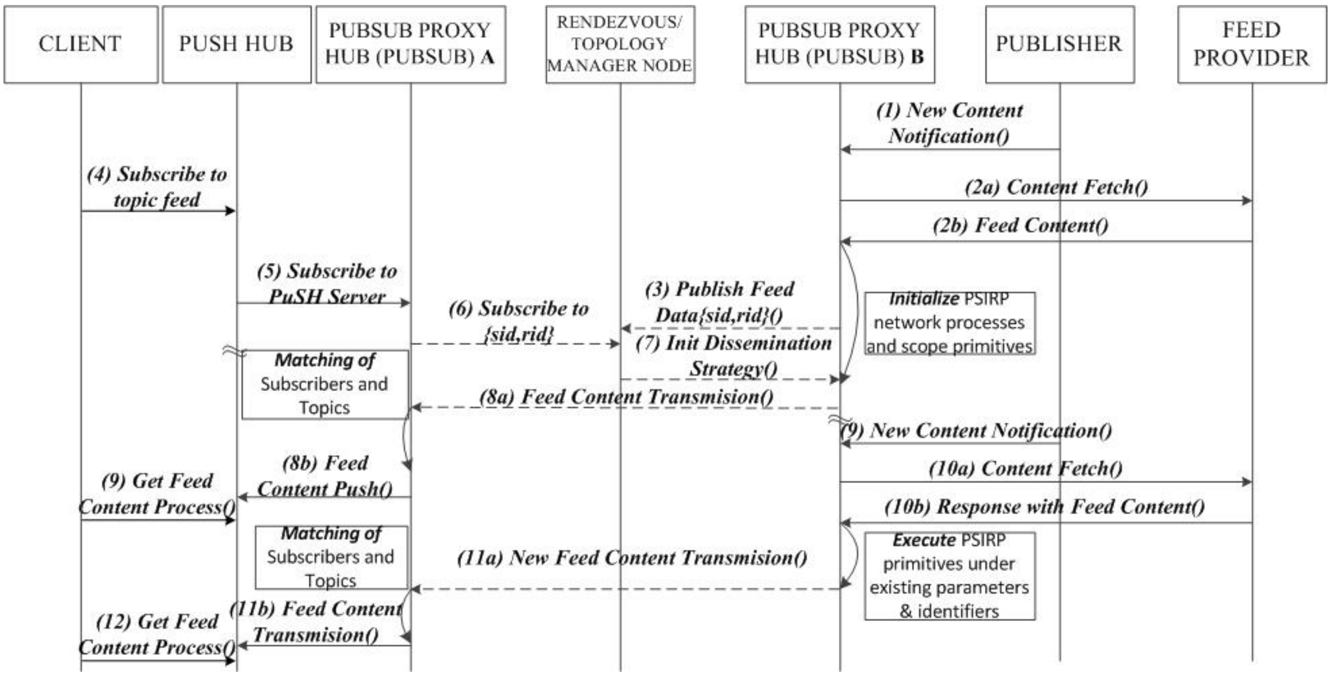

CLIENT
PUSH HUB
PUBSUB PROXY HUB (PUBSUB) A
RENDEZVOUS/ TOPOLOGY MANAGER NODE
PUBSUB PROXY HUB (PUBSUB) B
PUBLISHER
FEED PROVIDER
(1) New Content Notification()
(2a) Content Fetch()
(2b) Feed Content()
(4) Subscribe to topic feed
(5) Subscribe to PuSH Server
(6) Subscribe to {sid,rid}
(3) Publish Feed Data{sid,rid}()
(7) Init Dissemination Strategy()
Initialize PSIRP network processes and scope primitives
Matching of Subscribers and Topics
(8a) Feed Content Transmision()
(9) New Content Notification()
(8b) Feed Content Push()
(9) Get Feed Content Process()
(10a) Content Fetch()
(10b) Response with Feed Content()
Matching of Subscribers and Topics
(11a) New Feed Content Transmision()
Execute PSIRP primitives under existing parameters & identifiers
(12) Get Feed Content Process()
(11b) Feed Content Transmision()

Figure. 5: Communication Diagram

The terminals used for testing end-to-end communication ran Windows and Linux operating systems. The PSIRP network is composed by two PSPH machines that are connected through a 100/1000 Mbps Ethernet switch. Every PSPH has two network cards attached, one for the PSIRP domain and another for the PuSH domain. Since the PSIRP network does not required IP packets in order to work, the PSPHs do not have any other sort of IP layer configurations in the PSIRP side. The additional network card has an IP which is part of its specific PuSH hub. In addition, the PuSH Hubs are programmed in Java JDK 1.6, and run Ubuntu 10.04 64 bits, and make use of the latest PSIRP implementation called Blackadder v0.2.0, which is more efficient than the old Blackhawk v0.3.0 implementation.

A. Publish/Subscribe Feed Scenario

When a client subscribes to a specific URL topic, which represents the information the client wants, this request goes to the PuSH Hubs and then it communicates with the PSPH using the PuSH protocol. The PSPH transforms the requested information (the URL topic) into an information-centric compatible request, previously generating the scope identifiers <Sid,Rid>, and then express it interest on this kind of information through the PSIRP network. In the publisher side, when the content producer publishes data, it goes to the PuSH server using PuSH messaging who informs the PSPH regarding this new information. Thus, the PSPH publishes the information in the PSIRP network using scope identifiers. In order to deliver information from the PSIRP publisher side to the subscriber, both identifiers Sid and Rid must match.

1) Publish/Subscribe Feed Network Flow

In the first step (1), the Publisher informs the PSPH B that there is new content available in its feed provider while attach the *href* URL that identifies its feed. This PSPH integrates the PuSH Hub's functionalities in order to support feed publishing and simplify the communication diagram. Afterwards, the PSPH sends (2a) a content fetch request to the right feed provider and gets (2b) the content which needs to be published. Once it has the information, it publishes (3) its willingness to provide content delimited by the Sid and Rid, through the PSIRP network. These identifiers are generated by the Conversion Function using the *href* URL as the seed. In this step, the PSPH B initializes the PSIRP API, which will be listening until it is informed that a subscriber is interested in its content. In the same manner, the Mapping function maintains a registry of active topics for controlling the way future content, which match existing Sid/Rid, will be published.

The client, within the Subscriber domain, who already knows the *href* URL where data is located, subscribes (4) to its preferred PuSH Hub. Then, the PuSH Hub sends (5) a subscription message to the PSPH using the PubSubHubbub protocol. The PuSH effectively locates the PSPH A IP address by asking its DNS SRV record [9]. In the next steps, the PSPH A issues (6) a subscription information using the new Sid/Rid identifier. These scope identifiers are generated by the Mapping/Conversion function by splitting the *Fully Qualified Domain Name* (FQDN) and the host part of the *href* URL. In this point, the subscription data is stored by the Control Function using a Bloom Filter. Then, the PSPH A makes use of the PSIRP API in order to send (6) the subscription information to the PSIRP network.

When the PSIRP Rendezvous node receives the subscription information from the PSPH A, it matches this subscription with a publication that already exists in the network. Thus, the Rendezvous Node sends (7) an Init Dissemination Strategy messages to the Publisher. This message allows, the PSIRP API, sending (8a) the feed content to the subscriber's network address, through the LIPSIN [12] identifier. From now on, every time a new piece of data matching with a PSIRP identifier is available, the PSPH gets the information event from the PSIRP network and recovers the corresponding topic. Following this, the control function verifies the subscription status of the PuSH Hubs, and pushes (8b) asynchronously the information to the right Hub.

In the next phases, the client gets the information (9) from the PuSH Hub, using standard HTTP GET polling request generated by its web-browser. The method the client uses to receive these data is completely separated from the proposed architecture. Indeed, there are other efficient mechanisms, such as HTTP long polling or WebSocket[6], which are more suitable for this task.

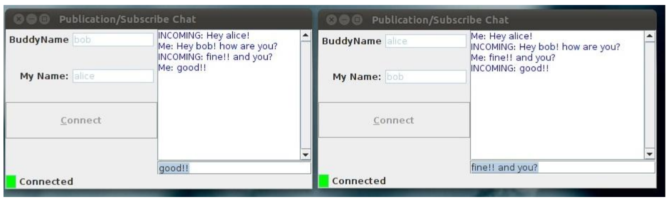
In the final step, the PSPH B gets content (9, 10a, 10b) as before. However, as it becomes aware of a previous publication under the same scope, and the subscribed has not been unsubscribed, it directly republishes (11a) the new content to the subscriber. In the same way, the PSPH A transmits (11b) the information to the PuSH Hub and the client gets (12) the content.

B. *Bidirectional Chat Communication Scenario*

The chat scenario shares similar characteristics with the scenario explained in section IV-A. As we described in section III-B.1, the PSPHs are slightly modified in the Control Function Interface as the first entry point of chat messages. We used the following XML data format for transmitting messages between chat participants:

```
<?xml version="1.0" encoding="UTF-8"?>
<rss xmlns:dc=
"http://purl.org/dc/elements/1.1/" version="2.0">
  <channel>
    <title>New message from:</title>
    <link>http://ericsson.com/alice</link>
    <description>Hello Bob! How are you?</description>
    <dc:creator>alice@ericsson.com</dc:creator>
  </channel>
</rss>
```

This scenario takes the chat participant username and the domain (e.g. alice@mydomain.com) as the chat participant identifiers, which are included in the same XML. This scenario implements a similar hashing technique than the one used in the scenario described in section IV-A in order to generate Sid and Rid for the PSIRP network. In the whole scenario, both chat participants subscribe to their buddies' messages through the PSPH. However, they do not notice the existence of the PSIRP network. This scenario assumed that chat participants already know their buddys username and domain. The chat scenario demonstrates that PSPHs can support subscriptions and publications under the same TCP/IP interface and accomplish bi-directional communication; however, there are still factors, which should be taken into consideration, such as: user identification, security and discovery mechanisms, but are out of the scope of this publication.



INCOMING: Hey alice!
Me: Hey bob! how are you?
INCOMING: fine!! and you?
Me: good!!

good!!

Me: Hey alice!
INCOMING: Hey bob! how are you?
Me: fine!! and you?
INCOMING: good!!

fine!! and you?

Figure. 6: Chat Communication Scenario

C. *Deployment of Applications*

Because data discovering mechanisms are out of the scope of this article, we assume the subscriber is aware about the information which wants to subscribe. In both scenarios, a terminal first subscribes to the PuSH Hubs using its web-browser. Next, this PuSH hub communicates with the PSPH. Consequently, our scenario does not need any specific configuration in the terminal side. The same fact occurs in the publisher's side which does not need to be modified.
All the PuSH hubs follow the PubSubHubbub reference Hub [7]. In the case different applications wish to communicate through the PSIRP network, the PSPH has to implement additional logic in order to understand the new protocol and the data content. Thus, the control function could have filtered information, register the clients subscription, and generate the identifiers. In the Publisher's side applications must follow the same process.

V. Experiments

The purpose of this section is to evaluate the properties of the *PubSub Proxy Hub* (PSPH), as well as, compare different options to have a more efficient PSPH. The results of these experiments led us to take some implementation decisions in our new version of the prototype. The experiments were based on the same PSIRP testing network described in section IV.

A. *Impact of the Publication's Data Payload on the PSPH Node*

The first measurement examines how the payload of a publication messages affects the processing time of the PSPH node. The measurements of the different payloads range from 287 bytes to 1442 bytes. The minimum payload is marked as 287 bytes which is the smallest consistent payload the PSPH node should send or receive following the Atom RFC [10]. The maximum payload is 1442 (1500 - 58 = 1442). This 58 bytes is the theoretical maximum TCP/IP overhead. If a packet is larger than this value, the packet is fragmented before arrives to the PSPH.

Figure 7 shows the data delay aggregated by the PSPH node as the data payload increases. The X axis shows the data payload while the Y axis shows the time (in seconds) since the publication is issued to the network until the PSPH node, which acts as subscriber, receives the information.
The results show that the average delay is 9.607ms. As the delays are quite consistent, we confirm that the implemen-

[6] WebSocket API Homepage - http://dev.w3.org/html5/websockets/

[7] http://code.google.com/p/pubsubhubbub/wiki/DeveloperGettingStartedGuide

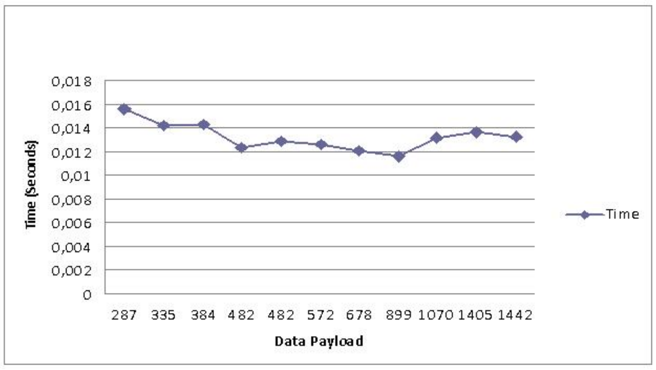


Figure. 7: Impact of the Publication's Data Payload

tation performs stable and there is not a real impact in the PSPH node.

B. *Impact of the Number of Publications on the PSPH Node*

The second measurement examined how the number of publications affects the PSPH latency. As explained in section III-A, the PSPH node must perform a conversion function for every *topic URL* and the Sid /Rid identifiers. These processes not only affect the subscription events that have to be inserted into the PSIRP network but also it increases the conversion and matching functions that must be carried out. For this reason, we focused on keeping a sustainable delay since data is received until the data is available to the right subscribers.

In the first version of our prototype [4], we matched *topic URL* events and their corresponding subscribers using standard Java vectors classes. This method, which allowed us to keep ordered topics and subscribers information, was enough for a basic tested scenario where topics keep below than 50. However, as the figure 8 shows, this method lacks of performance when the PSPH node needs to register multiple topics. Thus, we have implemented a new storage strategy for topics by using Bloom Filters[8], which have proved to offer space and time advantages over hash tables, arrays and other data structures.

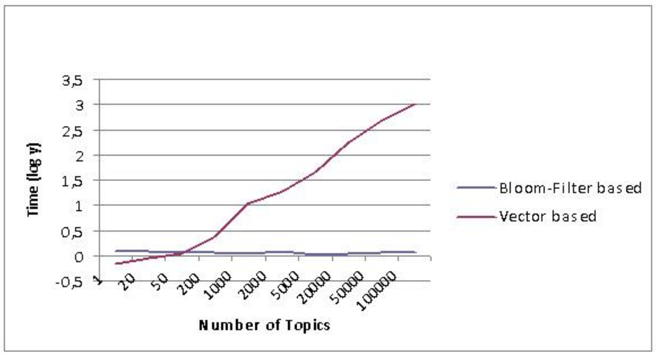


Figure. 8: Impact of the Number of Publications

Figure 8 depicts the advantages we introduce by using Bloom Filters in the matching processes. We have set the Bloom Filter's expected number of elements in ten millions, and the false positive probability to 0.00001%. In the first evaluation phase the vector strategy performs better than the Bloom Filter strategy, mainly, because the PSPH node has to initialize the Bloom Filter and the vector comparison make use of already loaded java classes. The topics' processing time threshold starts in 50, and we can clearly sustain that using Bloom Filters provides much better performance than the previous strategy.

C. *Impact of the Bloom Filter's key-selection on the PSPH Node*

The PSPH node - when acts as a subscriber node- has to maintain a persistent state between the *topic URL* and the subscribers subscribed to that topic. Our implementation uses Bloom Filters to keep this data persistent. However, there are two different alternatives to store the topic-subscribers correspondence information in the Bloom Filter. The first alternative carries out a Subscriber-based matching, in which the Bloom Filter is sorted out by subscribers. The Topic-based matching approach sorts out the data by topics.

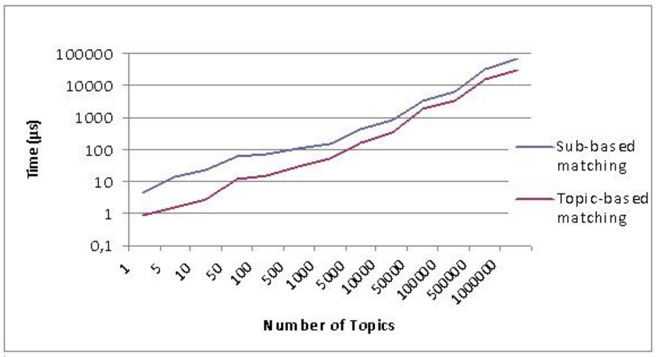


Figure. 9: Bloom Filter's key-selection

The third measure studies how the key selection of the Bloom Filter affects its processing time. This option considers the case when some data has been sent to the PSPH nodes and the PSPH node has to distribute this data to the different subscribers in the host centric network.

Figure 9 shows the difference in delay in the matching process using both approaches while increasing the amount of topics that are recorded in the matching registry. The Topic-based matching approach is four times faster, on average, than the Subscriber-based matching approach. These results emphasize the use of the Topic-based matching approach in the implementation.

VI. Conclusion

We have presented and evaluated an alternative for integrating current host centric network with a Future Internet Architecture [9], as well as proved that a real time feed protocol such as PubSubHubbub can be supported over PSIRP [8] by developing our proxy-based architecture. Our

[8]Java-BloomFilter API- https://github.com/magnuss/java-bloomfilter

[9]http://www.psirp.org

contribution does not pretend to demonstrate which of the several Future Internet Network Architectures provide the best advantages. However, it is clear that even with the deployment of novelty network, migration mechanisms are still needed. Some architecture model proposals for Future Internet are based on the Publish/Subscribe paradigm, which presents many advantages over the classic send-receive paradigm, such as decoupling between location and information and asynchronous communication. Hence, with the purpose of providing a feasible migration path, it is necessary to define interoperable solutions among current Internet services and the capabilities offered by Future Internet.

The architecture proposed in this paper achieves interoperability between the currently evolving PSIRP architecture, which proposes a new Internet model based on Information-centric networks, and the PuSH protocol, which enables distributed communication between publishers and subscribers. In addition we have validated solutions for migrating a chat application into the PSIRP network using PubSub Proxy Hubs, as an additional example of future service interoperability. Other issues have been addressed, such as subscription, mapping and conversion of different identifiers. Since information and location are decoupled in the PSIRP architecture, our paper proposes a solution for taking advantage of this characteristic by generating compatible identifiers from standard Internet resource identifiers.
In conclusion, after the evaluation phase, we confirmed that the amount of subscription entries in the PSPH node can be increased using a Topic-based Bloom Filter strategy. We have also found that our Proxy barely affects the performance of the Inter-network communication, when the size of the payload increases. Thus, as our architecture's implementation scales well for feed content, we have provided standard evaluation information which could be useful for future large-scale deployments with multiple PSPHs.

Acknowledgments

We would like to thank Jimmy Kjällman for his help with the Blackhawk and the Blackadder implementation's. Also, we would like to thank Kristian Slavov.

References

[1] S. Arianfar, P. Nikander, and J. Ott. On content-centric router design and implications. *ACM Workshop on Rearchitecting the Internet (ReArch) at Conext 2010*, 2010.

[2] Andrei Broder and Michael Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, pages 636 – 646, 2002.

[3] Lagutin D and Tarkoma S. Forwarding challenges and solutions for a publish/subscribe network. *ICT Mobile Summit 2009*, 2009.

[4] Augusto Morales Domingues, Oscar Novo, Walter Wong, and Tomas Robles Valladares. Publish/subscribe communication mechanisms over psirp. *Next Generation Web Services Practices (NWeSP), 2011 7th International Conference on*, pages 268 – 273, December 2011.

[5] P. Eugster, P. Felber, R Guerraqui, and A. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 2003.

[6] Anja Feldmann. Internet clean-slate design: What and why? *Computer Communication Review, ACM SIGCOMM*, 2007.

[7] Nikolaos Fotiou, George C. Polyzos, and Dirk Trossen. Illustrating a publish-subscribe internet architecture. *Journal on Telecommunication Systems*, 2011.

[8] Nikos Fotiou, George C. Polyzos, Pekka Nikander, and Dirk Trossen. Developing information networking further: From psirp to pursuit. *an invited paper in 7th International ICST Conference on Broadband Communications, Athens, Greece*, 2010.

[9] IETF. *RFC 2782. A DNS RR for specifying the location of services (DNS SRV).*

[10] IETF. *RFC 4287. The Atom Syndication Format*, 2005 December.

[11] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking named content. *Proceedings of the 5th international conference on Emerging networking experiments and technologies, Rome, Italy*, 2009.

[12] Petri Jokela, Andras Zahemszky, Christian Esteve Rothenberg, Somaya Arianfar, and Pekka Nikander. Lipsin: line speed publish/subscribe inter-networking. *SIGCOMM '09: Proc. of the ACM SIGCOMM 2009 Conference on Data Communication*, 2009.

[13] G. Pallis and A. Vakali. Insight and perspectives for content delivery networks. *Communication Magazine. ACM 49*, 2006.

[14] PSIRP. *PSIRP Publish-Subscribe Internet Routing Paradigm. Deliverable D3.5. Final Description of the Implementation*, 2010.

[15] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2:277–288, November 1984.

[16] Dirk Trossen, Mikko Sarela, and Karen Sollins. Arguments for an information-centric internetworking architecture. *SIGCOMM Comput. Commun*, 2010.

[17] A. Zahemszky, A. Csaszar, P. Nikander, and C.E. Rothenberg. Exploring the pub/sub routing & forwarding space. *IEEE International Conference on Communications Workshops*, 2009.

Author Biographies

Augusto Morales received his B. Sc (2007) from the University of Panama, and his M.Sc. (2010) from the Technical University of Madrid, Spain. Since 2008 he has been working in several areas related Distributed publish/subscribe Middlewares, Service Architectures and Network Security while he pursues his PhD. He holds several IT Certifications such as CEH, Security+, Linux+ and CCSE. He is member of IEEE and ACM

Oscar Novo received his M. Sc. degrees in Telecommunication Software and Computer Science from the Aalto University (Finland) and from the *Universidad Politècnica de Catalunya* (Spain). He is currently working at Ericsson Research in Finland. His research interests include signaling, multimedia applications, and transport protocols.

Walter Wong received his B. Sc and M. Sc and Ph.D. degrees in 2005, 2007 and 2011 from the University of Campinas, Brazil. Since 2011 he is reseacher at CPqD and University of Campinas. His research interests include clean-slate architectures, security protocols and naming systems.

Ramon Alcarria received his Master's degree in Telecommunication Engineering from the Technical University of Madrid in 2008. Currently, he continues his studies as a PhD student and participates in several national and international research projects. His research interests are Service Architectures, Sensor Networks, Service Composition and Prosumer Environments. He is a member of IEEE, IEEE Communication Society and ACM.