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

DELIVERABLE D2.3

Architecture Definition, Component Descriptions, and Requirements

Document Properties:

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:	Architecture Definition, Component Descriptions, and Requirements
Date of preparation	27.02.2008
Author(s)	Mark Ain (TKK-HIIT) (editor), Dirk Trossen (BT), Pekka Nikander (LMF), Sasu Tarkoma (TKK-HIIT), Kari Visala (TKK-HIIT), Ken Rimey (TKK-HIIT), Trevor Burbridge (BT), Jarno Rajahalme (NSNF), Janne Tuononen (NSNF), Petri Jokela (LMF), Jimmy Kjällman (LMF), Jukka Ylitalo (LMF), Janne Riihijärvi (RWTH), Borislava Gajic (RWTH), George Xylomenos (AUEB), Petri Savolainen (TKK-HIIT), Dmitrij Lagutin (TKK-HIIT)
Responsible of the deliverable	Sasu Tarkoma (TKK-HIIT) Phone: +358 50 384 1517 Email: sasu.tarkoma@hiit.fi
Target Dissemination Level:	PU
Status of the Document:	Completed
Version	1.0
Document location	http://www.psirp.org/publications/
Project web site	http://www.psirp.org/

SEVENTH FRAMEWORK PROGRAMME

European Commission
Information Society and Media

Table of Contents

1 Introduction

The PSIRP project is an EU FP7 funded project with a 30 month lifetime. Its ambition is to investigate major changes to the IP layer of the current Internet, finally replacing this layer with a new form of internetworking.

As Mark Handley puts it [Han06]: 'the Internet only just works'. We can observe that the current dominant internetworking solution, the Internet Protocol suite, suffers from a number of limitations, although it works reasonably well for current demands. Challenges include ones related to efficient support for mobility, efficient global multicast, and multi-homing. But in addition, vulnerability to unwanted traffic and Denial-of-Service (DoS) attacks is a growing and often fundamental problem with the current Internet architecture.

We see the main reason for the shortcomings of the current IP-based Internet being deeper embedded in the underlying paradigm of communication than in its operational shortcomings. More specifically, the current endpoint-centric communication paradigm places rather arbitrary topological constraints on the delivery of information. With the observed increase of information-centric services, such as the World-Wide Web or newer contemporary applications such as sensor networks, the inflexibility of endpoint topologies increasingly places a burden on solution developers that needs circumvention by virtue of ever increasing number of overlays. This leads to a lack of flexibility and to an increasing rigidity; a circle that PSIRP intends to break.

Our approach in this project is to investigate a new information-centric communication paradigm in which we consider any communication scenario as being concerned with the production, retrieval and consumption of **information**, all of which are surrounded by concerns of the different players involved in that particular scenario. The notion of **intention** becomes crucial in the establishment of communication between two parties, where the match of intentions is the key to successful communication, not the reachability of endpoints. With that, our approach moves the balance away from the currently sender-driven IP model. Instead, we advocate a **publish-subscribe model** in which receivers of information have control over their expression of interest and therefore the reception of information.

A Vision-driven Methodology

Our work is driven by a clear methodology that combines bottom-up and top-down approaches, as outlined in [PSI2008d] and shown in Figure 1.1 below, in phases of vision making, definition of goals, leading to principles and requirements for the architectural work with the eventual definition of components and an architectural specification that can be used for the implementation work and evaluation. All these phases interact through educating each other in feedback, e.g., from implementation experiences back into architectural concepts and vice versa.

It is the definition of a clear vision that drives our work, as expressed in [PSI2008b]. This vision outlines a future driven by the principle of conflicts (**tussles**) being resolved in runtime through mediation of conflicting policies as well as isolation of tussles, all of which is enabled by a provisioning plane that is entirely driven by information-centric concepts. While we formulate this vision with broad concepts, therefore outlining what a PSIRP world might look like in the future, we define the scope of our work within this vision as clearly being focussed on the routing and forwarding fabric that underlies this vision, i.e., the provisioning plane as outlined in [PSI2008b].

This vision enters into our work through the idea that *information is everything* and *everything is information*, allowing for providing the basic building blocks to incorporate data and metadata concepts into the architecture. This will enable us to extend towards the formulated vision of automated tussle mediation during runtime.

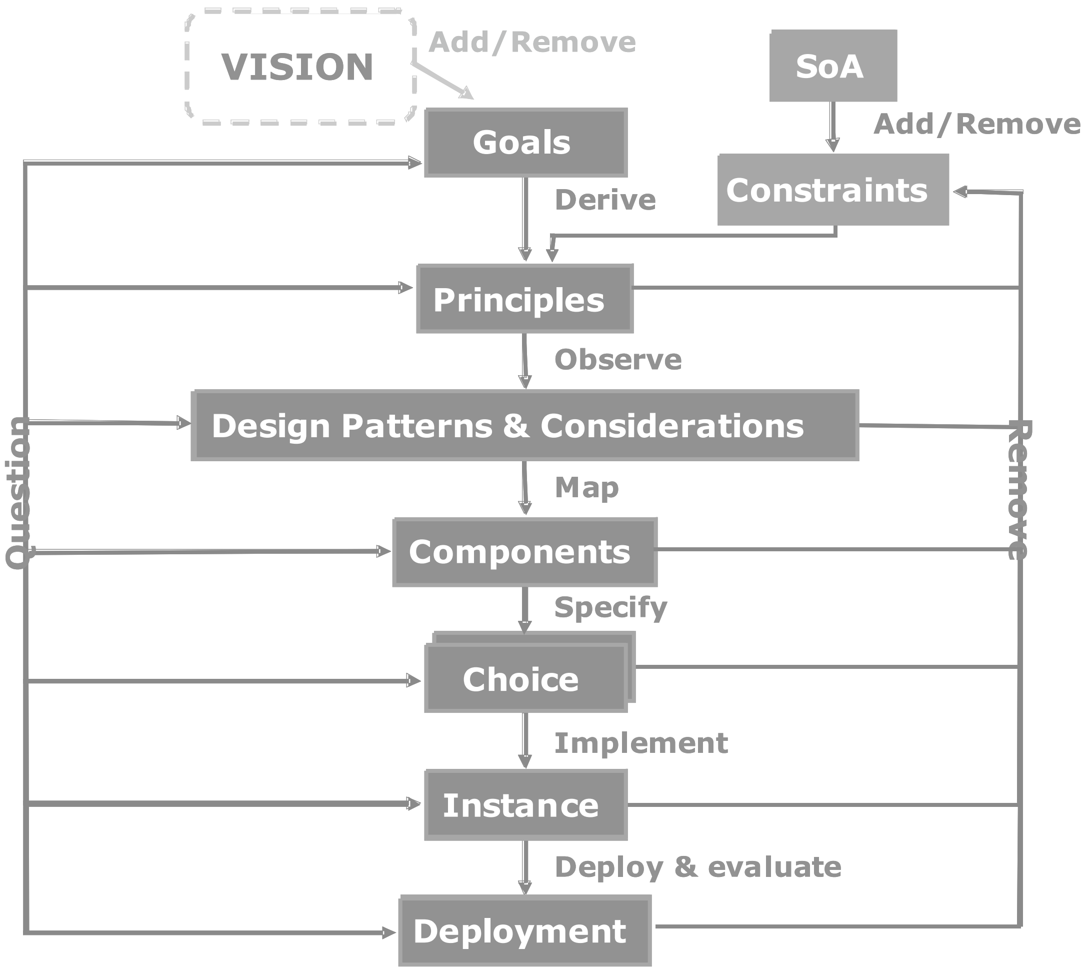

Figure 1.1 – System design process of the PSIRP project

From Goals to Principles

Following our methodology [PSI2008d], we define clear **design goals**, being expressed as follows:

- Devise methods for building information hierarchies as layered graphs.
- Devise methods for flexible information reachability.
- Devise a basic communication model, which allows for increased receiver control.

Moving one step further leads us to **design principles** for our work. These principles are not only concerned with the particular goals of our architecture along the lines of the abovementioned goals but with general principles for developing any Internet-scale architecture. Hence, we divide our principles into **general** ones and those specific to the **PSIRP** design. The former can be described as principles that are relevant from an architecture design angle, i.e., one can argue about their applicability for similar architectural efforts. An example for this is the *Trust-to-Trust* (T2T) principle [Cla2007] that describes the placement of functions according to the trustworthiness of the component implementing the function. Another principle resolves around the importance of enabling markets in an internetworked architecture, taking into account the (potentially opposing) interests of the different stakeholders. These general principles can be found in [PSI2008d]. Our PSIRP-specific design principles can be summarized as follows:

- **PS1 – Information is hierarchically organised**: Higher-level information semantics are constructed in the form of "directed acyclic graphs" (DAGs), starting with meaningless forwarding labels and moving towards higher-level concepts (e.g., ontologies), or vice versa.

- **PS2 – Information scoping**: Mechanisms are needed that allow for scoping information on different levels of semantics in the architecture (e.g., via means of rendezvous, discovery, search, and others). With this, information scoping limits the reachability of information to the parties having access to the particular mechanism (e.g., rendezvous) that implements the scoping.
- **PS3 – Scoped information neutrality**: Within each scope of information, data is forwarded based on the given (scoped) identifier, i.e., the architecture is neutral with regard to the semantics and structure of the data.
- **PS4 – The architecture is receiver-driven**: No entity shall be delivered data unless it has agreed to receive it beforehand through appropriate signalling methods (i.e., excluding physical means of receiver choice), wherever this can be implemented in the conceptual solutions for our architecture. Limitations to this principle can apply, for instance, at the physical level.

Returning to Our Methodology...and Moving Forward

While the formulation of clear design principles is seen as an important step in our architecture development, we now return to our methodology. It is important to stress that all steps, including the formulation of our design principles, are part of a constant educational cycle during which lessons learned in other steps, such as in the implementation or results of our evaluation, lead to refinements and revisions of steps elsewhere, such as the (careful) revision of design principles.

In the following chapters, we present the results of the further steps of our methodology, namely the **concepts** that are derived from design patterns that we recognized in our first main deliverable D2.2 [PSI2008d]. These are manifested in the information concepts and high-level architecture view that are outlined in Sections 2 and 3 of this report, respectively. The more detailed description of our components is presented in Section 4. Design considerations spanning across the entire architecture, such as for security and mobility, are outlined in Section 5 before turning to application considerations in Section 6.

2 Information Concepts

With the focus of PSIRP being *information*, it is critical to understand the various *information concepts* that are provided by the PSIRP architecture. These concepts are directly reflected in the various architecture components, in particular the identifiers (see Section 4.1) and the operation of the rendezvous system.

Everything is Information

The most basic notion of PSIRP is that everything is information by virtue of identifying a bucket of data with a statistically unique label. This label is used in the process of *rendezvous* (see Section 4.4) to create the transient relation between the publisher of this information and the subscribers to this information. For this reason, we call the identifier being used for information the **rendezvous identifier** (RId).

The PSIRP information concepts do not prescribe what exactly this bucket of data, the actual information, would be. We will see later that the architecture itself builds up more complex information structures from simple memory pages (as buckets of data) over channels of data to versioned documents or streams of video data. Typical examples at the application level could be files, e-mails, still or individual video images etc. As introduced in D2.2 [PSI2008d], the information items can include information about other information items by virtue of reference. This allows for introducing the concept of *metadata* of data, which can be used to group information at the application level, based on some specific semantic for the particular group of information. We will return to the problem of grouping information after the next section.

Building Networks of Information

The PSIRP concepts allow for grouping individual pieces of information, e.g., mail or images, into so-called *information networks*, following the scoping concept as introduced in [PSI2008d]. With that, information can be logically grouped together in order to enable, e.g., controlled access to this information for a limited set of publishers and subscribers or to enforce particular governance rules for the given information network. This is illustrated in Figure 2.1.

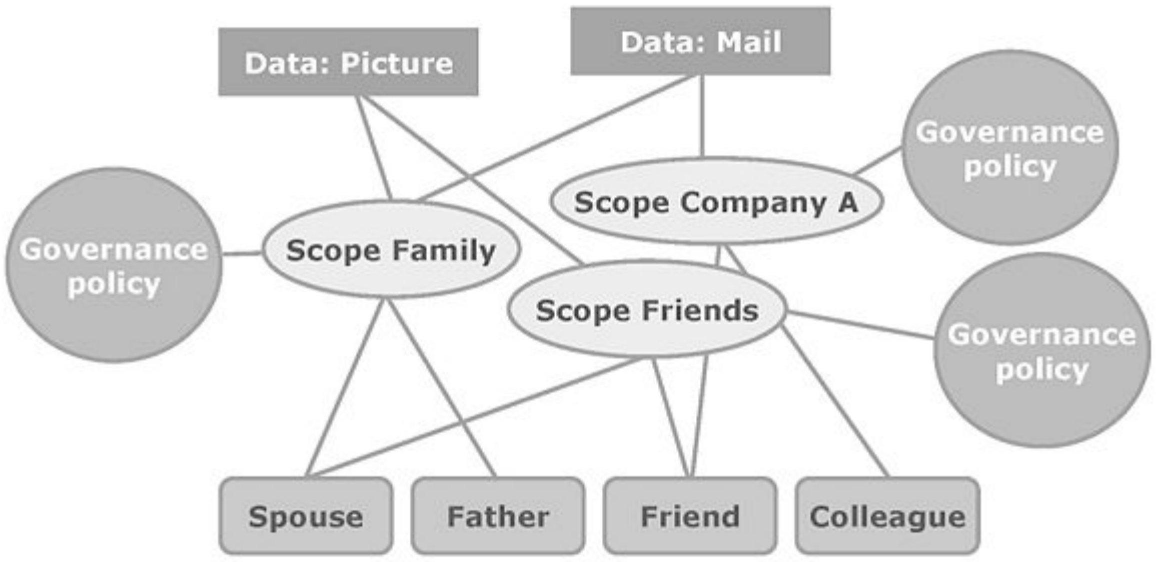

Figure 2.1 – Building information networks

Each information network represents information in itself and therefore requires an identifier in order to be addressed. To reflect the specific purpose of information networks, i.e., to scope the reachability of information, we introduce the **scope identifier** (SId) as a specific subclass of rendezvous identifiers. We will later see in the rendezvous function (see Section 4.4) that SIds represent the rendezvous points implementing a particular information network (identified by the respective SId), i.e., the *information item* the SId represents is the information network itself (being a collection of information items).

Grouping Information within Networks

While information networks already enable the grouping of information into larger sets, the PSIRP architecture introduces an intermediary concept, i.e., one that ranges in-between scopes and individual information pieces. We call this concept *information collections*.

As a motivating example, we use a video stream. Such a stream can be identified with a single RId, labelling the entire bucket of video data with the same RId. If, however, a subscriber would like to subscribe to individual segments of the video or even individual images[1], the opaque addressing of the entire stream would not allow for this. As an alternative, the publisher of the video stream could label each individual image as a separate information item. This would create some 25 individual information items per second. In a typical application scenario however, these seemingly separate information items belong semantically together. Knowledge about this semantic relationship could enable the network to optimize operations of, e.g., forwarding (using same forwarding trees) or caching.

Other examples of information items that semantically belong together but would be created as seemingly separate information items arise due to fragmentation or retransmission.

While most of these semantic relations could be implemented solely on application layer, using the aforementioned concept of metadata (i.e., using a metadata information item identified with one RId – the metadata containing the Rids of the individual information items in the information collection), the PSIRP architecture does provide means to group information by virtue of **algorithmic rendezvous identifiers**. With this concept, rendezvous identifiers are generated based on a common algorithm. Hence, it is the algorithm that is tying the individually created information items together as one logical unit rather than the metadata information item described above. For more information on algorithmic identifiers, please see Section 4.1.

Grouping Information Networks

We introduced the concept of *information collections* as a means to collect individual information items into sets of information collections within a single information network. But information networks themselves can be collected. These collections of information networks can represent networks within organizations, e.g., corporations, or networks within social structures, e.g., social networking sites. For example, one can publish one of his/her information networks to a particular directory, i.e., the SId of the specific information network is published in another information network, the directory. The nature of SIds as subclasses of RIds obviously enables this kind of grouping.

[1] Realistically, an application would need to take into account encoding-specific constraints like key frame based coding – only allowing to start re-winding video to a key frame rather than an arbitrary one. However, the basic idea still remains, i.e., allowing the subscriber to subscribe to an arbitrary (key) frame.

The following figure shows a particular example of grouping information network.

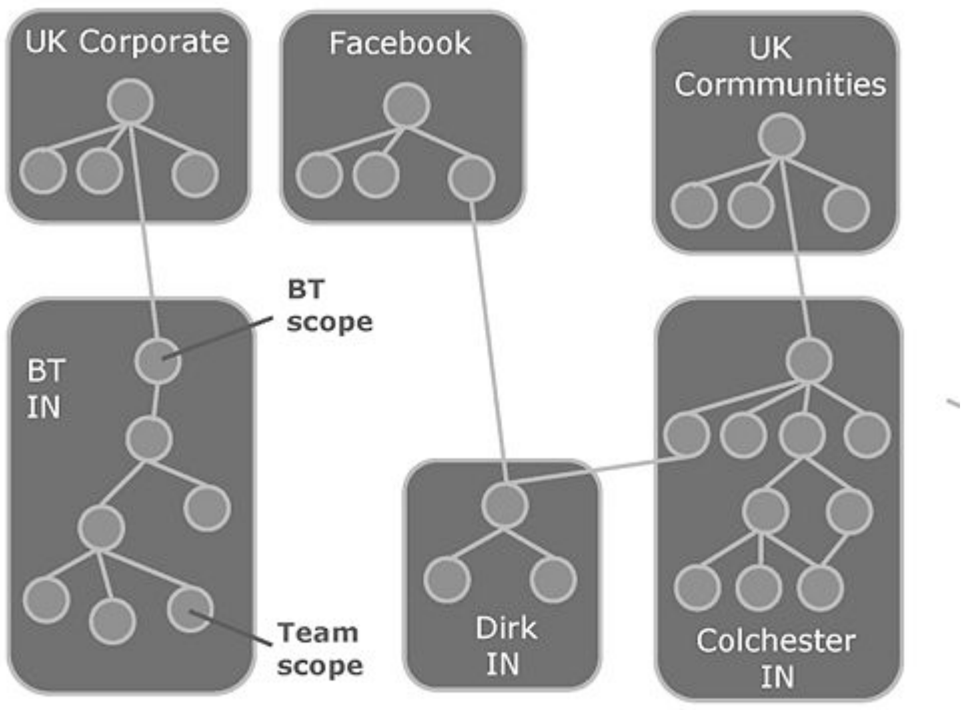

Figure 2.2 – Grouping information networks

When grouping such information networks, we, however, implicitly create information by virtue of the structures themselves. As can be seen from the example in Figure 2.2, this can contain sensitive information such as information networks within corporate organizations like BT. It is unlikely that organizations would like to have these structures revealed without any control over the access to this information.

Given that the information networks are identified by SIds and used in the rendezvous process (see Section 4.4) to resolve to the appropriate rendezvous node that serves the particular SId (imagine every grey circle in Figure 2.2 being such a rendezvous node), there is potential for information disclosure to untrusted parties. For example, an employee of BT would need to divulge the specific SIds of the information networks within BT she would like to connect to. For this, the SIds would need to be transferred to a local rendezvous system (see Section 4.4 for the inner workings of this rendezvous system), which is potentially untrusted. Through gathering the used SIds, the untrusted part of the rendezvous system can easily deduce the information structure of the BT information network as a whole.

To overcome this problem, the usage of an **algorithmic scope identifier** promises a solution. The information structures within a certain governance, e.g., the BT information network as a whole, can semantically be grouped together through a common algorithm (and a seed to initiate a particular sequence). It is this algorithm that is conveyed via the rendezvous system, reaching the topmost node in the information network structure. Executing the algorithm with private information, e.g., encoded in the payload of the information item, allows the topmost node to forward the request to the appropriate rendezvous node, which serves the (algorithmically determined) SId. Hence, the algorithm can be seen as being similar to a top-level DNS domain, such as bt.com, which can be easily divulged to any untrusted party within the rendezvous system. In addition, we could expect that the rendezvous point that serves the publicly exposed (algorithmic) SId has improved security and resilience mechanisms to cope with failures and potential attacks, compared to the individual (private) SIds that are determined by the algorithm. Hence, with the mechanism of algorithmic scope identifiers, we allow for building *private information networks*.

Summary of Concepts

The following figure summarizes the PSIRP information concepts together with the resulting identifiers.

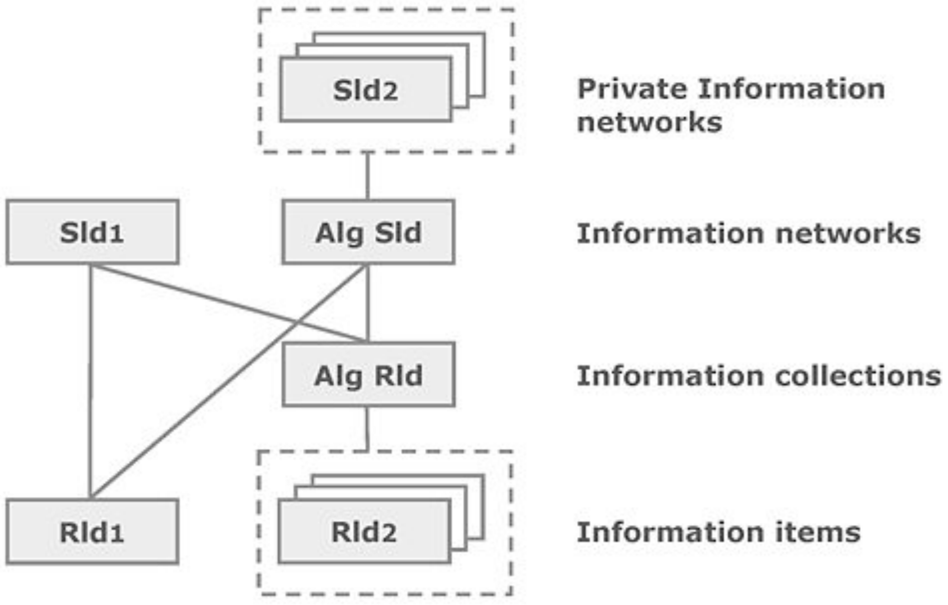

Figure 2.3 – Overview of PSIRP information concepts

It can be seen that concepts span from single information items over collections of semantically similar items to networks of information. Given that networks of information are information themselves, they can be collected together either in new information networks or via information collections, with private information networks being one example of these collections. We believe that these information concepts are rich enough to map any application-specific concepts, such as ontologies or identities, onto these generic concepts while being simple enough to build a scalable routing and forwarding fabric underneath. In Section 4.1, a more detailed presentation is given of the different identifiers that can be found in Figure 2.3.

3 Architecture Overview

This section gives an overview of the functional architecture developed so far in PSIRP. We first outline the extent of our architecture before presenting the major architectural concepts and parts. We then present the service model and the various APIs of our system before delving into the interplay of the architecture components in the so-called *component wheel*. Finally, the components themselves are presented.

3.1 Extent of the Architecture Work

Before delving into the details of our architecture, it is important to understand the **extent** of the architectural work presented here. For this, we distinguish *horizontal* from *vertical* extent, explained in the following paragraphs.

Horizontal Extent

Our architecture aims at providing an internetworking solution with a similar role as IP in the current Internet. Hence, we need to develop solutions that span from local via intra-domain to inter-domain reach, all of which need to scale to the expected dimensions of a future Internet. This requires solutions for, e.g., intra-domain forwarding as well as for inter-domain rendezvous and topology formation.

While we present all of these areas within this document, we need to recognize that solutions in the inter-domain area are largely in an early stage of development.

Vertical Extent

While being centred around the internetworking layer of a traditional layered architecture, our work needs to consider not only the interaction with other layers of the architecture but even the very nature of layering itself. The latter is targeted, for instance, by the RTFM design choice presented in [PSI2008d].

As a consequence, apart from traditional internetworking functions, e.g., inter-domain routing and forwarding, our work also includes considerations on the node architecture (see Section 3.4), considerations on including traditional transport functions via the notion of helper functions (see Section 3.5.4) as well as considerations for the design of applications based on our service model work (see Section 3.3). But it also includes considerations on the concept of layering itself, e.g., through the component wheel concept (see Section 3.4), which aims at a largely layer-less, more modular notion of a node architecture.

3.2 Architectural Entities and Processes

The basis for our architecture is a collection of well-defined **entities**. These entities are based on the major concepts for our architecture, most notably the information concepts as described in Section 2 and the publish-subscribe communication paradigm that underlies our work (see Section 6.4 in [PSI2008d]). With these concepts in mind, we can identify the involved entities as follows:

- *Information item*: as discussed in Section 2, any data can be labelled with an identifier, forming an information item that our architecture can process. At the application level, we assume *application identifiers* of some sort to be used. These are resolved into *rendezvous identifiers*, either directly by the application or via a helper function (see Section 3.5.4). The application finally communicates with the network architecture through the rendezvous identifier, which is used to match the interest for this particular information item between the publisher and the subscriber(s). With this simple labelling of data, i.e., forming of information items, we can implement data and metadata concepts as outlined in Section 6.1 of [PSI2008d].

- *Information network* (or *scope*): information item can be grouped into networks of information, as outlined in Section 2. For this, a *scope identifier* is used to enable labelling the particular information network to be used. A scope determines the elements of the rendezvous system that act on the published data. An information item may be associated with one or more scopes and these scopes can be interpreted at various levels of abstraction.
- *Information subscribers and publishers*, which consume and create information items, respectively.
- *Domains*, which are administrative network areas that can be connected using the inter-domain forwarding architecture. Propagation of information items within and across domains takes place via labelling the items with *forwarding identifiers*. The nature of these identifiers and their function is determined by the intra- and inter-domain forwarding functions.

Figure 3.1 presents an example overview of the above entities and highlights their relationships, based on the current stage of development; these relationships are subject to change based on our implementation experience. As discussed before, data packets and their associated metadata are published and subscribed to in a domain. Typically, the data is associated with one or more application identifiers and one or more scopes. Currently, there are still multiple options for the placement of the scoping information; for example, the scope may be included in the metadata part of the packet or it may be represented as a separate identifier in the packet header. Each application first resolves application identifiers to rendezvous identifiers and then hands the rendezvous identifiers to the network, which then uses the scopes to properly map each rendezvous identifier to one or more forwarding identifiers, both within a domain and between domains.

Based on these architectural entities, we can identify the following five **key processes** to be implemented in the PSIRP architecture:

- *Rendezvous*, which is the process of resolving rendezvous identifiers into forwarding identifiers within a given scope. The scope determines the part of the rendezvous system that is used by the network. The forwarding identifiers may change as the rendezvous state changes. For example, there may be no forwarding identifier available when there are no subscribers for a given publication. The three simplistic, topology-oriented cases for scopes, reflecting the current usage, are link-local, intra-domain, and inter-domain scopes. However, we expect that future applications will use more semantic-based scopes instead of such topological scopes, implementing, e.g., scopes based on social networking structures. The rendezvous and scope identifiers are handles to the rendezvous system. Together they uniquely identify the given publication and the associated policies and metadata.
- *Forwarding* pertains to data delivery within a single administrative domain or across multiple domains. While intra-domain forwarding is concerned with local policies, the inter-domain forwarding system is configured through the rendezvous process and takes into account any inter-domain policies in effect. We note that the routing substrate can be either a new protocol operating on layer 3 or it can be an inter-domain friendly overlay, such as Canon [Gan2004].
- *Topology Management & Formation* is responsible for managing intra-domain delivery topologies as well as forming inter-domain forwarding graphs for publications.
- *Helper functions* are used to extend the core functionality of the network architecture and the provided services towards functions such as management and transport.
- *Network attachment* is responsible for discovering network attachment points and configuring components in such a way that communication becomes possible.

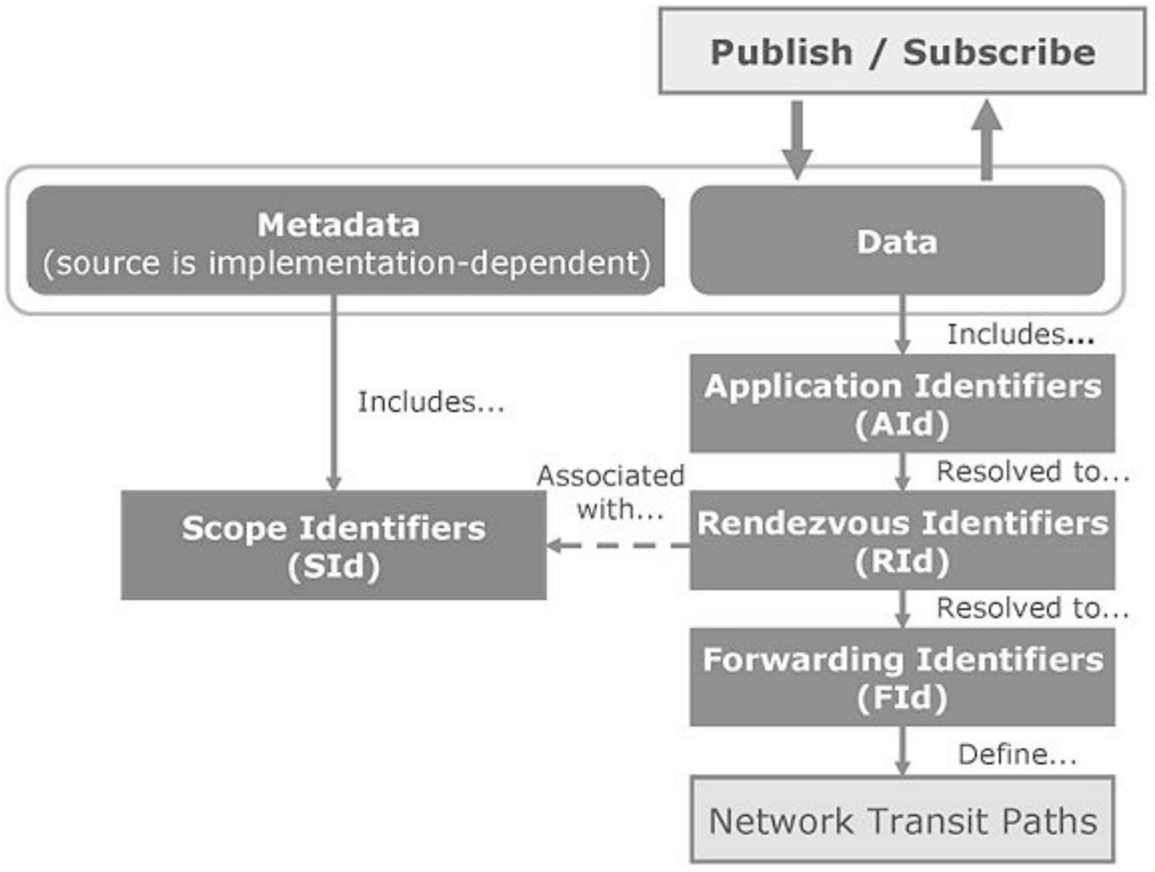

Figure 3.1 – Entities of the PSIRP architecture

With these entities and key processes in mind, we can now divide the architecture into three parts, namely the provided **service model**, the **node architecture**, and the **network architecture**.

The service model (outlined in Section 3.3) below defines the service interfaces towards data subscribers, publishers, and network services. The node architecture (presented in Section 3.4) defines how the entities and key processes are organized and how they communicate within a single node. The network architecture (presented in Section 3.5) defines how the node architecture extends towards the PSIRP network as a distributed set of functions.

3.3 Service Model and API

We currently consider four different classes of network services in the architecture:

1. A low-level page model that exposes the network forwarding and rendezvous/topology formation functions
2. A mid-level memory object model.
3. A mid-level channel model.
4. Various high-level service models, including shared state and document models.

At the lowest level, the size of a publication is one page[2] and there is no error or congestion control mechanism. Publishers (or higher level API functions as publishing intermediaries) can interact with the forwarding function of the PSIRP architecture by sending packets to a forwarding identifier (FId) or identifiers (where more than the initial FId may be specified by the publisher). Subscribers will similarly choose to listen to selected FIds. To obtain FId

[2] For transmission purposes, a page is mapped to a single packet.

information, the publishers and subscribers interact with the rendezvous (to discover subscriber locations) and topology formation functions (to establish policy compliant forwarding paths).

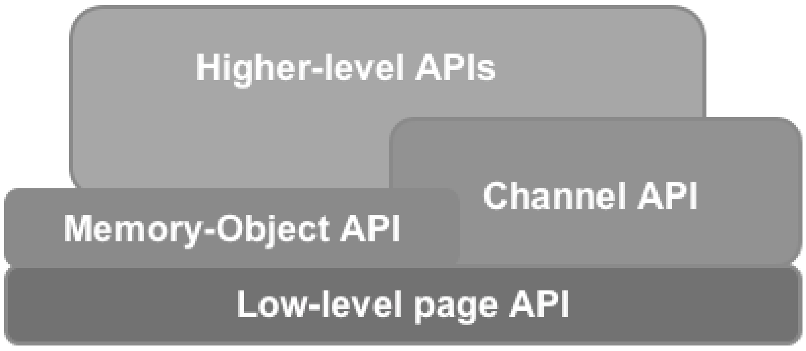

Figure 3.2 – Different levels of APIs

The lowest level API provides a generic exposure of the PSIRP network functionality and may be used to implement different communication concepts. One such concept is to provide an API that shares **shared memory objects** from publishers to subscribers. Such memory objects may be as low level as the packets themselves or higher level application meaningful objects. Publications at this higher level may consist of multiple pages and may have some recovery from packet loss. However, full reliability functionality needs to be implemented by upper level services.

Another communication concept is that of **channels**, where the rendezvous identifier (RId) and subsequently established forwarding paths are viewed as group communication channels. These channels have a meaning that is defined by the application semantics associated to the RId. Such meanings may be "document X", "requests for document X" or they may even encapsulate service concepts such as "document X delivered at 512kb/s" or “a printing service”. Separate channels may be used for flow control or error signalling. The forwarding function of the PSIRP architecture provides unicast and multicast delivery, while the rendezvous function may participate in the formation of multi-publisher channels (for example to a single subscriber, e.g., a server which subscribes to receive requests).

Other mid-level APIs may of course be implemented over the lowest level API, however we expect the shared memory and channel models to be generally useful. Many different specialised APIs (for example, implementing transport functions with customized flow control or error recovery algorithms over the channel model) can also be built upon these mid-level APIs. Other higher-level functions could include reliable multicast, N→1 and N→M services, state sharing, and document models. At this level, we have managed semi-reliable caches, indications of document changes, transmission of document deltas, etc.

In the following sections, the low level model and the shared-memory and channel models are defined in more detail.

3.3.1 Low-Level API

The lowest level PSIRP network service reflects the services offered by the forwarding, rendezvous and topology formation functions within the network. While this is the lowest level service, which can be offered, many other communication paradigms with their own higher level APIs may be offered above this basic low-level service.

At this lowest level, the maximum size of a publication is considered to be one packet and there are no error or congestion control mechanisms. The packet sent to the network is either destined for subscribers (via the forwarding function) or destined for the various rendezvous and topology formation functions whose task it is to provide sufficient information to be able to subsequently forward packets to subscribers. Also, the packets to the rendezvous and topology formation functions travel via the forwarding function. Thus, the operations described in this low-level API can be considered to be layered above the raw forwarding function API methods.

Forwarding

To implement this level of service, there are basic network-sender and network-receiver components attached to the component wheel. They take packet-sized publications and forward them over a FId.

At this level, the conceptual API is very simple:

- listen(FId) → returns stream
- mute(stream)
- forward(FId, meta-data, data)

This interaction with the forwarding function is by design publish/subscribe in nature since this concept flows through all aspects of the PSIRP architecture. However, we deliberately avoid over-using the terms publish and subscribe since we reserve these method names for the interaction with the rendezvous service.

At this API level, the FId denotes a channel (to subscribers, rendezvous components, onward forwarding components etc.). At the local link, FId is only used to distinguish packets into separate channels; however, the node at the end of the link may be configured to forward some FIds (while not forwarding others), making packets sent on certain FIds to flow further than just over the local link. At this level, the structure and contents of FIds are opaque; all the information of which FIds are forwarded, which FIds are listened to by the other end of a link, etc, are available only via the rendezvous and topology components. It is expected that at any given point of time, the vast majority of FIds will be inactive on most links, meaning that an attempt to publish on such a FId will not cause anyone to receive anything.

The in-packet meta-data is limited to a few bytes (exact number open at this writing), and it may be used to carry upper-layer data, such as version, sequence number, signature, or other similar data.

From this service and API point of view, we try to make as few assumptions as possible about the FIds. In particular, one should not assume that certain FIds would be forwarded due to FId-specific state at the forwarding nodes. Instead, for example, a FId may encode a source route, allowing it to be forwarded because it explicitly contains a bit pattern that makes a forwarding node to forward any packets containing that pattern. For the current ideas related to intra-domain forwarding, see Section 4.6.1.

The semantics can be defined as follows: calling listen(FId) at one node prior a call to forward(FId) at another node, with the same FId, may return the packet provided in the call to forward(), depending on whether the delivery tree identified by the FId is configured to flow from the publisher to subscriber. In the case of a local physical link, no pre-configuration is needed; all that is needed is that a node is listening to a FId and another node at the same link is forwarding on the same FId. There is no storage or memory of past events involved at this level.

Rendezvous

The low level PSIRP network service also includes the interaction with the rendezvous components. Although the forwarding functions (and appropriate FIds) are used to communicate packets to the rendezvous system, such FIds are managed by the local rendezvous component within the API and hidden from the application or higher level services.

At the time of writing, it is unclear whether the local host, a local network service, or the rendezvous function will interact with the topology formation function. For the purpose of clarity, it is assumed in this section that the topology formation function is hidden from the host, and hence no direct methods are detailed in this API for interaction with the topology formation function. Instead, the register methods, described below, utilise the functionality of both the rendezvous and topology formation functions.

The purpose of the rendezvous/topology formation methods is to allow subscribers to register their interest and needed topological information to reach them with the rendezvous system as well as to allow publishers to determine which FIds they should use in their publication of data to the forwarding function. The relevant API is the following:

- register_provision (SId, RId, lifetime, policies) → lifetime + FId(s) used in forwarding
- withdraw_provision (SId, RId)
- register_interest (SId, RId, lifetime, policies) → lifetime + FId(s) to listen to
- withdraw_interest (Sid, Rid)

The *register_provision* method is used when a publisher is ready to commence transmission to subscribers and requires forwarding information to do so. As an immediate result, the publisher gets notified about the accepted lifetime of the registration. More importantly, as soon as the rendezvous system determines that it would be useful for the publisher to actually forward data, it provides the publisher with one or more FIds. That is, in the typical case once there is at least one active subscriber, the rendezvous system provides the publisher with a FId; this may be instantaneous in the case there were subscribers before the publisher. In some cases, the rendezvous may provide a FId even if there may not be any subscribers, or it may provide multiple FIds, in which case the publisher should publish the same packets over all of the FIds. If the rendezvous system decides that it is no longer useful for the publisher to publish data, e.g., since there are no more active subscribers, it may forward a path update to the publisher stating that there are no Fids to currently forward subscriber data over, at which point the publisher should cease forwarding.

In a similar manner, the *register_interest* method is used by a subscriber, when it is ready to start receiving data. Again, this function immediately returns the accepted registration lifetime, and one or more FIds. Again, in some cases the rendezvous may provide the subscriber with a FId (or multiple ones) even if there are no publisher yet, and in other cases it may provide the subscriber with multiple FIds; the latter might be used, e.g., for swarmed documents.

In addition to the FIds used for actual data transmission between publishers and subscribers, the rendezvous system internally uses additional FIds. For example, the node-local rendezvous component typically needs one or more FIds for forwarding provisioning and interest registrations to rendezvous points. Likewise, there will be for Fid(s) that are listened to by the publisher or subscriber nodes; that the rendezvous point uses to forward updates on FId status to both the publisher and subscribers.

Publish & Subscribe

It is likely that the direct forwarding and rendezvous function methods detailed in the previous sections will not be exposed to applications. Instead, the lowest level that we expect to be used by applications is a combined publish and subscribe interface that manages and hides the details of the interaction with the forwarding and rendezvous functions.

Subscriber Methods

- associate(RId, SId)
- subscribe(RId)
- unsubscribe(RId)
- attachPolicy(SId, policy)
- attachPolicy(RId, policy)

The most basic subscription method would have been to include both SId and RId in the subscription method. However, the separate SId→RId association allows RIds to be managed and transferred between scopes without requiring the application to issue an unsubscribe request. A single subscribe method for the RId will subscribe to the RId within any associated scopes.

Similarly separate policy management methods allow the policies to be changed without needing to unsubscribe and re-subscribe to the RId.

Publisher Methods

- associate(RId, SId)
- publish(RId, data)
- attachPolicy(SId, policy)
- attachPolicy(RId, policy)
- advertise(RId)

3.3.2 Memory Object Service

This section discusses a communication model for sharing memory objects, through the network, from publishers of new (or revised) objects to subscribers.

Simple Unreliable Page Service

We start with a low-level page service where pages can be mapped to packet level transmissions in the lowest-level PSIRP service. This concept can be used to individually identify and cache individual packets within the network.

Pages sent or received by this service are assigned **pageIDs**, which allows for such memory to be built for the purposes of caching or retransmission at the next level up. Note that pageIDs are not an extension of the PSIRP information concepts (see Section 2) but an internal implementation concept for the memory object service model.

The pageIDs are unique to each FId, allowing one to refer to any page recently sent over the network. If the exactly same bit pattern is resent over the same FId, it should be associated with the same pageID. In the current implementation design, the pageIDs are computed by taking a cryptographically strong hash over the page contents, meaning that the pageIDs are statistically unique over all FIds, not only the one FId it is sent over.

The pageIDs are also intended to be used as a means for error detection, allowing the receiving end to determine if the received packet data does not match the pageID.

Simple Memory-object Service

On the top of this page service, the service model supports larger-than-page memory objects as well as updates to memory-objects. For these, there are rudimentary transport functions that implement recovery from data packet loss, but no congestion control or end-to-end delivery guarantees.

At this level, the API can be presented as follows:

```
  a pair of <metadata, memory-object[length]> :=
       create(credentials, requested-length)
  MO-RId := publish(SId, tag, pair of <metadata, memory-object> )
```

A potential publisher requests a meta-publication and a memory object, indicating the requested memory object length. The service provides the metadata and the memory object, which may be of different length than what was requested. In practise, the credentials are typically attached to the calling process and provided implicitly. The metadata is attached to the memory object so that the calling process cannot mix but match them.

The publisher may fill the memory object with any data it wants, up to the allocated length. Once the object contains the desired data, it can then be published. Only the publisher can create RIds for memory objects (called *MO-RId* in the following), as the RId is created as a result of the first publication. The memory object may be published several times, either with a different or the same tag, resulting in the same or a different MO-RId. If the memory object is re-published with the same tag, it is considered to create a new version of the same publication. Publishing with a new tag is considered a new publication, instead.

```
  MO-RId := create_RId(credentials.public, tag)
```

For creating a MO-RId without actually publishing, anyone knowing the public credentials (public key) of the publisher and a suitable tag can create an MO-RId that matches one that will result from using the full credentials and the same tag.

```
  event-stream := subscribe(SId, MO-RId)
  unsubscribe(event-stream)
```

A subscriber subscribes to an MO-RId received from a publisher. The same SId that was used in the publication is expected to be used.

As a result of the subscription, the subscriber receives an event stream. In practise, the event stream might be represented, e.g., as an open file descriptor that can be used in select() and other similar system calls.

```
  memory-object[length] := receive-notification(event-stream)
```

Once there is an active subscription, every time the local node learns that a publisher has published with the same MO-RId, the subscriber is notified and receives a new memory-object, representing the corresponding version of the object.

```
  event-bitmap := receive-notification-delta(event-stream, memory-object)
  update(memory-object, event-bitmap)
```

As an optimisation, the subscriber may opt to receive only a bitmap where there is one bit representing each page that has changed since the previous version of the memory-object. The update() call updates the corresponding pages in the memory object. The subscriber may turn some of the 1 bits to 0 prior to calling update(), causing only some of the changed pages to be updated. In that way it may, for example, first update the first page of the memory object, and only afterwards the other pages.

Once a process (publisher or subscriber) no longer needs a memory object, it should dispose it by using the free() function.

```
   free(memory-object)
```

Emulating lowest-level service

If it is desired to use the lowest-level service but the system does not allow direct sending to FIds, the lowest-level service can be emulated with the memory object service as follows. We assume that the local rendezvous has been configured to publish anything placed on the low-level SId using the given FId.

```
   low-level-SId := get_sid_for_fid(FId)
   MO-RId := new_RId(credentials, tag)
   memory-object := create(credentials, one page)
   do {
     publish(low-level-SId, tag, memory-object)
   } until done
   free(MO-RId)
```

This creates a number of page-sized packets, each having a distinct version number and page RId, corresponding to the contents.

At the subscriber end, the subscriber executes the following:

```
   low-level-SId := get_sid_for_fid(FId)
   event-stream := subscribe(low-level-SId, MO-RId)
   do {
     memory-object[one page] := receive-notification(event-stream)
     free(memory-object)
   } until done
   close(event-stream)
```

3.3.3 Channel Model

The presence of publishers and subscribers within the rendezvous system can be interpreted as a communication channel between the publishers and subscribers. This channel can have different meanings to different applications or internal components of the PSIRP network (such as the rendezvous system). For example, a component of the rendezvous system will subscribe to an information network (designated by an SId) to receive publication resolution requests or subscriptions from end applications.

To end applications, these channels may be extremely short-lived (for example representing a single sensor temperature transmission), or represent a longer-lived concept (such as a sequence of temperature readings from the same sensor). Channels that are effectively closed down (when the last publisher or subscriber leaves the network) may be re-opened at a later date with the same or even new semantics.

An individual RId may thus represent many concepts. It may be a multicast channel over which an item of content is delivered. Several RIds may be used together to layer or chunk the content for easier consumption. Alternatively, a RId may be used to receive requests for that item of content (which itself may be considered to be information separate from the content itself). RIds may also represent flow control or error signalling information concerning the receipt of packets by a single subscriber. RIds may also be used to duplicate the channels bound today to network locations (IP addresses), hosts (domain names), or individuals' email inboxes.

The low-level PSIRP service API can already be considered representing a channel model at both the rendezvous and forwarding functions. As discussed, the presence of publishers and subscribers can be considered to form a channel within the rendezvous function. This channel is used to transfer subscriber location information (abstracted to forwarding paths, represented by an FId) to the publishers. If the advertisement methods are used, these form channels for subscriber activity information to be transmitted to potential publishers.

The forwarding path can also be considered to represent a channel instantiation of the publisher/subscriber information within the rendezvous system. This channel then provides the fast forwarding path for data between publishers and subscribers. One problem is how to avoid this forwarding channel becoming stale or out of date when subscribers change. To avoid this, the topology formation function must be revisited to update the forwarding path.

Specific channel services may then be created over the low-level PSIRP API. These specific channel services can provide additional or more specific services such as flow or error control. We can also implement other communication modes such as duplex channels or many-to-many group communication channels from the underlying primitives.

3.3.4 Higher-level Service Models

In this section, we briefly describe a number of higher-level services, utilising the underlying memory object and channel models. The two considered interaction techniques are many-to-one services and concast. The former pertains to the way many clients access a single server, and the latter focuses on combining on-going communications from multiple senders to a single receiver.

Many-to-one Services

There are different ways to implement services where many agents communicate with one (or a small number) of central servers. In this section, we briefly outline three possibilities, two utilising the basic services provided by the PSIRP architecture and one requiring a more active role from some of the forwarding nodes.

It is worth bearing in mind that there appears to be a large difference in terms of resource consumption between supporting N→1 in the rendezvous system compared to the forwarding path. For the rendezvous system, N→1 signalling simply means that we allow multiple publishers to request forwarding paths towards the subscriber; see the usage of scopes below. When the publisher obtains a forwarding path, the path is typically a 1→1 path between the publisher and the single subscriber.

The concept of a N→1 forwarding path does not make much sense for source controlled forwarding (e.g., using the Bloom filter approach described in Section 4.6.1) and doesn't seem to add much in the alternate model where we configure forwarding state in the network. On the other hand, supporting concast-like service may be useful, but their economics is an open question at this point.

Using scopes as a signalling mechanism

Most types of scopes can be used as a rudimentary many-to-many signalling mechanism. To do so, the parties interested in receiving notifications of new communication requests, e.g., the central server(s), need to subscribe to the scope itself. As a result, whenever any other party creates a new publication within the scope, the scope-subscribing parties receive a notification that a new RId has been assigned to the scope. As a result, they can subscribe to the new RId, receiving the new publication. Figure 3.3 outlines the required signalling.

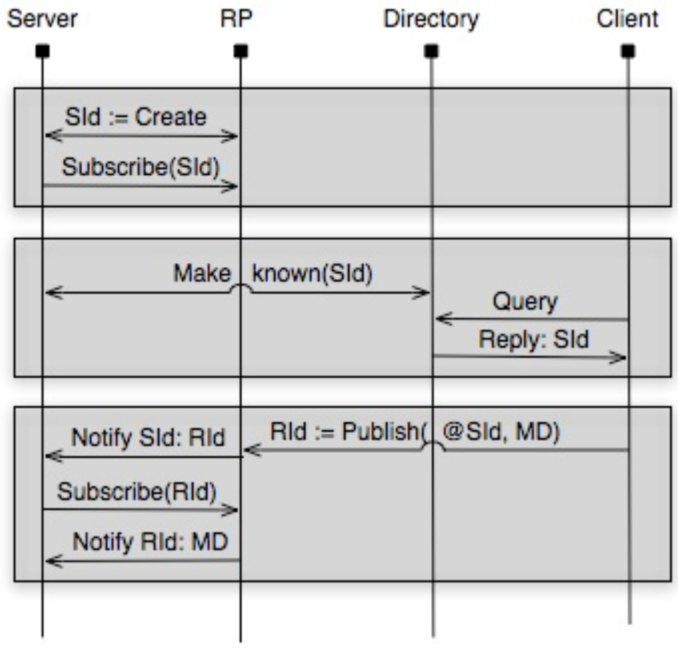

Figure 3.3 - Using a scope to initiate client-server-like communications

Comparing with what happens in the present Internet, one can consider the first two messages as corresponding to a listen() system call or acquiring a DHCP lease in a typical TCP/IP implementation. In the TCP/IP case, a listen() involves no communication outside of the node; on the other hand, a typical DHCP request/reply involves at least two messages. The second phase of making the SId known and the client acquiring the SId can be compared to registering an IP address at the DNS in the current Internet. The final phase of the client publishing a meta-data document and the server interpreting it can be compared to the connect() system call and TCP SYN exchange.

Inside the network, scopes are implemented by the rendezvous system. As described in Section 3.5.1, rendezvous may be implemented in different ways, from a completely decentralised fashion to a centralised service. Hence, the fine details of how a new scope is created, how a new publication is registered at the rendezvous, or how any parties subscribing to the scope get notified, all depend on how the rendezvous is implemented. Note also that the signalling diagram is simplified in the sense that it collapses the rendezvous point (RP in Figure 3.3) and caching of data. In the typical case, the metadata might not be returned directly by the rendezvous point but instead from either the client or some cache.

Reverse channel setup using invitations

Specific invitation documents can be used to initiate communication. In practise, the party willing to communicate (server) publishes an *invitation* and makes its SId and RId well known. Now, anyone wanting to communicate with the party may subscribe to the RId and receive the invitation[3]. The invitation contains details of how to communicate with the serving party, such as the algorithmic RIds that the party subscribes to.

Inside the network, the rendezvous system is involved multiple times. First, the serving party creates the invitation (memory object or channel) and registers it at a rendezvous service. Then, at a later time, a client subscribes to the invitation, typically by publishing a subscription at a lower level scope and making the lower-level rendezvous to deliver a notification to the upper-level rendezvous. The upper-level rendezvous arranges the client to receive either a recent copy of the invitation (for document-like invitations) or join the invitation (for channel-like invitations). Finally, depending on the instructions in the invitation, the rendezvous may be

[3] Note that it appears possible to provide invitation-like services both using the memory object and channel models, though the details differ slightly.

involved a few more times in order to create or subscribe to the actual documents or channels involved in the communication.

Figure 3.4 outlines the required signalling for this case. Note that the content of the invitation is left open in this description. We expect algorithmic identifiers or HIP-like puzzles to play a role in the actual invitation, allowing variation over the RIds used by the clients.

Also note that the sequence diagram is simplified, and the actual behaviour is likely more complicated, depending on how the RP will be implemented.

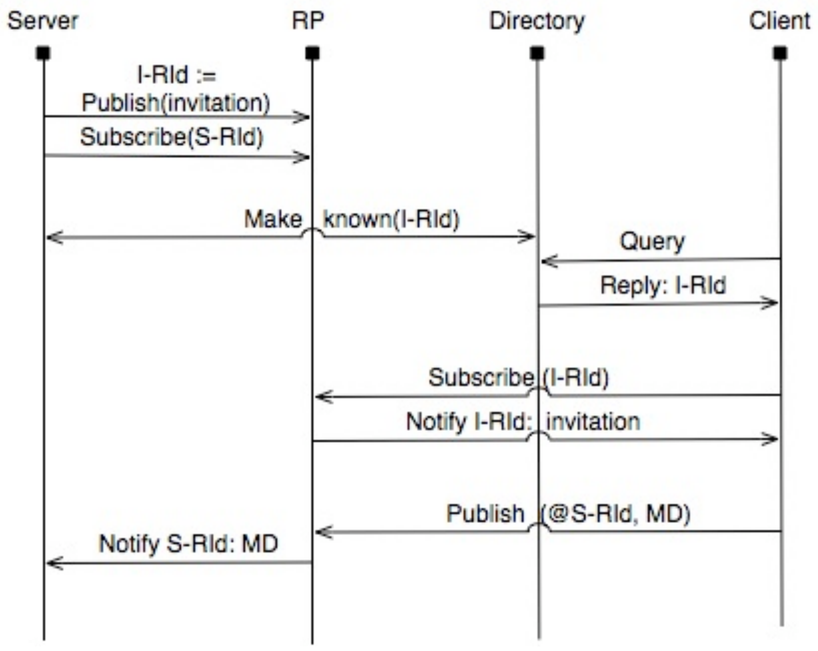

Figure 3.4 – Using an invitation to initiate client-server-like communications

Concast-like services

Finally, we consider the case where merely initiating communication from many points towards a single point is not sufficient, but where there is a genuine need for many parties for continuously sending information towards a single one. We call this service *concast-like*, since it involves active participation by forwarding nodes within the network. At this point, it is open whether such active participation should always be implemented in the slow path, or if there are variants of this service that are frequent enough to warrant fast path implementation.

In the following, we consider the simplest variant where there are two publishers/senders and one subscriber/receiver. The service is assumed to be channel-like; it is an open question if a concast-like service can be modelled on the top of the memory-object service.

The simplest case looks like the following, A and B being the publishers, a and b their respective data streams, CP being the *connecting point* / *concast point*, and a,b or f(a,b) the resulting data stream.

```
A--->>-a--
          \
          CP--->>---a,b..   or   f(a,b)---S
          /
B--->>-b--
```

Here, the CP receives messages from multiple directions and processes them into a single output. If its memory gets full, it may signal upstream to buffer their messages. This kind of tree would protect the bottleneck of the receiver network from DDoS attacks because congestion control in branching nodes would stop messages near the senders. Here the semantics are that the receiver wants to receive all messages sent to him, and the messages are buffered near the senders until received. These kinds of concast points could also be used to perform some function on the inputs and then just send the output of the function.

The service requires the CP to be actively involved in processing the messages along traversing towards the tree root. The current expectation is that the operation will take place at a separate processor at a forwarding node and not as a part of the fast-path processing. The actual forwarding identifiers used at A→CP, B→CP, and CP→S can all be distinct. The rationale is that CP will need state to process the messages anyway, and therefore the state can also include the required FId mappings. That is, at minimum the CP needs to separately keep track of the amount of data arriving from the different sources so that it can request upstream buffering and selectively drop messages in the case an upstream does not comply. Using separate FIds for each flow from upstream is likely to make this job easier.

The exact signalling for creating the required state is open at the moment, as is all explicit signalling along forwarding paths. However, one potential approach is described in Figure 3.5 below, and can be described as follows.

1. The subscriber *S* requests a concast scope *SC* from a rendezvous service *RS* providing concast services. This results in a scope ID *SC-SId* and a special document *SC-RId* that *S* will use to signal back to *RS*.
2. *S* subscribes to *SC-SId*.
    a. A meta-subscription message, containing *SC-SId*, is passed from *S* to *RS* through the slow path. The slow path accumulates the information needed for determining FIds towards *S*.
3. *S* publishes an empty document at *SC-RId*.
4. A new publisher *A* wants to join the concast group.
    a. *A* publishes a meta-document *MD* at *SC-SId*.
    b. The meta-document gets assigned *MD-RId*.
    c. A meta-meta message, containing *MD-RId*, is passed from *A* to *RS* through the slow path. The slow path accumulates the information needed for determining FIds from *A*.
5. The *RS* gets notified about the meta-meta message. It updates and re-publishes *SC-SId*.
6. *S* gets a notification of *MD-RId*.
7. *S* subscribes to *MD-RId*.
8. *S* gets a notification that it has received the contents of the meta-document *MD*.
9. *S* consults the contents of *MD* and decides to accept *A* as a concast sender.
10. *S* adds information about *A* to the special document and re-publishes it under *SC-RId*. In practise, *S* adds some *RId*, *A-RId*. that it has agreed with *A* through *MD*.
11. *RS* gets notified that the special document *SC-RId* has been updated.
12. *RS* consults the contents of the *SC* document and finds a new RId*, A-RId*.
13. *RS* asks the topology layer to add *A*, as identified by *A-RId*, to be added to the concast group ending at *S*.
14. The topology layer determines a unicast FId from *A* to a CP, using the information collected by rendezvous and the topology itself.
15. The topology layer signals the *CP* to add the FId to the concast tree.

16. The topology layer signals the unicast FId to *A*, allowing *A* to start sending fast-path messages to *CP*.

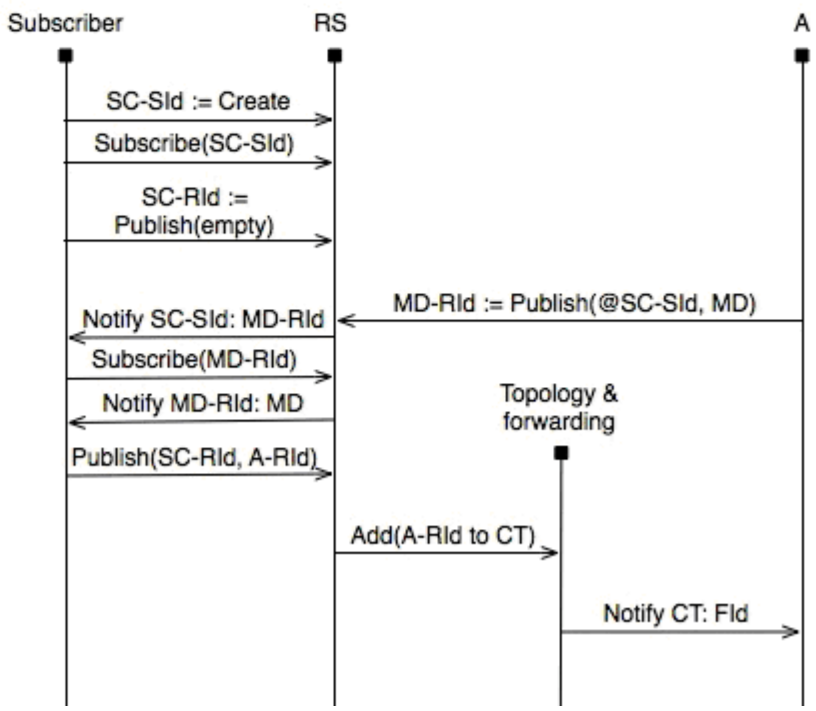

Figure 3.5 – Simplified signalling for adding a node to a concast-like tree

Again, note that the sequence diagram is simplified, and the actual behaviour is likely more complicated, depending on how the RP has been implemented.

3.4 Component Wheel

The PSIRP conceptual architecture is based on a modular and extensible core, called the **PSIRP component wheel**. The architecture does not have the traditional stack or layering of telecommunications systems, but rather components that may be decoupled in space, time, and context. The idea of such a layer-less network stack has been proposed before, for example, in the Haggle architecture [Hag2007]. The novelty of the PSIRP proposal is to use publish/subscribe style interaction throughout the conceptual architecture, and thus support a layer-less and modular protocol organization.

Figure 3.6 presents an outline of the architecture with the PSIRP component wheel. At the edges of the wheel, we have APIs that allow for using different networking features. The figure illustrates a few typical components needed in the wheel for inter-domain operation, such as *forwarding*, *routing*, *rendezvous*, and *caching*. The wheel may also include various other functions that are used to implement additional network features, such as congestion and flow control and reliable data delivery. The “keyhole” at the top of the wheel indicates the open design, i.e., applications may insert new components to the wheel during runtime.

The node architecture (see Section 4.2) is structured around the component wheel. The applications, components (akin to Haggle managers), and the network and disk interfaces are all attached to the local blackboard (BB), at the centre of the component wheel. The blackboard is used to share publications, both data and meta-data, and the pub/sub paradigm is used to signal changes to publications. The blackboard and pub/sub are used to realize a reflective protocol suite that can cope with both static and dynamic changes.

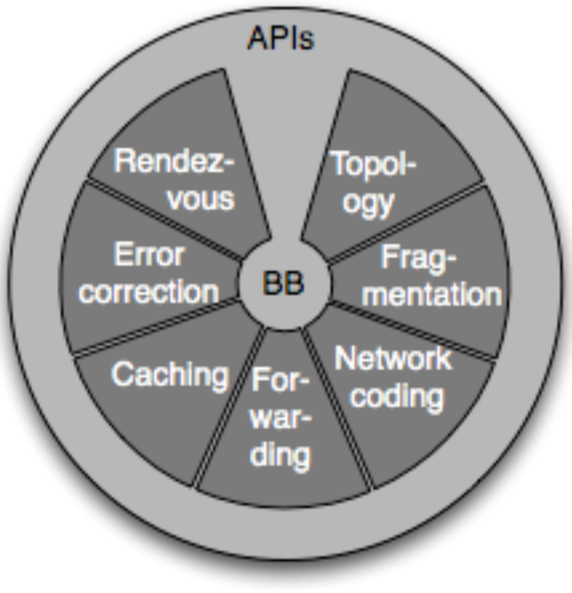

Figure 3.6 – PSIRP component wheel

When a publish operation is invoked by an application, memory for the publication is allocated and subsequently the publication is placed in the local blackboard (see Section 3.3.2). When an application wants to subscribe to some data, it issues a subscribe call. This subscription will create an object that maps to the content received based on the issued subscription. The subscription object can export methods to access the content or it can map directly to system memory or permanent storage.

In the memory object API, the kernel is usually responsible for allocating memory for the publication objects and subscription objects, correspondingly. For a subscription object, the kernel provides the application with a memory object, initially without any content. The actual pages are provided as they are needed or become available. For local IPC, the actual memory pages are shared between the publisher, the black board, and the subscribers. Only if the publisher proceeds to prepare a new version, or a subscriber wants to directly modify the data it has subscribed to, copies of the affected memory pages are created.

The scope and publication metadata may store RIds, acting as pointers to other metadata or data areas. These objects can then be subscribed to by applications or system components.

A local subscriber of a publication can subscribe to receive a notification when a certain event occurs. For example, if a component wants to know when a new publication is added in the local blackboard, it subscribes to notifications about updates to all locally known scopes. As all publications belong to one or more scopes, including the scopes themselves, addition of new scopes can also be detected. To allow bootstrapping (of the local system and the RP) and an ad-hoc mode of operation, a special node-local Scope 0 is always created during the system startup.

All system metadata are potentially visible to all components through the blackboard. Hence, it is easy to add a new component into the system without having to change the basic system calls and the API. For instance, it is possible to create a special caching component that subscribes to all local publications and writes them to some external media.

The component wheel uses the blackboard to maintain an internal execution plan for the components. This plan determines the sequence of components that handle an element added or updated to the blackboard. The plan may allow two components to start processing blackboard data at the same time. The plan is necessary in order for components to act on data in the proper order, for example, first consult the cache and only then perform actual network signalling.

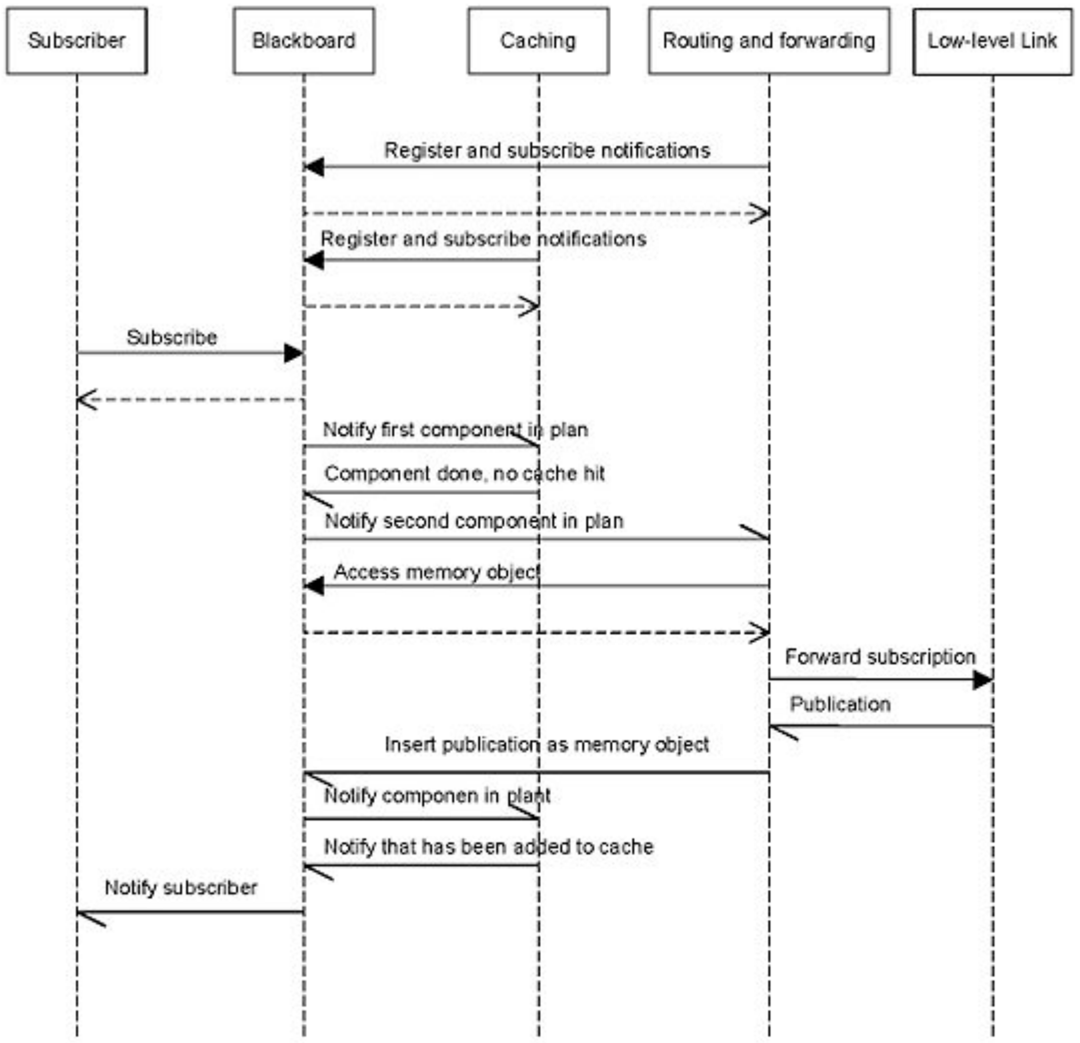
Subscriber
Blackboard
Caching
Routing and forwarding
Low-level Link
Register and subscribe notifications
Register and subscribe notifications
Subscribe
Notify first component in plan
Component done, no cache hit
Notify second component in plan
Access memory object
Forward subscription
Publication
Insert publication as memory object
Notify componen in plant
Notify that has been added to cache
Notify subscriber

Figure 3.7 – Example interactions in the component wheel

Figure 3.7 presents example interactions in the component wheel and highlights how a subscription request may be processed. First, a subscription memory object is created in the blackboard. The component wheel then informs the components about the new subscription according to the component execution plan. In the example, the cache component is first consulted, and then the subscription memory object is handled by the routing and forwarding components, respectively. The subscription is then sent on the network using the low-level API. Later, a publication is received that is associated with the subscription. This results in a notification to the routing and forwarding component and the insertion of the publication as a memory object in the blackboard. The components are again invoked on this object based on the execution plan. The object is added to the cache and then a notification is generated towards the subscriber.

3.5 High-level Overview of the Components

In this section, we present the key components of the PSIRP architecture. As we have discussed, the component wheel is the central element of a single PSIRP node; however, the distribution of information is achieved through topology management, forwarding, and rendezvous. These components are presented in Figure 3.8. The arrows in the figure illustrate information flow during basic data delivery.

The topology function is responsible for mapping, monitoring, and coordinating physical network topology information. The rendezvous implements the function of determining the set of subscribers and publishers for a given publication and instructing the underlying network machinery to perform the needed data delivery. The packet forwarding function is responsible for using the forwarding information provided by the rendezvous and topology functions, and for making forwarding decisions on the forwarding nodes.

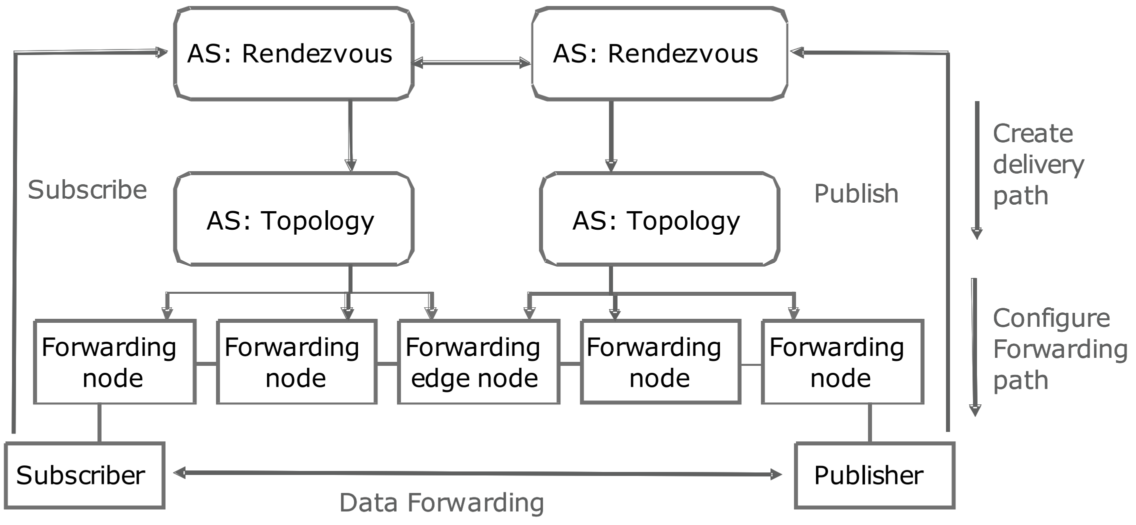

Figure 3.8 – Overview of PSIRP components

3.5.1 Rendezvous

The *routing* function of the PSIRP architecture is divided into several functions. In addition to the fast-path packet *forwarding*, routing is needed for information level control message forwarding that enables the forwarding function. This control plane like functionality in a publish/subscribe architecture is (a) to match the interests of senders and receivers and (b) to form a suitable and policy-compliant inter-domain topology between publishers and subscribers that can be used for fast forwarding. In the PSIRP architecture, the former function is called *rendezvous* and is the focus of this section.

The main objective of the PSIRP *rendezvous system* is to associate subscribers and publishers of a given scope and rendezvous identifier with each other, implementing the PSIRP design principle **PS2** (Information scoping). In this context, rendezvous is inherently related to the notion of *publication scoping* in that a scope (amongst other things) determines the subset of *rendezvous nodes* that are responsible for a given publication and/or subscription. The importance of the services offered by the rendezvous system cannot be understated; rendezvous is crucial in providing a control plane for connecting publishers and subscribers in a policy-compliant fashion both within and between administrative domains.

Within the PSIRP architecture, rendezvous exists in many ways. At the lowest level, the node-local blackboard implements node-local rendezvous. Above that, we expect to have distributed rendezvous over each local link. However, in most cases the rendezvous service is accomplished through specialized functions that operate between physical network devices known as *rendezvous nodes* (RNs). RNs host *rendezvous points* (RPs), which can be viewed as logical, fixed or non-fixed indirection points for pub/sub network communication. Rendezvous nodes within a collaborating set of network domains form *rendezvous networks*. These networks can be viewed as units of rendezvous deployment. Finally, global reachability (i.e., the ability to establish pub/sub-relations in an inter-domain sense) requires *rendezvous network interconnection*, allowing pub/sub signalling to be passed between rendezvous networks.

A refinement of the original end-to-end (E2E) principle by Clark et al. [Cla2007] outlines the importance of **trustworthiness** of execution rather than particular placement of functions in endpoints. This so-called *Trust-to-Trust* principle is captured as our design principle **G1** and considered in our design in the sense that **rendezvous should occur at locations within the network that are trusted to operate correctly in terms of communal, economical, and functional requirements**. Bearing this in mind, the process of determining the optimal network locations for RPs must account for various environmental factors as well as inter-domain traffic policies and pub/sub scoping mechanisms. The specific nature of these concerns is obviously highly implementation dependent, and further research and development of the PSIRP architecture is required in order to arrive at a conclusive set of evaluative criteria. It is expected that the scoping mechanism, with different polices associated with different scopes, may help in achieving a scalable solution.

Due to its importance in policy enforcement and defining (often user-created) information scopes in various situations, the rendezvous system constitutes a relatively well-defined environment where tussle is likely to commence [Cla2002]. The rendezvous system is therefore a policy enforcement point in the architecture and a mechanism for supporting freedom of choice for network end-points. On the abstract level, some aspects of the rendezvous functionality can be found in many existing or proposed distributed systems, for example HIP [Mos2008][Egg2004], IP multicast (RFC2362), i3 [Sto2002] and Hi3 [Nik2004], FARA [Cla2003], PASTRY [Row2001], and HERMES [Pie2004].

Figure 3.9 presents an overview of rendezvous within local, intra-domain, and inter-domain network environments. As discussed in the previous section, the rendezvous entities involved in each of these settings are necessarily determined via scopes (see Section 2). As briefly discussed above, the inter-domain rendezvous is further subdivided to rendezvous within and between rendezvous networks (see Section 4.4).

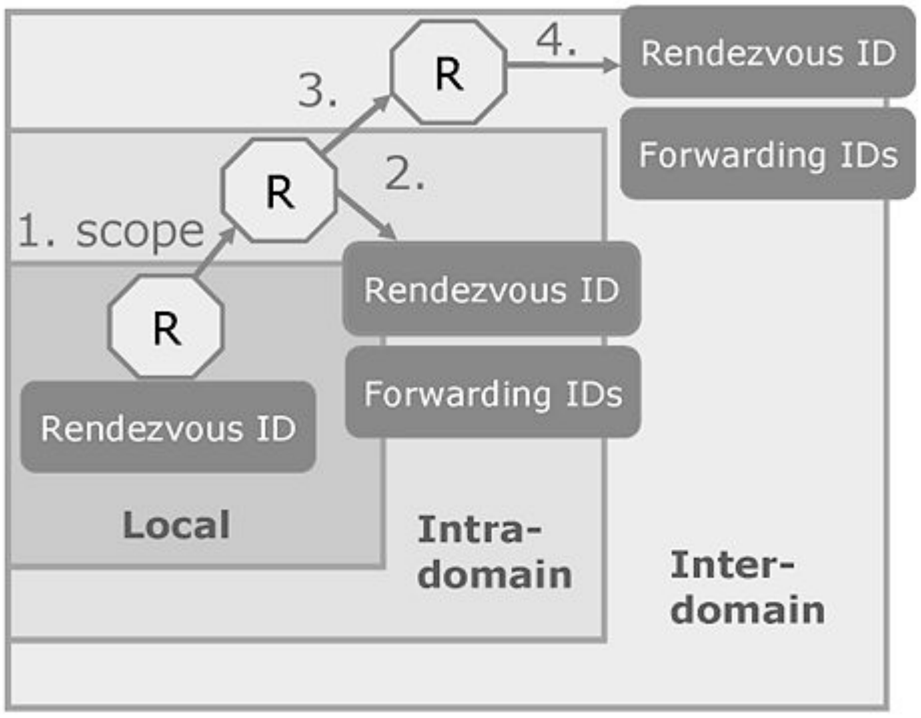

Figure 3.9 – Different levels of rendezvous

A *rendezvous identifier* (RId) is associated with policy-compliant data dissemination graphs for publication delivery, both in the local domain (intra-domain) and between domains (inter-domain). The rendezvous identifiers are chosen from within a large enough set to provide a probabilistic guarantee of uniqueness without requiring a central allocation authority. Applications may resolve *application identifiers*, which are contained within published data, into rendezvous identifiers. It is then the responsibility of the topology formation function (with assistance of the rendezvous function) to find suitable data transit and delivery paths in the network and denote them with *forwarding identifiers* (FIds). This resolution from a rendezvous identifier to a set of forwarding identifiers is based upon the rendezvous identifier in conjunction with scoping (as identified through the scope identifier) and policy mechanisms. The breadth of reference of given FIds is variable, potentially limited to single hops or dynamically expandable to encompass full multicast trees.

Rendezvous state is created to allow subscriptions and publications to meet within a rendezvous node hosting the specified scope. Subscriptions and pre-publication notifications (or *advertisements*) are utilised by the rendezvous system to create partial forwarding state from publishers towards subscribers. When a publication becomes active, i.e., when there is both an active publisher and one or more active subscribers, the rendezvous system instructs the topology function to complete the forwarding path by mapping the rendezvous identifier to intra-domain and inter-domain forwarding identifiers. This late mapping can be used to implement dynamic policies, such as inter-domain routing and caching.

The rendezvous system (in combination with the topology management and formation function, see Section 3.5.3) ensures that neither traffic policies nor publication/subscription policies and scopes are violated. One design goal for the rendezvous system is that the overall network overhead remains within reasonable bounds. The system should dynamically adjust its operational characteristics and control tradeoffs between configuration latency and data delivery overhead.

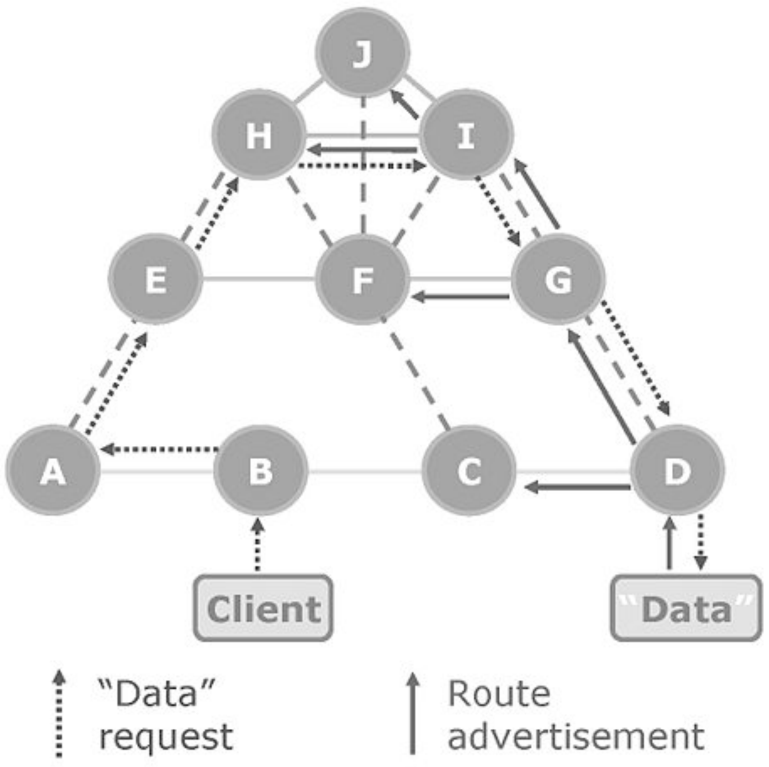

Figure 3.10 – Inter-domain rendezvous within a rendezvous network

In practice, inter-domain traffic policies impose effective limitations over the distribution of rendezvous state between different network domains. Figure 3.10 illustrates a sample inter-domain rendezvous scenario involving a network with ten domains. Those RNs that are selected to handle inter-domain coordination are likely responsible for providing a view of the entire local domain to neighbouring domains. Here, the dashed lines represent transit relationships between domains and solid lines represent peering relationships. In this example, advertised publication routing state is distributed to all immediate peers and providers (solid arrows). Subscriptions are routed to siblings and providers (dashed arrows), and follow advertised publication routing state when encountered (in domain H in this example). This structure expressed in this example denotes one possible inter-domain methodology for rendezvous within a rendezvous network that arises from existing business relationships amongst separate administrative domains.

3.5.2 Forwarding

The packet forwarding function implements the efficient packet delivery through the network. In the PSIRP design, this function is not responsible for locating the publication (implemented by the rendezvous function, see Section 3.5.1) or creating the path from the source to the destination (implemented by the inter-domain topology formation function, see Section 3.5.3). Instead, by using the forwarding information provided by the rendezvous and topology functions, the forwarding function makes decisions on the forwarding nodes. In addition to the forwarding decisions, depending on the selected solution, some additional operations can be considered to be its task, such as switching between different forwarding identifiers.

The forwarding has been divided into two separate areas: *intra-domain* and *inter-domain forwarding*. On a high level, the difference between these different forwarding areas is that in the intra-domain case, the party managing a given network domain is more likely to control all the nodes and information inside the domain. Therefore, the system can base the forwarding decisions on this information. However, in the inter-domain forwarding case, the information that is available from the separate domains may be limited (e.g., the internal structure of the network or local policies for intra-domain forwarding), and also the amount of information that is required for creating forwarding paths end-to-end may exceed the processing capabilities for calculating the forwarding paths. In other words, the division makes it possible to distribute the required path calculation to various parts of the network.

The current results suggest that it is not feasible to use a single forwarding identifier from data location to subscribers, if the complete path (tree) spans several domains, as the per-packet overhead may be high and policy issues may also arise. This problem can be avoided by separating the forwarding into intra- and inter-domain parts. Furthermore, scalability can potentially be improved by recursively introducing additional levels of hierarchy.

Intra-domain Forwarding

The main design target is to provide a forwarding mechanism that does not rely on end-to-end addressing schemes, can be used for multicasting, and does not create too much state information in the forwarding nodes. It is assumed that the size of a single domain is limited, e.g., within a metropolitan area network. This is important when considering the topology management function (see Section 3.5.3) that needs to create the forwarding identifier for each publication. Also, the information that must be placed into the packet header is limited, when compared to the inter-domain forwarding case with its requirement of Internet-wide scalability.

Our initial work on a Bloom filter based forwarding mechanisms (see Section 4.6.1) indicates that Bloom filters can be seen as a suitable and scalable candidate for domains sizes up to metropolitan area networks [Jok2009], supporting also native sparse multicast, without maintaining any state information in the network. For dense multicast, a small amount of state information is required to be maintained in the forwarding nodes.

Inter-domain Forwarding

Solutions for inter-domain forwarding should avoid the maintenance of too much state information in the network nodes. While the most obvious solutions are based on switching the forwarding identifier on the domain borders based on the rendezvous identifier in the packet, the problems arise when the number of simultaneous publication transmissions is very high. Other types of solutions are needed to support a huge amount of simultaneous transmissions, still keeping in mind the requirements for scalability, non end-to-end addressing scheme, and efficient multicast support.

Relation to Rendezvous and Topology Management and Formation

The forwarding function itself is not capable of creating the required forwarding information for different publications. Thus, the forwarding layer relies on the forwarding information created within the topology management and formation function (see Section 3.5.3). This function is capable of determining a suitable forwarding graph, both locally for intra-domain forwarding, as well as globally, for inter-domain forwarding. The forwarding system uses the forwarding identifiers received from this function to efficiently forward the packet from the current data location to the subscriber.

Depending on the solution that will be adopted for the inter-domain forwarding, the forwarding function may handle the packet transmission end-to-end, either using a single forwarding identifier, using multiple, stacked, forwarding identifiers in a single packet, or by switching forwarding identifiers in suitable places. However, it is also possible that the topology function may have to be contacted at some points, if such a solution is found to be suitable during the forthcoming research.

3.5.3 Topology: Management and Formation

The PSIRP architecture assumes a structure of autonomous systems being interconnected via some inter-domain mechanisms, similar to the current Internet. These systems are assumed to be individually governed by organizations, which are responsible for the optimal utilization of their resources. With this in mind, topologies are formed and managed within two different realms, namely at the intra-domain and inter-domain levels. The following describes first the intra-domain management before outlining the inter-domain topology formation.

Intra-domain Topology Management

Within each domain, we intend to enable domain-specific topology management schemes in order to cater to the particular aspects of each involved technology within that domain. For instance, we can foresee an intra-domain topology management that is optimized to a mesh-like operation of wireless nodes that form said domain. Or we can have MPLS- (multi-protocol label switching) like topology management for fixed networking domains. The optimization and organization of the resources required to forward PSIRP primitives and data is handled by a dedicated function, the **topology management** (TM) function, within each domain. This function also holds information about peering relations to other autonomous systems, this information being used in the processes of inter-domain topology formation and forwarding.

Inter-domain Topology Formation

Based on the intra-domain forwarding functionality, the function of **inter-domain topology formation** (ITF) is responsible for constructing a forwarding graph that connects publishers and subscribers residing in different domains. This forwarding graph is used by the inter-domain forwarding function to deliver data between the set of publishers and subscribers. One or more functional ITF components may exist with the potential for creating a market for inter-domain peering relations.

3.5.4 Helper Functions

We use the term *helper functions* to cover the spectrum of various components and software entities that are neither responsible for the core network service, nor can be considered as traditional applications either. Instead they provide additional network (or associated) capabilities. Common examples of helper functions in today's networks would be network management solutions (such as SNMP), diagnostic tools (ping, traceroute, tcpdump), caches, proxies and so on. While not absolutely essential to the operation of the network, they will play a major role in any operational deployment of the network architecture, and should, in our opinion, be already considered during the architecture design phase. Within the node architecture, implementing the component wheel (see Sections 3.4 and 4.2), the helper functions can access the various functions in the wheel through the memory object API (see Section 3.3.2).

The various helper functions that are discussed in more detail in Section 4.3 fall into several categories, depending upon their positioning in the architecture including network management functions, remote service functions, and host service functions. We discuss in particular how these helper functions are foreseen to operate using the basic API and publish/subscribe paradigm. In several cases, we find that the publish/subscribe communication model is very natural for many of the helper functions, such as content encoding and protection or network management. We also discuss in Section 4.3 the relation of the various helper functions running on individual nodes in relation to the blackboard paradigm and the component wheel architecture selected for the implementation.

3.5.5 Network Attachment

In order to communicate in a network, a node must be attached to at least one point, located at the next hop of a data link that provides access to further nodes. In addition, a node must know a set of configuration parameters, including various identifiers as well as, for example, security-related information. The task of setting up, maintaining, and renewing this kind of state is handled by the **network attachment** component. That component handles discovery of attachment points, authentication of nodes and users, their authorization to use network services, as well as providing configuration information to entities involved in the attachment process. On more detailed level, these essential attachment operations can be described as follows:

- Network and attachment point (AP) discovery can provide information on compatibility, identity, services, policies, etc. in addition to attachment initiation parameters. AP selection and attachment procedure bootstrapping are related to this phase.
- Authentication/authorization (possibly mutual) of users and nodes implies verifying their identities and/or access rights. Explicit or implicit contracts regarding compensation for services may be established or checked at attachment time.
- Configuration includes exchanging and setting up identifiers (FId, SId, RId) needed for communication, for example with the nearest forwarding node and with rendezvous points (rendezvous bootstrapping). Another notable task related to configuration is the negotiation of security parameters (e.g., key exchanges, TESLA bootstrapping, etc.).
- Change of attachment points, which is closely related to mobility, requires the initiation of communication with new APs, and possibly re-authentication and re-configuration. States for publications and subscriptions also need to be updated (see Section 5.1).

Section 4.7 provides a more detailed discussion of the network attachment function and component.

4 Architecture Components

While Section 3 presented a high-level overview of the different architectural components, the following section delves deeper into the main technologies and solutions being developed for these components.

4.1 Identifiers

Different identifier concepts are used within the PSIRP architecture, as outlined in Section 2. The following sections present the different identifiers being used within PSIRP.

4.1.1 Application (Level) Identifiers (AId)

Applications can use many concepts including flat and structured identifiers and namespaces to identify relevant objects and information. Such AIds can be used to represent many different concepts from people to network locations and devices, units of information, communication sessions, geographical location etc. Although some standardisation efforts have emerged to consolidate identification and naming schemes, we can expect that applications will continue to use diverse mechanisms to meet their own needs. Application names and identifiers are also likely to follow human readable or meaningful structures, meaning that the utilisation of the identifier space is not uniform.

In the PSIRP architecture, applications are expected to continue to use their own identification and naming schemes. Where such AIds form the basis of communication between subscribers and publishers, they will be resolved into network Rendezvous Identifiers (RIds). Applications will use many different models and services for performing this resolution including search engines and directory services, announcements of new AId->RId mappings over the network and fixed algorithmic mappings of content to RIds (such as hashes of the content or primary attributes).

4.1.2 Rendezvous Identifiers (RId)

Rendezvous Identifiers (RIds) are used within the PSIRP network by the rendezvous functions. Their primary goal is to allow subscribers and publishers to meet within the rendezvous system by using a RId that is known to both parties.

Thus, at the application level a RId represents some meaningful concept about which the publishers and subscribers wish to communicate. For example, a RId may be used to represent a social group (for group communication) or be used to transmit a unique item of information represented by the RId. Some rendezvous systems and transport mechanisms, such as the shared memory object system implemented within the PSIRP testbed, may enforce stricter notions of the meaning of a RId.

4.1.3 Scope Identifiers (SId)

Scope Identifiers (SIds) can be considered a specialised subclass of rendezvous identifiers (RIds) that are used by the rendezvous system of the PSIRP architecture to aggregate other rendezvous identifiers. Publications and subscriptions to a RId will be made within a context of a SId, and hence will be seen by the components of the rendezvous system that have subscribed to that scope identifier. A particular RId can be assigned to different SIds, as also outlined in Section 2. In technical terms, we can observe that both publications and subscriptions to a RId are themselves publications to the SId. This underlying publication to the SId is used to forward the RId publications/subscriptions to the interested rendezvous system components (that have subscribed to the SId).

There are a number of reasons for maintaining a separately controlled scope for each publication:

- To allow the publishers and subscribers to select a scope and to migrate information between scopes such that interests are grouped together. This provides a key benefit to the network as subscriptions may be aggregated into more scalable scopes, particularly for inter-domain management. This also isolates the inter-domain rendezvous system from the churn in RId interest and the resultant signalling traffic.
- To allow aggregate operations on the scope, such as access control and other metadata controlled operations, e.g., defining a scope-level caching policy. This can be used by the applications to manage sets of information in a scalable fashion.
- To allow separate publications to have shared control of the dissemination through the scoping mechanism, yet allowing each publication to be separately authenticated. For example, multiple publishers, all publishing to a single scope, can still be separately identified by the network.

To the application, a scope represents an *information network* (see Section 2), or a collection of whatever information the RIds represent. Through the use of a single scope identifier, such collections can be communicated with the assistance of a single trusted part of the rendezvous system where additional policies may also be enforced.

4.1.4 Forwarding Identifiers (FId)

Forwarding identifiers are used by the forwarding function of the PSIRP architecture to communicate data between publishers, subscribers, and system components. Where RIds and SIds are used by the rendezvous system (on the slow path), FIds are used by the fast forwarding fabric. This enables the operation of a scalable and flexible rendezvous system that just carries the publication and subscription requests. For this rendezvous network achieving high bandwidth, low latencies, or a stretch close to one is not critical.

However, the bulk of the data between the publisher and subscribers is carried directly over an established forwarding path (as a result of previous publication and subscription requests). The forwarding function provides the ability to create efficient forwarding paths over high performance communication nodes for the information transfer. It is expected that the PSIRP forwarding function will also fit well with emerging label switched and optical networks.

There are two models for setting up a forwarding path, presented in the following paragraphs.

Source-controlled packet soft state

In the source-controlled model, the publisher requests the rendezvous system and the topology formation function (see Section 4.5.2) to identify the subscribers and form a domain level (multicast) path to the subscribers. The publisher then uses the encoded path information as a forwarding identifier when sending publications. In this approach, the FIds identify partial distribution trees (or even single links) that can be determined via the forwarding state in the packet header. In the current PSIRP implementation, this approach is realized via Bloom filters which are used to encode the forwarding paths directly into the packet headers.

At the inter-domain topology level, such Fids identify inter-domain adjacencies (conceptual links between autonomous domains), while within a single network they may relate to established multicast trees to existing subscribers, fragments of paths, or even single links.

This approach has the benefit of keeping state out of the network and allowing local control of how publisher’s data is routed over the network. A number of disadvantages are also evident, such as the need for the publisher to receive updated FIds from the rendezvous function in order to identify new network domains that have subscribers or to react to changes in the set of subscribers. Another problem is that the publisher may retain the ability to send data to the subscribers even after they have unsubscribed from the rendezvous system. Thus such forwarding identifiers clearly need to be cycled or refreshed within the network.

In-network state

An alternative approach to forwarding is to configure FId switching state within the forwarding nodes. In this approach the publisher only identifies an initial Fid, which may then be switched to other Fids until the information arrives at subscribers.

The actions of publications and subscriptions on the rendezvous and topology formation functions are used to configure forwarding paths across the network. FIds are still used to identify path fragments within the networks, but these now serve as a basis of communication between the forwarding nodes and the topology formation functions.

4.1.5 Algorithmic Identifiers (AlgIDs)

Algorithmic Identifiers are defined as identifiers that have been created automatically by a function from an existing identifier or identifiers. This allow for implementing the concept of *information collections*, as introduced in Section 2, i.e., a collection of semantically similar information items (the semantic similarity being represented by the algorithmic determination of the individual Ids).

This algorithmic relationship enables tests to be performed to detect relationships between two or more identifiers. Such relationships may include:

- **Ordering**. Does one identifier (immediately) precede another, for example for a sequence of information? This can also be used to determine if an identifier is within a range of information objects.
- **Composition**. Is one identifier contained within a set specified by another (for example to determine whether a fragment of information belongs to a larger item of content)? Such sets may be exclusive (an item cannot appear in more than one set) or non-exclusive.
- **Completeness**. Do we have all the identifiers that compose another information object (which may or may not have its own identifier)?

Furthermore, given one or more identifiers we may also wish to automatically generate the identifiers for related information in order, for example, to subscribe automatically to extra information. There are many potential applications for this, including automatically deriving identifiers for information fragments, calculating the next identifier for a sequence of sensor events, deducing the identifier used for flow control information etc.

Obviously, the network cannot prevent applications from generating their RIds and SIds according to various functions. This edge host processing of algorithmic identifiers does not concern the network, which will just see a series of seemingly unrelated identifiers. However there are also a number of potentially useful functions that the network or transport functions can provide if they can generate and test relationships between identifiers. The next subsections present examples for these potentially useful functions.

Subscription Management

Information may be arranged as the leaves of a tree, with upper tree nodes representing collections of information. Applications may subscribe to higher identifiers and be automatically subscribed (by the transport or network) to the underlying collection of identifiers over which the information is transmitted. We may require the capability to express a certain range within a larger set of information, such as a range of sensor events. The overall concept is very similar to the use of hierarchical topics within a topic-based pub/sub middleware.

Subscription Aggregation

Some subscriber(s) may be interested in sets of information. Instead of maintaining multiple identifier subscriptions, the transport or network may produce a new identifier that represents this collection. If such operations are performed by the application or transport function, the subordinate identifiers are effectively tunnelled over the network. However, it is also possible that only the subscriber network participates in the process. In this case, we would require an ability to determine whether an identifier is contained within the subscriber interest set.

Forwarding State Aggregation

Similar to subscription aggregation, algorithmic identifiers may also perform a role in the forwarding function. Sets of FIds composing a more substantial path may be aggregated into a single algId. Such aggregate forwarding paths may be used in the network forwarding state or in soft-state headers. The use of Bloom filters to specify a forwarding path (see Section 4.6.1) can be considered as an *algorithmic forwarding identifier*.

Caching

To perform effective caching we must identify useful chunks of information. For example, it is probably of little use to collect a few frames of video without the associated meta-information, or at least it may be more useful to retain complete frames than frame deltas in the case a cache needs to perform selective dropping. The cache therefore needs to be able to identify a complete useful set of information to be cached. This could be achieved if such a set were identified by an identifier and the cache was aware of how many related identifiers were children of such a set identifier. For example, an instruction could be sent along with an item of information that the identifiers of related 'siblings' in the useful set are generated using a sequence number 1..N, a function f, and a set identifier S.

Coding

It is possible that the same information can be sent over the network using different encodings. Such encodings may be lossless, preserving the original information, or may transform the information (e.g. compaction to different video bitrates). Such encodings may be identified automatically by generating algIds. An application, that wishes to adapt its bitrate, would therefore be able to automatically generate the algId of the required encoding and subscribe to this new information feed. Alternatively, mobility to a different device with a different screen size or audio capabilities might also result in subscription to different encodings. This approach would also work for layered video, where each layer would be identified with an algId produced from the original information identifier.

Return Path

Many applications may wish to operate in a client-server type of mode. The server would subscribe to a rendezvous identifier to receive requests for information. To return the response, the client also needs to subscribe to an identifier. The identifier used for this return path may be automatically generated as an algId. Thus, a request-reply transport layer might automatically subscribe to the reply algId before sending the request on the original identifier.

Flow Control

In the discussion on coding, we have already touched on how an application (or transport) might adapt its receiving rate by subscribing to different compactions of the information with corresponding algIds. However, algIds could also be used to perform flow control requests without alteration of the information. A simple example is that the information could be sent at different rates using different algIds. Alternatively, an algId could be used for signalling information to control the sender rate (like TCP). Any application wishing to adapt the rate for a particular rendezvous identifier would send requests to the algId automatically created for such a purpose.

Content Fragmentation

Fragments of content (such as BitTorrent pieces) may be sent by using algIds. Any application wishing to receive a complete item of information can generate and subscribe to the identifiers for each fragment. Such fragmentation can also be structured semantically – e.g. voice, video, biography, trailer etc. components of a movie.

Sequence Numbering

Any application wishing to produce a sequence of information items may use algIds for each item in the sequence produced from a sequence identifier. For example, the temperature reading from a sensor may be identified by a single identifier. Each separate reading is then allocated an automatically generated algId. Any application wishing to follow the sequence must adapt its subscription ready to receive the next item in the series. This allows previous items to be repeated without burdening applications that have already received them.

Error Control

Similarly to flow control, an algId can be automatically generated for any application that wishes to receive network delivery errors associated with another information identifier.

Reliability

Using an algId for error notification as described above can allow a sending application to receive delivery errors and retransmit information. Alternatively, a reliability feedback channel can be automatically generated, similar to the flow control signalling channel discussed earlier. Any application that detects lost information can request that the information is resent using this channel. Separate algIds may also be generated to transmit logs of the information that is being sent over other identifiers so that applications can detect missing deliveries.

Announcements

Prior to sending information, announcements may be sent over corresponding algIds. These announcement channels may carry announcements for a variety of other identifiers. Thus, an application can subscribe to few announcement algIds, which cover their broad information interests. When an announcement is received they can then join the correct rendezvous identifier to receive the information, reducing the average subscription state in the network and allowing receivers to pick and choose which information they receive over a rendezvous identifier.

Designing for algIds

We can express the general design of algorithmic identifiers as a **Directed Acyclic Graph** (DAG). Identifiers are related to each other through a function that forms the edges of the graph. A graph can use different functions provided that an identifier does not have an incoming edge for multiple separate functions. Multiple incoming edges into a single identifier node mean that the identifier was created using a composite function over all of the parent identifiers. It is clear that the functions used within the graph can be used to automatically create the graph of identifiers from an initial identifier set. It is, of course, also possible to test relationships within a graph.

Although graphs are only normally partially ordered, by applying an ordering to the functions used in the graph it is possible to fully order the identifiers. For example the same function may be used to produce a set of children using a branch sequence number as part of the function. The DAG representation allows (sub)graphs to be constructed that represent sequences and sets of information.

Some specialised structures that may be created include:

- Trees produced through the application of *n* (potentially infinite) one-way functions from an initial identifier. Sequences can be expressed as the recursive application of a singular function, or as the children of a single identifier ordered by the function ordering. Since no identifier has multiple parents a reversible function can allow the tree to be navigated in both directions, providing function information is maintained alongside the child identifier.

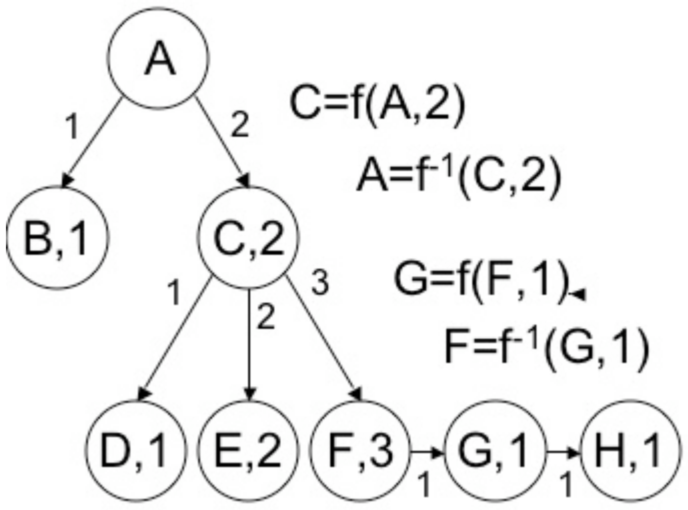

Figure 4.1 – 2-way tree

- Reverse trees created from the application of a composite function over a number of initial identifiers. Since information is lost in the aggregation of multiple parent identifiers, no reverse function exists.

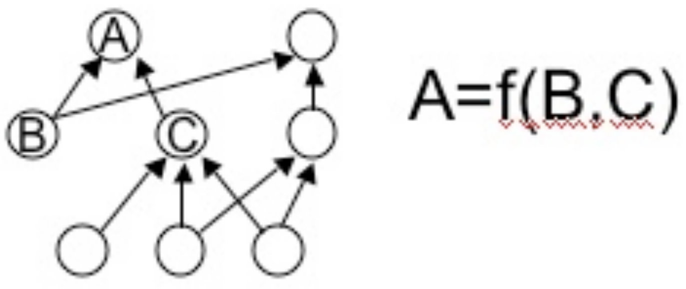

Figure 4.2 – Reverse tree

- Sequences created from the recursive application of a one-way function. The use of a reverse function can allow the generation and navigation of identifiers in both directions. This structure can be considered as a specialised case of the tree.

The ultimate question remains which capabilities the network should support, and thereby which functions (for the production and testing of AlgIDs) should be implemented by different architecture components. Although different network components could use their own specialised AlgID functions, we have shown that it is feasible to consider schemes where the identifiers can be produced and tested by multiple components sharing a common scheme (or schemes).

An alternative to AlgIDs: Relationship Codes

Although we have discussed that an AlgId is algorithmically generated from one or more other identifiers (such as RIds), an alternative approach exists where the identifiers themselves are not related. Instead we can take an approach of attaching relationship codes (for example within the packet header). Such a relationship code, in its most basic form might be a list of parent and child identifiers within the graph. To save space these identifier lists may be aggregated within a bloom filter. This approach allows us to test parent-child relationships. Although some false positives will result from the use of the bloom filter, the result can be tested in both directions reducing the chance of error. Although parent-child relationships may be tested by these means, there remain open questions on how to test ordering or sibling relationships (without the parent). The use of Bloom filters would not allow the generation of related identifiers, but only the testing of identifiers (and relationship codes) that we already hold. In this respect, it is similar to the reverse tree method described above.

One advantage of this approach is that such relationship identifiers can be applied after the RIds or other identifiers have already been chosen and are in use by the application, since the identifiers themselves remain unchanged.

4.2 Node-internal Architecture

The node architecture constitutes an important element in the overall PSIRP system architecture. It serves as a kind of nucleus from which the distributed functions of the architecture, e.g., the rendezvous and forwarding system, is built up. For this, the architecture applies the main PSIRP concepts, e.g., information items within scopes of information, applying the component wheel structure as described in Section 3.4.

4.2.1 Blackboard Approach

The node architecture employs a *blackboard* approach in order to implement the communications of components within the component wheel. In addition, the notion of local *helper functions* (see Section 3.5.4) is used to implement the necessary components for the node functionality. The local blackboard provides a service that is essentially similar to the memory-object service described in Section 3.3.2.

Forwarding to the network via the Blackboard

For communicating with the low-level packet service, each node includes a number of local *output queue* scopes, each corresponding to a single FId (determining a destination for the memory objects placed in the particular queue). We recognize that there may be some access control to these low-level scopes although the exact nature of such access control is for further study.

Whenever a process places a new object in any of these scopes at the node-local blackboard, a *sender-helper* function inspects the newly inserted object. If it fits into one packet, the sender helper forms a packet containing the object, adding a packet-level RId by computing a hash over the content. After that, the sender-helper sends the packet out with the FId. Other header fields beyond the FId and packet-level RId needed are a partially open question; see below. At the same time of sending the packet out, it should most probably also be published in the local "opportunistic cache" scope.

Receiving from the network via the Blackboard

At the receiving end, there are *input queue* scopes, each corresponding to a FId. Whenever a node-local *network-receiver* gets a packet, it places it in the "input queue" scope corresponding to the FId, publishing it locally using the packet-level RId that is included in the packet. Additionally, the packet is cached in a local "opportunistic cache" scope, subject to caching policy.

This simple design forms the obvious difficulty of correlating any related packet at the receiving end, as many higher-level publications (memory objects / streams/whatever) can share a single FId. Solving that seems to require a next "layer", with corresponding helper applications, effectively implementing particular service APIs above the memory object API.

4.2.2 Segmenting Memory Objects

To handle larger than packet-sizes objects, we introduce further helper applications and low-level scopes. These are called *packetiser-helpers* as well as *output* and *input places* (as opposed to "queue" scopes, used for the actual sending and receiving.)

At each node, there is a number of local *output place* scopes. Whenever any process places a new object or updates an existing object in any of these, the *packetiser-helper* inspects the object. If it is a new object, the packetiser-helper chunks it into packets and places each of the packets separately at the right *output queue* scope. It is important to note that this causes no data copying, as the packetiser-helper simply maps its input object into its memory, and publishes each of the pages separately, merely causing a "directory entry" to appear in the "output queue" scope. If it is an updated object, the packetiser-helper determines which pages have changed, and places only those to the output queue.

At the receiving end, the *depacketiser-helper* needs to be able to reconstruct the larger and/or updated object. For that, it needs information from the packetiser-helper at the sender end. For this, the packetiser-helper constructs a meta-data object, containing the packet-RIds of the individual packets, and publishes it in the same output queue scope, causing it to be sent to the receiver, making it visible to the depacketiser. With this, the packetiser can pick the right packets from the "input queue" scope, re-arrange them in a continuous memory area (this may need a new system call), and eventually republish it in the "input place" scope.

Similarly, if there is an update to an object, the packetiser will update the meta-data object, republish it, eventually triggering the depacketiser to get the new version, allowing it to construct the new version of the object, and republish it. This meta-data object is also needed for the next layer, e.g., for minimal error correction, although this is out of scope for this discussion.

Given this design, it would be sufficient for all of the packets to contain just the FId and packet-RId. The metadata packet would contain the "object" RId, and any scopes beyond the FId-related scoping. The "object" RId would allow the depacketiser to recognise new versions over old objects. However, at this time we are not sure yet that these are the right design choices, as there are open issues related to resilience and security. Furthermore, an investigation of the usage of algorithmic identifiers (see Section 4.1.5) is necessary in the context of segmentation.

In a practical implementation, the packetiser-helpers and network-sender/receiver helpers are combined. However, it is useful to realize them as being separate logical entities, as discussed above.

Potential implementation details

At the node-local level, each publication of a memory-object results in events being queued to all processes that have subscribed to the memory-object. A new subscription to an existing memory-object results in an immediate event to be queued, allowing the subscribing process to receive the currently cached version immediately.

At the network-level, the service is defined in terms of the lowest-level service, described in Section 3.3.1. Each publication of a memory-object results in a number of calls to the packet--level forwarding service, once for each page. Furthermore, if the memory-object contains more than one page, a meta-data summary page will be forwarded.

In pseudo-code, the service is defined as follows:

```
create(requested-length) := {
  metapage, memory-object := allocate(1 page + requested length)
  metapage.mo_len  := length(memory-object)
  metapage.version := 0
  return pair of <metapage, memory-object[metapage.mo_len]>
}
```

```
publish(SId, MO-RId, pair of <metapage, memory-object> ) := {
  metapage.version      += 1
  metapage.delta-bitmap := changed_pages_as_bitmap(memory-object)

  for each FId active for this publication {
    for each page in metapage.delta-bitmap
      metadata = { type=page, MO-RId, version=metapage.version,
      number=page.number }
      forward(FId, in-packet-meta-data = metadata, page)
    if (metapage.mo_len > 1 page)
      metadata = { type=meta, MO-RId, version=metapage.version,
      number=#pages }
      forward(FId, in-packet-meta-data = metadata, metapage)
    }
  }
}
```

At the receiver end, the operation of the packet collector helper can be defined in pseudo-code as follows

```
receiver-helper(Fid) => {
  stream := listen(FId)
  until stopped do {
    get next < page-RId, in-packet-meta-data, packet > from stream
    find metapage for in-packet-meta-data.MO-RId

    if (in-packet-meta-data.type = meta) {
      if (metapage.version < in-packet-meta-data.version)
        page.delta-bitmap |= metapage.delta-bitmap
        metapage := page
    }

    if (in-packet-meta-data.type = page &&
        in-packet-meta-data.version = metapage.version)
      clear the corresponding bit at metapage.delta-bitmap

    if (metapage.delta-bitmap is empty)
      publish(SId, RId, memory-object)
  }
}
```

Note that the pseudo-code above specifies only node-local behaviour. In particular, it gives no guarantee that an act of publication would trigger any events at any other nodes, other than the local node. The details of how such memory objects are exchanged between nodes, and therefore on the network service semantics, depends on the upper level services.

4.3 Helper Functions

We define helper functions to cover the spectrum of various components and software entities that are neither responsible for the core network service, nor can be considered as traditional applications either. Instead they provide additional network (or associated) capabilities.

As such, all helper functions must be capable of operating using the basic network API and publish/subscribe paradigm. Helper functions fall into several categories, depending upon their positioning in the architecture:

- **Network management functions**: These functions provide information about, and control of, the various network elements from forwarding nodes to rendezvous points.
- **Remote service functions**: These functions are not truly in the network, since they are, by definition, not required for basic network communication. Examples of such service functions are encoding or caching where the end host applications wish to make use of local supporting services located at suitable points throughout the network.
- **Host service functions**: These functions are network support functions operating on the same hosts as the application. Under this definition network *transport* functions or other network middleware can be considered as helper functions along with traditional utilities such as ping or traceroute. Some of these functions may require capabilities from the network, such as marking or they may use network management functions to obtain network information.

4.3.1 Network Management Functions

Important use cases of helper functions are network management and network information gathering, since incomplete or inaccurate understanding of the network hinders planning and management and might accordingly lead to suboptimal performance. Management functions vary widely in nature, ranging from gathering of information about present network resources, to their allocation and configuration, performing upgrades, and dynamical adaptation and optimization of network parameters for maximum performance.

There are various ways of collecting required data. In the current Internet the most prevalent is the manager/agent model used in SNMP (Simple Network Management Protocol) where the role of the manager is to provide interface between human and network management system while the agent provides connection between management system and physical device being monitored. SNMP uses small set of messages for information exchange (GET, GET-NEXT, SET, GET-RESPONSE and TRAP). The managed system generates a TRAP message as an asynchronous notification of an occurred event. In addition to SNMP, additional protocols (such as ICMP) are in use to obtain connectivity information roughly along paths between endpoints, although the granularity of the information obtained is severely hindered by modern traffic engineering practices and limitations in the underlying protocols.

We believe that addressing network management as an integral part of the architecture effort is warranted, and are thus at present actively exploring the related solution space as part of the architecture work. Many of the problems in existing network architectures are caused by having management functions built in as afterthoughts, and not being really integrated into the original design.

The pub/sub networking, in the form of the PSIRP service model, provides for a much more natural basis for network management applications than traditional endpoint-centric mechanisms. An architecture for gathering network information, which closely resembles the publish/subscribe paradigm, is the Unified Link Layer API (ULLA) [Soo2008] developed as part of the GOLLUM project. ULLA introduces an interface for collecting link layer information using flexible query and notification mechanism. Link aware applications express their interests by specifying various link conditions in the form of a lightweight subset of SQL. Each SQL query contains different link attributes as well as thresholds for notifications, of which applications want to be informed. The notification mechanism supports both event-driven and periodic reports. The ULLA work can be used as a basis for some of the network management related PSIRP helper functions in the sense that (helper) applications interested in specific network conditions “subscribe” to them by making a lightweight SQL query, defining these network parameters and thresholds of which they want to be notified. This subscription can be mapped in the usual way to a unique RId, using an algorithmic mechanism for generating IDs. Algorithmic IDs can also be used to provide access to the synchronous query mechanism of ULLA, essentially by mapping the relevant information structures to the blackboard of the node, and standardizing the way these are assigned identifiers based on the structure of the underlying physical network. For queries, once the specified conditions are fulfilled, notification is sent back to the application. In this way, the application is informed only about the events of interest, saving time, traffic and processing resources compared to the simple case of notifying about all possible values related to the network attribute of interest.

4.3.2 Remote Service Functions

Segmentation

A possible application of helper functions is in the case of transferring large amount of data over the network. Due to the different conditions and properties of links over the path, various data segmentation patterns could be applied. In this case, assuming that the forwarding path from publisher to subscriber is known, an essential piece of information needed for efficient transmission is the MTU size over the entire path. Following the approach above, the MTU helper function would subscribe to the MTU size information. This information can be used as one of the inputs for the segmentation helper function, which can command final segmentation of data. The segmentation helper function can subscribe to data that has to be transmitted and based on the information it gets from the MTU helper function, it publishes data in the form of segmented parts.

Similar functionality can be provided at the subscriber side, as well as intermediate nodes where the merge helper function can perform merging of data if needed. The merging helper function subscribes to data, that has to be received or forwarded, and based on information received from MTU helper function, it publishes the merged data.

We can also consider another monolithic approach to the segmentation helper function implementation. To avoid the helper function partitioning into a large number of separated components, all data needed for the segmentation decision could be collected by the segmentation function itself, without introducing additional helper modules. Here, we would face the trade-off between complexity of this solution and its compactness and independence. On the other hand, breaking the helper functionality into small modules adds flexibility and reusability to the system.

FEC/Content protection

Another potentially important role of helper functions is in error control of data transmission. In order to be able to recover partially corrupt messages at the receiver side without retransmissions, the sender can add redundant data to its message (error correction code). Upon receiving a message containing redundant data, the receiver is able to discover possible errors and to correct them. In this way, a back channel is not needed, and the probability of correct data transport without retransmissions is higher.

Many coding techniques have been developed for transforming messages in a form that improves the probability of reception and correct decoding, starting from more traditional Reed Solomon block code (N,K) which adds N-K symbol redundancy so that after receiving any K symbols out of N sent, the receiver is able to decode to rate-less codes which do not introduce a limitation of the number of encoded packets generated and apply various algorithms over different sets of message bits.

However, the static determination of FEC mechanisms to be used as well as its code size and other properties may degrade its performance due to inaccuracy in the estimation of the underlying channel state, and possible variations of environment characteristics, e.g., changes in BER over time may lead to inefficient bandwidth usage due to fixed FEC schemes applied. In this sense, a FEC helper function might yield the ideal way to bypass this problem by collecting data necessary for decision on the appropriate FEC mechanism to be used in the same way as in the segmentation helper function. A FEC helper function can subscribe to data on which FEC coding has to be applied, and based on the current environment information it gets, it applies a FEC scheme with the most appropriate parameters. Such an operation might be especially appropriate near wireless edges of the network to provide additional protection over lossy segments only.

Content re-encoding

As an example of a remote service function we shall examine how content encoding can be supported by the PSIRP network and provided by a service provider. In this example a host wishes to send multicast video to a number of subscribers. However, some of these subscribers require the video to be encoded at a lower bit-rate and resolution.

The application could choose to send both pre-encoded streams, but to improve network utilisation it decides to send only the higher rate stream and employ remote services to perform the transformation to the lower rate encoding.

The main problem is the selection of an encoding service at a suitable network location and the dynamic management of such service selections as the current subscriber base changes. One model of how this can be achieved is for the encoding service to subscribe to an identifier that represents the particular service in that location. Although geographic location can be obtained by the service using its host information (either through a GPS device or manually configured location), this will only provide an approximation of network locality (and perhaps a poor one if we consider today’s interspersed mobility networks). Instead we require that the network attachment process provides a network location (such as the forwarding network identity) to the host. To improve the selection of location dependent services the network attachment could also provide the identities of neighbouring networks.

Using the network location/identification information the encoding service in our example would subscribe to an identifier that represents “requests for encoding services in network X”, along with perhaps a further subscription to “requests for encoding services adjacent to network Y”.

The video streaming server will start by selecting identifiers for the various items of video it has available. It will then produce an associated identifier on which to receive requests for this item, and subscribe to these identifiers. These identifiers will be announced to potential clients through a range of announcement channels, directory and search services, which we will not discuss in detail in this section.

A potential video subscriber will identify which items of video it is interested in receiving and hence select the identifier on which to send its request. Within the request the client will include the network information obtained from the network attachment process, along with details about the desired encoding.

The video server receives the client request and in turn sends a request for video encoding services in the network (proximity) of the client. It now has the option of directing the client to one of the available encoding services, or providing a direct stream from the video server. In our scenario the server decides to utilise one of the available encoding services close to the subscriber to convert an existing higher rate stream from previous clients. To do so, it instructs the encoding service to subscribe to the existing stream, and create a new identifier for the video content on the lower bitrate. This identifier is returned by the service to the video server, which in turn passes it onto the client.

The client may now subscribe to the identifier representing the re-encoded content from the local encoding service.

Application-specific caches

Specific caches can offer interfaces that match those of the original content providing application. For example they may offer the same search facilities, or the same flow control protocol.

The use of a local cache must be considered in two parts. Firstly the content must be delivered to the cache through the actions of an initial client obtaining some content. Secondly a subsequent client must be able to obtain the content from the cache instead of the original content provider. In both phases the application must be able to identify cache services near the client in the network, since the reason for using a cache is to obtain content locally (for reasons of performance or cost).

Similarly to the local content re-encoding scenario described above, each cache service can subscribe to identifiers related to the service they offer within, or in proximity to, forwarding networks. These identifiers can then be used by either the content server or the client to instruct the cache to participate in the content delivery.

The process starts with the client discovering which identifiers to use to obtain the content. Such identifiers may relate to the content provided by a single server, or may be used by multiple servers to subscribe to receive content requests. In the latter case, a decision may be made by the client as to where to obtain the content (perhaps based upon the locality of the content provider). Alternatively, such a selection process may be performed by the cache upon instruction by the client.

For applications where the client controls the cache selection, the client will send a request for local cache services to the identifier related to such services within its local network. It will then get responses from services that are willing to participate and may select the cache preferentially (based on response times, price etc.). The client will then communicate to the chosen cache the identifier, or identifiers, to use to obtain the content from the content host(s) (or previous caches). The content may be multicast simultaneously to both the cache and the client.

In a model where the content server selects the use of the cache, the client will communicate with the content server (including information about its network locality). The content server will then select the cache service and instruct it to subscribe to receive the content. The content will then be multicast to both the cache and the client simultaneously.

It is also possible to arrange the cache such that the content is delivered through the cache instead of multicast simultaneously to both the cache and the client. This model is likely to be worse in terms of performance, but may offer economic incentives (such as the cache provider being allowed to insert advertisements in return for its services).

Once the cache has received a copy of the content item, it may then register as a content provider for that item.

4.3.3 Host Service Functions

Service functions on the host include transport and other application middleware network services. For example, the current PSIRP implementation provides a shared blackboard metaphor for distributed applications using the network. Other host services provide local tools and utilities to help with the usage and management of the network. Traditionally, some of these tools have been provided with the host operating system such as ping or traceroute, along with diagnostic functions such as tcpdump or wireshark.

All of these services can utilise the PSIRP network API, and potentially also utilise other remote services (such as network management services). They may provide a programming interface (API) to the applications using legacy application integration technology (such as JMS or a Windows API), or may alternatively utilise the PSIRP network. In this latter approach the local host implementation of the PSIRP architecture can provide a publish/subscribe network internally to the host for component and application integration, directly replacing technologies such as JMS or CORBA Notification Services.

As an example we can consider a transport component that provides a particular rate control and reliability algorithm for point-to-point delivery (similar to TCP). This transport component subscribes to one or more identifiers over which it receives information management requests from a number of local applications. The component would be instructed by an application to subscribe to certain information (from the local application) and make those information items available on a different set of identifiers. This second set of identifiers will be the ones shared by the application to potential remote subscribers.

4.4 Rendezvous

The main purpose of the rendezvous functionality in the PSIRP architecture is to provide pub/sub signalling reachability from publishers and subscribers to data or service specific *rendezvous points*.

The main requirements for this functionality are:

1. Scalability to Internet-like networks and data space sizes,
2. efficiency of operation, measured in signalling overhead and overall latency, and
3. deployability.

The first two requirements are usually considered for internetworking designs, but in many cases the third requirement, deployability, is often not explicitly addressed. The main considerations for deployability are incremental deployment and stakeholder incentives. The incremental deployment as a target has a direct impact on the architecture, as the architecture needs to be composed of modules, or parts that are individually deployable, and also *networkable*. Stakeholder incentive considerations tell us that parties (e.g., operators) will invest in (rendezvous) functionality only if it is in their economical interest to do so, and that the scope of their investment is likely limited by their ability to collect revenue based on the publish/subscribe traffic. This guides the architecture to avoid assumptions on globally centralized solutions, when it is not foreseeable that such solutions are economically feasible.

4.4.1 Rendezvous Networks

Rendezvous networks are units of deployment for the PSIRP rendezvous functionality. The rendezvous networks are formed by *rendezvous nodes* (RNs), organized as a BGP-like inter-domain hierarchy (Figure 3.10). In some cases big, geographically dispersed autonomous systems (ASes) may benefit from splitting their network to multiple such rendezvous networks, but in the typical case the rendezvous network would be a collection of rendezvous nodes from multiple cooperating ASes.

Likewise, smaller ASes may find it sufficient to operate one rendezvous node, but bigger domains will need to operate multiple rendezvous nodes in their rendezvous networks.

Rendezvous points (RPs) are logical meeting places of the pub/sub system. There is one (potentially distributed) RP for each <scope identifier, rendezvous identifier> pair. However, typically most of the RP-functionality can be shared between publications in the same scope. This is the mechanism by which e.g. scope level access policies can be shared by all rendezvous identifiers within the scope. RPs are hosted by (one or more) rendezvous nodes.

The interaction model within the rendezvous networks is *BGP-like* in the sense that rendezvous nodes operating over inter-domain boundaries explicitly adhere to the inter-domain traffic policies of the ASes. This translates to *the valley-free routing policy* [Gao2001]: publish messages are sent to all peers and to all provider and sibling ASes, and then recursively to their peers, and providers and siblings. Peers do not forward the publish messages further. This enables the corresponding subscribe messages to utilize the default route up the provider hierarchy. This process will result in the corresponding publish-related state to be found at the latest on the rendezvous network root level (Figure 3.10).

The valley-free routing policy leads to optimal policy-compliant paths for the signalling messages within the rendezvous networks. This helps keeping the rendezvous overhead at a minimum level. However, the participating domains may have incentives for other types of policies as well [Raj2008]. Our architecture places no limit on utilizing such non-valley-free policies.

Another rationale for the policy-based rendezvous network structure is the fact that non-operator ASes may offload most of the rendezvous signalling related state management to operators who are in the business of providing inter-domain networking services. This can be seen by considering stub-ASes (autonomous systems with no transit customers): They are only required to maintain state related to advertised publications received from their direct peers in the rendezvous network, and this only if they in fact agree to peer also at the rendezvous level.

The bigger the rendezvous network, the more signalling and state management overhead is placed on the root-level rendezvous nodes. This places a natural scaling limit to rendezvous networks. In practice it may be that transit ASes will instruct their biggest non-stub customers to form rendezvous networks by themselves, and providing themselves rendezvous service only to their direct stub-AS and smaller non-stub customers.

This structure of rendezvous networks begs the question of how they are connected together to provide global signalling reachability over the whole PSIRP network? This is the topic of the next section.

4.4.2 Rendezvous Network Interconnection

Given the description of rendezvous networks above, the next challenge is the interconnection of such networks to form a globally reachable rendezvous solution. There are multiple potential solutions to this problem. It seems that none of them can be prescribed, as the intricate preferences of the network stakeholders are not known in advance and also develop over time.

The most obvious alternative is that a central entity in the whole global network takes the responsibility of managing rendezvous signalling reachability between all rendezvous networks. However, this option is riddled with challenges relating to e.g. trust, incentives, competition, etc.

Some of these challenges could be addressed by competition of multiple such entities. The providers would compete in global rendezvous reachability coverage and thus would invite all the rendezvous networks to deliver their reachability state to them. Additionally, such competing rendezvous providers could then proceed to exchange rendezvous state between themselves, in practice shifting the interconnection problem up one level. Proactive state exchange between such providers is one possibility, but that again leads to an explosion of the state management overhead.

Finally we observe a possibility for a fully distributed mechanism for rendezvous network interconnection. The participating rendezvous networks self-organize to interconnect without any third party rendezvous infrastructure.

The participating rendezvous networks in such *virtual interconnection overlays* need to share a degree of mutual trust. The level of this trust needs to be the highest amongst the networks interconnecting directly, and may be lower between entities further away from each other. This gives a rise to a virtual hierarchy-like structure, where rendezvous networks join into virtual subdomains, which in turn are joined in higher-level subdomains. In the end all participating rendezvous networks are part of the same, overall virtual interconnection structure.

As global mutual trust is out of the question, multiple such interconnection overlays will be necessary in practice. While the virtual structures can fully operate without 3rd party rendezvous infrastructure, it may be beneficial to incorporate the virtual hierarchies into legal entities for contractual reasons. This would decrease the number of needed contracts from the maximum peer-wise number of $O(N^2)$ to something closer to linear, as each participant would need to contract only with the interconnection-related legal entity.

Our basic design criteria for the virtual interconnection structure is to enable efficient on-demand distribution of rendezvous state. This would leave most of the existing scope-related reachability state local to their rendezvous networks, as only a small fraction of all scopes are expected to gain global popularity. We also aim to utilize the expected power-law scope popularity distribution to efficiently utilize caching to keep the average rendezvous routing latency low [Ram2004].

One possible structure for providing the above is the Canonical Chord [Gan2004]. The overlay structure is virtual and all the interaction happens between the root-level rendezvous nodes of the rendezvous networks. The Canon hierarchy provides two important properties for efficiency.

The first of these is *locality*, making the communication within a subdomain stay within that subdomain, not leaking out to the rendezvous networks outside that subdomain. This property hinges on domain-wise interconnection of the Chord rings maintained at the rendezvous network root nodes, and per subdomain-level registration of the scope reachability state. This distribution of the reachability state also provides robustness against node failures.

The second feature is that at each subdomain level, there is a given exit node to the next level for each scope identifier. This makes caching the scope identifier reachability state very efficient, as the other nodes in transit need not cache the returned scope reachability state.

The Canon structure requires the participants to share a common view of the virtual hierarchy. This hierarchy could be informed by the underlying inter-AS topology, which reduces the signalling stretch, but can also be orthogonal to it. This latter option makes it possible to form the structure based on degrees of mutual trust or business interests, regardless of the underlying physical connectivity.

It should be noted that this interconnection structure is only used for the rendezvous signalling, and that the actual data being transmitted can still follow optimal policy-compliant forwarding paths, even when the signalling paths may be longer in order to achieve the required Internet-wide scalability.

It is also possible to mix and match these interconnection strategies. While participating in various interconnection overlays, the rendezvous networks may also elect to practice pair-wise peering with some of the other rendezvous networks. This may happen using the interconnection overlays (in form of additional fingers), or directly between the rendezvous networks. In such cases the participants may also agree on (local) popularity based proactive distribution of the most scopes, while the interconnection overlays will function as the "default route" to locate the rest of the scopes.

4.4.3 Rendezvous Example

In this section we give an overview of one possible way of implementing the PSIRP inter-domain rendezvous system. As the detailed design of the inter-domain rendezvous is still in progress this implementation overview contains several open issues, which are noted in the text below. Also, while typically there are multiple implementation strategies for each function, we present only one such strategy here.

Figure 4.3 represents an example inter-domain rendezvous system containing three rendezvous networks; A, B and C (dashed ellipses), which are interconnected through an interconnection overlay (the solid ellipse). In the system we have two types of physical entities, namely *rendezvous nodes* (RN) and *end-nodes* (N), and three types of logical entities: *rendezvous point* (RP), *subscriber* (Sub) and *publisher* (Pub). This example rendezvous configuration is used as the reference topology in the examples below where we go through the basic rendezvous operations in simple usage scenarios. This configuration is simplified for the ease of exposition and illustrates only the critical entities from the rendezvous viewpoint. In real scenarios both the rendezvous networks and the interconnection overlays would be more complex.

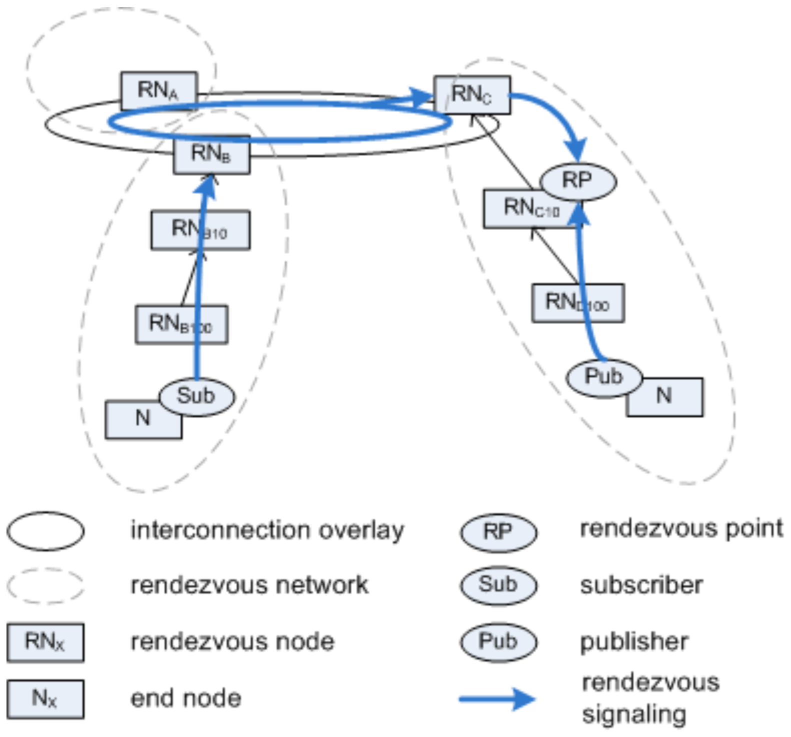

Figure 4.3 – Reference rendezvous topology

The overall rendezvous functionality can be divided into three aspects:

1. Node internal rendezvous,
2. rendezvous within a rendezvous network and
3. rendezvous between rendezvous networks.

In the example operations we consider the two latter parts, skipping the node internal aspects.

Rendezvous System Bootstrap

Before the rendezvous system can operate it must bootstrap. Figure 4.4 illustrates a simplified rendezvous bootstrap message sequence chart. From the figure we can distinguish three separate bootstrap operations taking place independently:

1. Rendezvous network bootstrap (1a), where end-nodes within the domain are made aware of at least one rendezvous node (RN), serving as the end-node's proxy to the rest of the rendezvous system. If an end-node has no knowledge of any rendezvous nodes, it can assume to operate only within the context of its immediate physical links (link-local operation). As part of the rendezvous network bootstrap the end-node becomes aware of the specific scope identifier (SId) with which it can reach the rendezvous node. If the local domain has several rendezvous nodes, each of them may have their own SId reserved for the intra-domain rendezvous use.
2. Inter-RN bootstrap (1b and 1c), where all rendezvous nodes belonging to a rendezvous network will establish topology for the rendezvous network, i.e. each RN becomes aware of their neighbouring RNs and their relation to them. The relation can be a *customer*, *peer* or *provider* relation, defining the behaviour of the communicating

RNs in the different usage scenarios below. For example, in rendezvous network C we have a "tube" rendezvous topology where RNC is rendezvous provider for RNC10, thus making the RNC10 rendezvous customer of RNC and similarly RNC10 is provider for RNC100. An RN that has no rendezvous provider is called a root-RN of its rendezvous network.

3. Inter-rendezvous network bootstrap (1d), where the interconnection overlay between the rendezvous networks gets established. Inter-rendezvous network bootstrap functionality is still work-in-progress and is not handled in detail here.

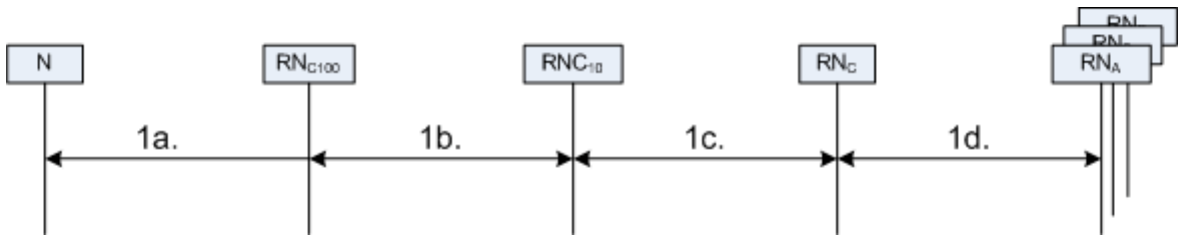

Figure 4.4 – Bootstraping the rendezvous system

Publish

The example of a publish operation in Figure 4.5 begins when an application in a node that is currently located and attached to the rendezvous network C wishes to publish some data outside the node. Due to the successful rendezvous network bootstrap phase the node is aware of an active rendezvous point (RNC100), reachable through one of its network interfaces. Each end-node in the PSIRP system contains the so-called rendezvous helper function which initiates the rendezvous signalling for the publish operation.

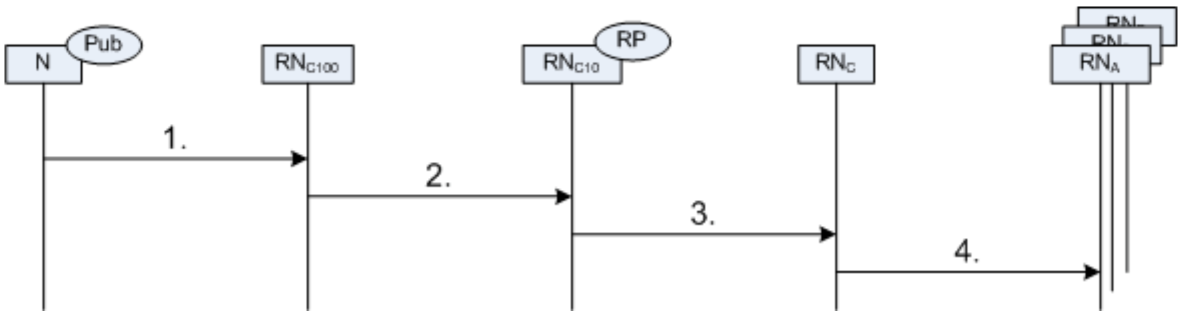

Figure 4.5 – Example of a publish operation

Each rendezvous signal consists of outer and inner headers and a payload. The outer header can be seen as a transport header within the rendezvous system and it may change in each hop. The inner header and the payload contain the identifiers, metadata and possible data relating to the actual rendezvous signal and the publication in question. The work on rendezvous signal formats is still in progress and we will not dig deeper into that issue here. In the messaging charts we have also left out the possible acknowledgement messages.

When the publish signal (1) arrives to RNC100, the receiving node first verifies the header information to confirm that the signal is valid. Assuming that the signal passes the validation tests, as a next step the RN must figure out what it should do with the signal. Let us assume that the SId in the inner header is unknown to the RNC100 – unknown in the sense that the RN's data structures have no reference to the SId among its active rendezvous points (RP) or in the scope cache. In such a case the RN needs to determine whether it should create a new rendezvous point for the scope in question or just forward the message further. To help with the decision, each RN should manage a list of rendezvous contract identifiers, indicating for which requests it is willing to create new rendezvous points. Obviously, the rendezvous signal should contain a matching identifier, for example in the metadata. Before the RN willing to register the publication state is reached, the message is forwarded up in the rendezvous provider hierarchy. In our example case, RNC100 forwards the signal (2) to its provider RNC10 – either there was no match in the contract list or RNC100 is not configured to create rendezvous points at all (caching only rendezvous node).

When receiving the signal, RNC10 runs the same algorithm as RNC100 above did, only this time the outer header consists of the (SId, RId) pair that RNC100 and RNC10 have agreed to use in their communication. RNC10 finds a match for the contract identifier from its list and therefore becomes an authoritative RP for the scope. Details of the RP initiation in the RN are skipped here. Once the RP is up and running and the metadata and related publisher forwarding information stored to the RP data structures, the newly added scope must be advertised within the rendezvous system. To achieve this, RNC10 will advertise the scope to all its rendezvous providers and peers. The RNs that receive this advertisement will add an un-authoritative scope entry to their scope cache data structures. The entry contains information that enables the forwarding rendezvous signal towards the advertiser when needed. In addition, if the advertisement is received from the rendezvous customer, the receiving RN will forward the advertisement to all its providers and peers. The same logic is followed recursively until the advertisement reaches the root-RN(s) of the rendezvous network. Advertisements received from a peering rendezvous node are not re-advertised, so that the resulting rendezvous signalling paths remain policy-compliant.

Once the advertisement reaches the RNC the scope and the publication (Rid) become reachable within the rendezvous network. The root-RN then initiates a response message back to the initiating node, informing it of the fate of the message, i.e. whether the publish was successful or not. This message is not depicted in the figure.

Finally, the root-RN, RNC will register the new scope with the interconnection overlay. At the moment we assume a Canon hierarchy [Gan2004] will be used for this, as described in Section 4.4.2 above.

Subscribe

Figure 4.6 illustrates a message sequence chart for the subscribe operation. The subscriber in the example is attached to the rendezvous network B (see Figure 4.3). As in the publish case, the subscribe operation begins when some application in the node wishes to access the publication content identified by a (SId, RId) pair. Because the requested data is not locally available, the node's local rendezvous helper initiates subscribe rendezvous signalling. The signal (1) is sent towards RNB100, of which the node is aware as a result of the successful rendezvous network bootstrap. Again, as in the publish case the RN first verifies the validity of the signal and only after that figures out what to do with it. The subscribe operation in the RN is in general very close to that of the publish operation - the subscribe signal can also lead to the establishment of a rendezvous point in RN, if the SId is unknown to RN and the signal carries a contract identifier that matches with the contract list within the RN. The reader should notice that in reality handling and managing of contract identifiers and creation of RPs can hardly be as simple as described here. The detailed solution is work-in-progress.

In practice, when the RN receives a *subscribe* signal it first checks whether it has active a reference for the inner header SId. If the first search fails and also the contract identifier in the signal results in no match, it means that the RN does not know of the SId and will not establish a RP. Thus it must forward the signal to all of its rendezvous providers and peers. If there is a match, the match can be either among active RPs or in the advertised scope cache. If the match is in the scope cache the signal will be send towards the RN identified in the cache entry. In case the match is in the RPs, the RN looks for a match among the publications within the RP using the inner header RId as the search key.

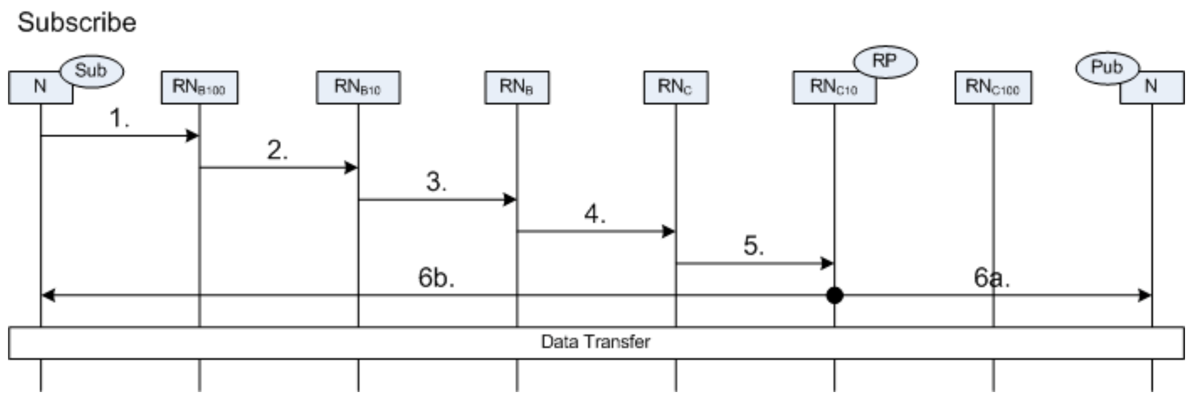

Figure 4.6 – Example of a subscribe operation

If a matching publication cannot be found from the RP, it means one of the following:

- If the RP is not distributed, the publication does not exist (yet) and the RP may register the subscription as a pre-established subscription (if requested by the subscriber) or alternatively drop the signal and indicate a failure to the subscriber.
- If the RP is distributed, the RN should find out if the RId can be located in some of the other RNs hosting the distributed RP. The details of the RP distribution are still work-in-progress.

Further on, if the SId in the inner header is unknown, but the contract identifier matches, the RN becomes an authoritative RP for the SId. In that case the RN registers the subscribe signal to the new RP and advertises the scope to its rendezvous providers and peers.

In our example, RNB100 fails both in the SId and contract search and therefore the signal is forwarded (2) towards the next rendezvous provider, RNB10. In RNB10 the same procedure and result recur, and the signal is forwarded to RNB. When RNB receives the subscribe signal it will repeat the procedure and find no match for the SId or the contract identifier. Because RNB is a root-RN, it forwards (4) non-matching rendezvous signals to the interconnection overlay, which stores pointers to all active scopes within the rendezvous system. Skipping the Canon forwarding details the signal is eventually received at RNC. In RNC the first search results hit in the scope cache. The cache entry points towards RNC10 and the signal is forwarded (5) towards it. On receiving the signal, RNC10 finds a match for both SId and RId in one of its active RPs – the rendezvous has happened. After this, several tasks still need to be finished before the publication data can be transmitted between the publisher and the subscriber:

- The subscription is stored to the RP.
- A forwarding path between the publisher and subscriber must be generated or the subscriber should be added to the existing forwarding tree based on the forwarding information provided within the rendezvous signals. The main responsibility of this is at the topology functionality, but the rendezvous signals need to carry the information that enable this to happen. The working assumption is that all publishers and subscribers provide their policy compliant network upgraphs to the RP, based on which the forwarding paths can be constructed.
- If the RP holds publication metadata, it is sent (6b) to the subscriber.
- The RP may also inform (6a) the publisher about the subscriber.

4.5 Topology: Management and Formation

The following section introduces the topology management and formation function, starting with the intra-domain management before presenting the inter-domain formation function.

4.5.1 Intra-domain Topology Management

The role of the intra-domain topology management is analogous to the functions carried out by the interior gateway protocols in today's Internet. After the inter-domain topology function has performed its task, one or more domain level paths have been selected or formed. The individual instances of the intra-domain topology management function then operate in the individual domains to manage the relevant intra-domain forwarding structures, ensuring that publications are delivered correctly both to the subscribers within the domains, as well as between ingress and egress points for transit traffic. Similar considerations to the granularity of the subscriber information used in these processes hold as for inter-domain topology management.

It should be noted that beyond having to satisfy these rather broad functional requirements, the individual domains have a great deal of latitude in terms of the detailed design of their topology management functions. At present, for improving our understanding of the issues involved (and also for prototyping purposes), the project is outlining in more detail a solution reminiscent of the existing link-state routing protocols (such as OSPF), but redone to benefit from the native publish/subscribe functionality in the network. However, this is not meant to lead to a canonical PSIRP architecture solution for intra-domain topology management, but only to serve as a single design example.

The high level functionality of the system consists of the following steps. For concreteness, we call a node interested in receiving information about the physical topology of the network a **Topology Manager** (TM). Each network might have a single TM or a few of them, either all participating into the topology management function simultaneously or with anycast functionality being used to select the most suitable one for each message; it is even possible for all forwarding nodes to be TMs. The use of native publish/subscribe allows the choice between these to be transparent to other elements in the network.

Each forwarding node can now learn its local connectivity by means similar to the ones used in network attachment (by publishing and subscribing to local HELLO messages). Local connectivity can be represented, for example, as a neighbour list, possibly annotated by link quality and forwarding node capability information obtained through appropriate helper functions. Nodes then publish this local connectivity information to the TM, which can then reconstruct the overall forwarding level topology of the network, and run any appropriate algorithms when triggered, for example, by the rendezvous system and / or inter-domain topology management.

Relation to Intra-domain Forwarding with zFilters

Section 4.6.1 presents a particular solution for intra-domain forwarding with which the topology management function needs to collaborate in order to perform its desired forwarding function. While we refer to Section 4.6.1 for the specifics of this forwarding solution, we outline the relation of the topology management function and the forwarding function.

In short, the zFilter approach of Section 4.6.1 requires knowledge of individual links (identified via so-called *link identifiers*). The appropriate link identifiers need to be inserted into a Bloom filter based packet header (the zFilter), enabling the forwarding nodes within the domain to make the appropriate forwarding decisions so that this packet may be delivered via the identified links. This insertion of appropriate link identifiers is based on the information on RIds and SIds for each publication, which is provided within the process of inter-domain topology formation (see Section 4.5.2). This information on RIds and SIds allow for determining the (domain-specific) endpoints for delivery between which an appropriate forwarding graph needs to be built (based on the link identifiers). The derived path information, encoded as a zFilter, is used by the *source node* of the delivery graph to send packets to the receivers of information.

However, there are cases (see Section 4.6.1) when it is required that *virtual trees* are created, e.g., for paths that are constantly used. The topology management function is responsible for defining these virtual paths that would lead to a more efficient data transmission. This task means that the topology manager needs to have methods to evaluate the current traffic patterns so as to create more efficient transmission paths as well as methods to decide when new virtual paths are needed taking also into account that each virtual tree in the network creates additional state at the affected forwarding nodes' forwarding tables. In addition, the topology manager also needs to define the lifetime for such paths. Managing such virtual trees efficiently in the local network is a complicated task and requires suitable algorithms and processing power in the topology layer. While this is not necessary in unicast or sparse multicast cases, it is, however, mandatory in the case of dense multicast trees with the proposed zFilter forwarding.

4.5.2 Inter-domain Topology Formation

The topology layer works in conjunction with the rendezvous and forwarding functions in the *routing phase* of the communication, i.e., it helps building the routing information of the forwarding nodes and the forwarding paths according to the policies set by operators and users. It also manages edge routers between domains that prevent policy violations and protect domain internals. The topology layer stores the policies and network topology information that are either hand configured or collected automatically from the network. In this section, we explain the problem of *inter-domain topology formation*, our assumptions and the requirements for the solution and technical considerations regarding possible solutions.

Background

We assume that a PSIRP network will be divided into autonomous systems, or domains, controlled by different, possibly competing, organizations, similarly to today's Internet. The domain level connectivity is largely determined by the relationships between these organizations and the needs of their customers, geographical, historical, and political considerations, and only indirectly guided by the technology used. The starting point for our work has been the current Internet topology but it is expected that the change in the underlying network paradigm will affect the evolution of the domain level topology. In the current Internet, the relationships between ASes can be roughly divided to customer-provider, peer-peer, and sibling-sibling types. Also, there are ~10 so called tier-1 operators that peer with each other and do not buy transit from other operators and form a de facto monopoly in the core of the Internet. In our work, we start from the assumption that the Internet in the future will be operated by a similar mixture of competing, commercial operators seeking profit and communities like universities and governments that may have other goals.

The path taken by packets must reflect the policies between operators and also RId specific policies negotiated by the publishers and subscribers of the RId in the rendezvous process. In the current Internet, the packets, in principle, typically flow along so-called *valley-free paths* [Gao2001] consisting of 0-n transit provider links upwards, then zero or one peering link, and finally 0-m transit customer links downwards. This means that the possible routes a packet can take are severely limited, but on the other hand, every domain on the path of a packet has clear incentives to route the packet forward in a relatively simple business model where customer ASes pay their transit provider ASes to reach certain destinations in the Internet.

Currently, home users typically buy peak bandwidth to their local ISPs network instead of paying based on amount of traffic. Furthermore, policy control on the path of the packet is fully left to the operators. This is understandable from a risk management point of view, because end users want to limit the total cost and do not want to actively think about it. On the other hand, in business access transit is often priced as a function of traffic load. In the Internet, the operators on the sender-side of the valley-free route choose the inter-domain path of the packets, taking into account considerations like the length of the AS-level path to the destination and the price to use a particular route. It is interesting to note that the path chosen from A to B may be different than from B to A and both of these can be suboptimal if we sum the costs for both A and B. Based on the current situation, we assume that typical users will require predictable costs in the future too.

Design Requirements

In the course of the work, we have identified the following list of requirements for the inter-domain topology forwarding function:

- The topology layer should allow the operators to flexibly control the routing policies of packets going through their domain.
- It should be possible for customers to define per-RId specific policies; these policies being taking into account in the overall topology formation.
- A PSIRP solution should take into account the costs and policies of both the publishers and the subscribers when building forwarding trees (in contrast to the current Internet where only sender side control of the routing exists).
- The topology layer should have enough expressive power to enable complex policies and business relationships between ASes, not relying on assumptions such as a fixed set of Tier-1 operators and strictly hierarchical AS topology. For example, multi homing and partial transit should be easily possible - this partial transit being RId-specific.
- It should be possible for the operators of domains to keep their intra-domain topology hidden and only be required to expose a minimal amount of information towards the process of topology formation.
- The inter-domain topology formation should not unnecessarily limit the implementation of intra-domain topologies and their management. There can be multiple different implementations inside domains that are compatible with the inter-domain topology formation.
- Incremental deployment on top of the current Internet AS topology should be feasible.
- The topology layer should automatically and quickly adapt to changes in the network topology and efficiently use available routing resources, in accordance with the constraining policies.
- Topology formation should take into account the fact that large domains are linked at multiple geographically dispersed PoPs even if they have only a single logical business association and by leaking some intra-domain information the routes could be further optimized between the domains.
- The topology formation should take into account potentially different policies between the same domains, depending on their point of interconnection.

Design Benchmark

A natural starting point for the work is the BGP protocol used in the current Internet. It is a policy routing path vector protocol and can be used to express complex policies at the price of mediocre scalability. The current implementation also has security problems but they are not an inherent failure of the approach. If we assume that each domain on the path of a message must somehow benefit from transiting a packet and we allow providers to provide paths to different parts of the Internet at different prices, then the path vector is probably flexible enough to accommodate any valid policy. Taking this into account, we believe that the valley-free model of the policy-compliant routes is fully general and stems from the fact that sender and receiver (domain or the end point) policies are asymmetric (maybe excluding the peering links that could be interpreted as symmetric partial transits). Communicating information about the complete paths instead of just their endpoints is required for policies that prevent certain domains on the path. We also use the topology information differently compared to the current Internet, which allows us to cater to application and publication specific policies. In addition, we investigate the usage of packet level authentication described in [Lag2008] that guarantees the authenticity and integrity of the packets, which may enable new policies and tools for management of the traffic.

Conceptual Components

Given the design goals expressed above, we can outline the following functional component architecture for discussing our design considerations.

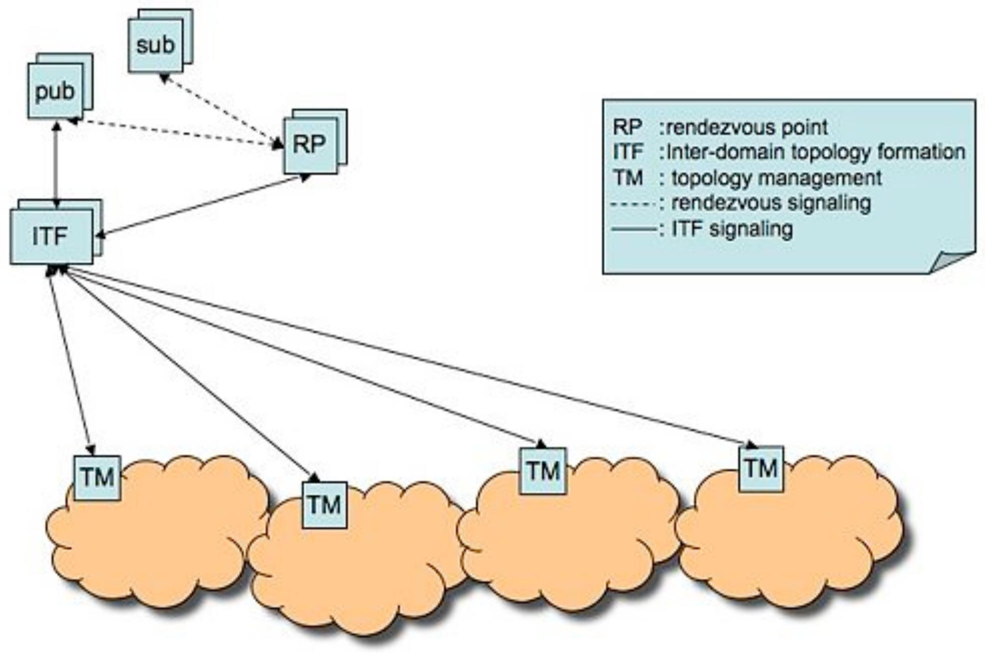

Figure 4.7 – Conceptual components for inter-domain topology formation

A **topology management** function is assumed to exist within each autonomous system (domain). As previously discussed, we assume that this function implements the local topology management and communicates the relevant peering information to the **inter-domain topology formation** (ITF) function. The **publishers** and **subscribers** come together in the rendezvous process within the **rendezvous point** representing the particular SId in which the information items (labelled via an RId) are located. The arrows in Figure 4.1 show the relations of these components and are not meant to illustrate the exact message and information exchange between them. However, the dashed arrows indicate relations stemming from the rendezvous process while the solid arrows show topology formation relations.

Design Considerations

The following section discusses considerations for the design of the inter-domain topology formation process, based on the conceptual architecture outlined in Figure 4.1. We do not present a particular solution for this process since this function is still under development.

Role of the ITF component

The inter-domain topology formation (ITF) function is clearly differentiated from the forwarding function for two major reasons, namely efficiency and modularity.

As for the former, it is expected that the required routing policy calculations may not only be complex but may also require information about the topology that will require a certain amount of storage space. Since the actual forwarding complexity must be low so that it can be implemented in hardware for line-speed operation, it seems natural that the ITF function will calculate the necessary routing data offline rather than in real-time. The ITF function also participates in building the forwarding identifiers (FId) in the rendezvous phase and part of the routing information can be embedded in the FIds that are stored in the header of each packet. Each router should then be able to make a forwarding decision including security checks based only on the routing table and the FId of the packet. In the design of the system, a balance between topology related state in packet headers and in routing tables must be found. For example, very large headers affect the bandwidth negatively whereas requiring lots of memory and processing power in core forwarding nodes makes the system more expensive.

The modularity aspect allows for considering several ITF functions to exist in the system, each of which 'primes' the forwarding functions according to a set of policies that reflect very particular business constellations. As an example, one ITF could reflect the peering information as well as policies that largely reflect the current best-effort Internet. The resulting forwarding graphs are largely similar to what could be observed in today's Internet. Another ITF, however, could implement forwarding graphs that take into account peering links exclusively for particular business relations. For instance, highly resilient peering links for financial services could be exposed to this ITF, likely leading to different forwarding graphs that will take these additional peering relations into account.

Inter-domain topology information

Before publication, the rendezvous system or systems provide the details of current subscribers for the SId and RId in question. The granularity of subscriber information registered with the rendezvous systems is still being considered within the PSIRP project. The broad options considered are:

- Forwarding network identifier
- Landmark identifier
- Forwarding tree identifier

If a forwarding network identifier is known for each subscriber, then the role of the ITF function is to resolve a list of forwarding networks, so as to complete a path tree between the publisher forwarding network and those of the subscribers. Each preceding forwarding network will be instructed to pass the publication onto the subsequent networks at any available peering point.

In order to improve the construction of an efficient publication tree, especially for large networks, we can use a couple of techniques. It is obviously true that a network provider may decide to subdivide its network, and so provide a sub-network identifier for their subscribers. A similar technique would be to specify a network landmark to which the subscriber is in proximity. This landmark may then be used by the preceding network, and indeed the ITF function, to determine the best peering points, and the best overall route for the publication.

An alternative is to construct partial distribution trees within the forwarding networks in anticipation of publications. In this case the tree identifier may be shared with the rendezvous system and the ITF function. Such tree identifiers would also need to be shared with neighbouring networks in order to select the best peering point for connection to the established tree. One drawback of this approach is that the rendezvous systems (along with inter-domain topology routing and neighbouring forwarding networks) would need to be informed when the subscribers within a forwarding network change and new trees are established for existing subscribers. This would therefore result in substantially larger volumes of signalling traffic in order to establish the requisite routing state.

A publish/subscribe approach to inter-domain topology formation

All of the components fulfilling or interacting within the process of inter-domain topology formation can utilise the publish/subscribe capability of the network itself for doing so. With this in mind, information about peering routes is seen as being published towards one or more ITF components (see the role of this component above), each of these ITF components implementing a particular scope of peering information, identified via an SId each.

In this manner, route changes do not have to be propagated towards one domain at a time, but can potentially be notified to all relevant ITF components simultaneously. This may help prevent some of the problems found in today's BGP such as route flapping and problems found in ad-hoc networks (e.g. gossip networks) such as knowing when to stop propagating notifications.

In order to be able to use publish/subscribe for the ITF function, it must, of course, be possible to 'bootstrap' the system. This problem is similar to the network attachment problem for end hosts (see Section 4.7), only in this case we attach individual networks to the inter-domain routing system.

Creating a peering market

Given that we identified the possibility for different ITF components to co-exist and the ability to use the pub/sub functionality of the network itself to bootstrap the inter-domain forwarding functionality, we can identify similarities to the rendezvous component (see Section 4.4) and the market structure that we can see in this space. Hence, the exposure of different sets of information to different providers of peering relations is likely to create a (policy-driven) peering topology market that is architecturally embedded into PSIRP.

Control of the formation process

Within the inter-domain topology formation process, there is significant flexibility as to which party initiates the process, which parties participate in the formation decision, and which parties simple provide relevant information (such as policies or peering information).

The choice largely comes down to three factors:

- Where is it most efficient to perform the role?
- Where can information (consisting of routing information and policies) be shared without compromising confidentiality?
- Who is trusted to make a policy compliant routing decision?

We can see that in order to make the routing decision, information is required from a number of parties:

- Publishers (or their local forwarding network or the local rendezvous system for the forwarding network) provide the publisher location and publisher routing policies
- The rendezvous point provides the locations of (and any routing policies for) subscribers
- Each forwarding network (via the topology management function) provides information on its peering arrangements and peering policies towards a particular ITF function.

If the information cannot be shared with a single party, then we must consider multi-party negotiation to arrive at a final policy compliant decision. While the rendezvous point may know about publisher and subscriber information, it is likely that forwarding networks will not wish to share their peering topologies and policies with rendezvous providers. A refinement would be for the forwarding networks to share their overall peering topology, but hold their policies privately. Each network's topology management function would then be consulted to establish whether the route was policy compliant.

A similar alternative is for the each network's topology management functions to share peering information between themselves. Thus, the publisher's local network would be able to establish the best path. In this case each network may still hold their policies privately and be consulted during the path formation.

Achieving fault tolerance and multipath routing

It is possible that the inter-domain topology formation function could select multiple routes from the publisher to the subscribers. Multipath routing can provide advantages for spreading network load, providing fault tolerance and for providing security against eavesdropping within the network (since secret information is divided between multiple paths). For fault tolerance, either the alternative paths may be used in case of failure, or alternatively the traffic may be shared across multiple paths, or even replicated in extreme cases where delays in retransmission are not acceptable. A downside of multi-path routing is that network characteristics between paths may vary widely, and may be unpredictable during heavily congested periods. This may cause problems with latency sensitive protocols (such as TCP), or where packet ordering should be preserved. A common suggestion, and one that could be taken in the PSIRP architecture, is for sensitive application flows to be assigned to single paths, but for multiple paths to be used by the overall application or host.

Although multiple paths can be calculated for the inter-domain topology, there is no guarantee that separation will be maintained for routes that share forwarding domains (which at least include the publisher and subscriber domain unless multi-homed). Where publishers and subscribers are multi-homed the selection of multiple routing paths must be made by the application or host, since the network will be unable to correlate the same publisher attached to multiple networks. If this is a requirement, then the implication is that the route selections are exposed to the host device. For example, when requesting a publication route the application may include a policy that excludes the domains used in an already existing path calculation.

Anycast

It is possible for the ITF function to support different notions of anycast. For example the list of subscriber domains obtained from the rendezvous system can be passed to the inter-domain topology formation function, which selects a path to only one domain. This decision may be based upon a minimum number of domain traversals and knowledge of average hop counts between peering points (although the exact path internal to the domain is outside of the control of the inter-domain topology function). Other forms of anycast may also be supported such as cheapest or least congested (although this data may be stale and may be better implemented through the end-to-end testing of multiple paths). Any metrics used in the decision must be collected from the forwarding networks and the criteria signalled through the publisher routing policy.

To achieve anycast, the capability within the ITF function must be mirrored within the intra-domain routing. Since at this point there is no longer an interaction with the publisher (or originating network), various anycast functions would need to be signalled through packet metadata (headers). This would signal to the end subscriber domain that only the 'best' subscriber within that domain should received the packet. Since a meaningful flow of information is likely to be larger than a single packet, there would also need to be some indication (such as a flow ID) that a series of packets were to be sent to the same subscriber.

4.6 Forwarding

It is a major assumption for PSIRP that there should not be any reliance on endpoint-based end-to-end addressing (as in IP). This directly reflect upon the forwarding structures that need to be considered for an architecture that names information items (see Section 2) instead of the endpoints publishing and subscribing to these items. It is further assumed that a multicast style packet delivery would occur by virtue of employing a publish/subscribe operation as the main communication paradigm.

Considering the forwarding fabric to be presented in the following subsections, we can formulate a few general design requirements. A major one is efficiency as being measured as the latency and bandwidth of the communication between subscribers and publishers. *Efficient* routes are likely to be correlated with their goodness in number of AS level hops, locality, reliability, and costs to the parties involved. Related to this, any forwarding node algorithm should be simple enough to allow a fast hardware implementation at current and future line speeds. With this in mind, it seems natural that more complex decisions related to routing, policies, security, and topology management should be done in specialized nodes (*the slow path*) in order set up the simpler forwarding nodes that form the basic routing fabric (*the fast path*).

Another crucial requirement is that FIds should be of temporary nature and be renewed for long-term connections. That is, senders are not allowed to send packets to receivers without an existing subscription (reflecting this longer-term connection). This protects the network from DDoS attacks because circumventing valid forwarding trees is not possible and communication along forwarding trees is scalable and access controlled. In addition, the forwarding algorithm should also support arbitrarily large (multicast) groups.

With this temporary nature, it becomes obvious that the FId should not be directly tied to a particular RId, meaning that a single FId can be used for transmission of within different RId relations. This allows re-using certain transmission trees, identified by the Fid, rather than creating new trees for each of the subscriptions. With such a solution, it is expected that the amount of necessary state in the network is reduced.

We further assume that the forwarding will take place in a similar domain-like structure as in the current Internet. This reflects the different ownership relations of physical infrastructure owners and allows for catering to the local requirements and constraints of these owners. With this in mind, we divide the presentation of the forwarding function into solutions for intra-domain forwarding as well as inter-domain forwarding. The former gives an outlook for a potential solution for single domains, based on a source routing style Bloom filter approach. The latter addresses design considerations to be taken into account for inter-domain solutions.

4.6.1 Intra-domain

In this section we present a solution for intra-domain forwarding in a PSIRP network, i.e., a network that does not have any end-to-end addresses. The packet forwarding is based on link IDs, identifying the links between the forwarding nodes, and on Bloom filter -style FIds in the packet header. This presented forwarding fabric is efficient when packet forwarding is considered in a single forwarding node, as well as it has a very compact forwarding header in each packet. This forwarding fabric can also be implemented in an efficient way on hardware.

Creating a Forwarding ID

To forward packets through the network, we use a hybrid approach where the topology system both constructs Bloom-filters-based forwarding identifiers, used in a source-routing manner, and on demand installs new state at the forwarding nodes. For both of these functions, we use an approach where the links, not the nodes, have names. First, we present the basic ideas in a somewhat simplified form, and then we introduce ways to handle, e.g., loop prevention and error recovery.

For each point-to-point link, we assign two identifiers, called *Link IDs*, one in each direction. For example, a link between the nodes *A* and *B* has two identifiers, *A->B* and *A<-B*. In the case of a multi-point link, such as a wireless link, we consider each pair of nodes as a separate link. With this setup, we do not need any common agreement between the nodes on the link identities - each link identity may be locally assigned, as long as the probability of duplicates is low enough.

Basically, a Link ID is an *m*-bit long name with just *k* bits set to one. In the context of this document, it is sufficient to note that typically $k << m$ and *m* is relatively large, making the Link IDs statistically unique (e.g., with $m=248$, $k=5$, # of Link IDs ~ $m!/(m-k)!$ ~ $9*10^{11}$).

The topology management system (see Section 4.5) creates a graph of the network using Link IDs and connectivity information, without any dependency on end-point naming or addressing (creating the *topology map* or *routing table*). Using the network graph, the topology system can determine a forwarding tree for any publication, from the locations of the publisher and subscribers.

When the topology system receives a request to determine a forwarding tree for a certain publication, it first creates a conceptual delivery tree for the publication using the network graph. Once it has such an internal representation of the tree, it knows which links the packets need to pass, and it can determine when to use Bloom filters and when to create state [Zah2009].

In the default case, we use a source-routing-based approach, which makes forwarding independent from routing. Basically, we encode all Link IDs of the tree into a Bloom filter, placed in the packet header.

Once all link IDs have been added to the filter, a mapping from the data topic identifier to the BF is given to the node acting as the data source, which now can create packets that will be delivered along the tree. To distinguish the BFs in the actual packet headers from other BFs, we refer to the in-packet Bloom filters (together with an additional byte) as *zFilters*, see Figure 4.8.

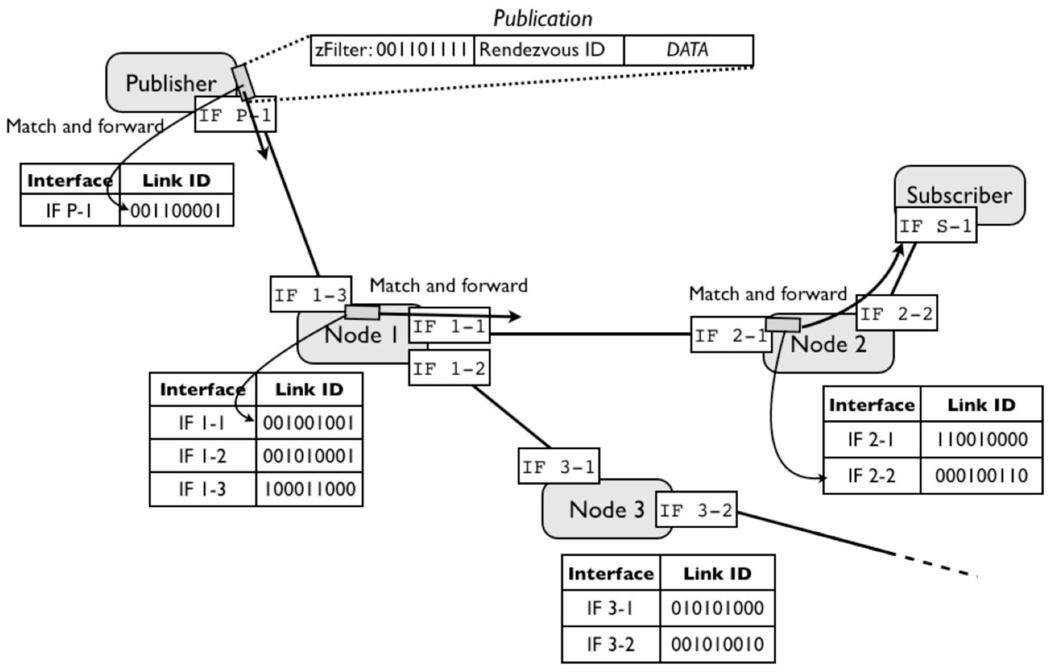

Figure 4.8 – ZFilter forwarding tables

Link IDs and LITs

To reduce the number of false positives, we now introduce *Link ID Tags* (LITs), as an addition to the plain Link IDs. The idea is that instead of each link being identified with a single Link ID, every unidirectional link is associated with a set of *d* distinct LITs (see Figure 4.9). This allows for constructing zFilters that can be optimized, e.g., in terms of the false positive rate, compliance with network policies, or multi path selection. The approach further allows for constructing different candidate zFilters and to select the best-performing BF from the candidates, according to any appropriate metric.

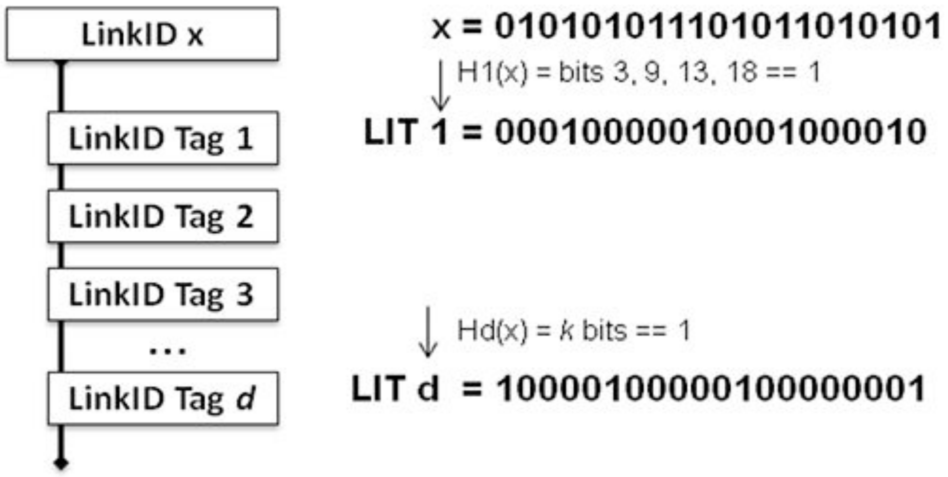
LinkID x
LinkID Tag 1
LinkID Tag 2
LinkID Tag 3
...
LinkID Tag d
x = 01010101110101101010 1
H1(x) = bits 3, 9, 13, 18 == 1
LIT 1 = 00010000010001000010
Hd(x) = k bits == 1
LIT d = 10000100000100000001

Figure 4.9 – LIT generation

The forwarding information is stored in the form of *d* forwarding tables, each containing the LIT entries of the active Link IDs, as depicted in Figure 4.10. The only modification of the base forwarding method is that the node needs to be able to determine which forwarding table it should perform the matching operations; for this, we include the index in the packet header.

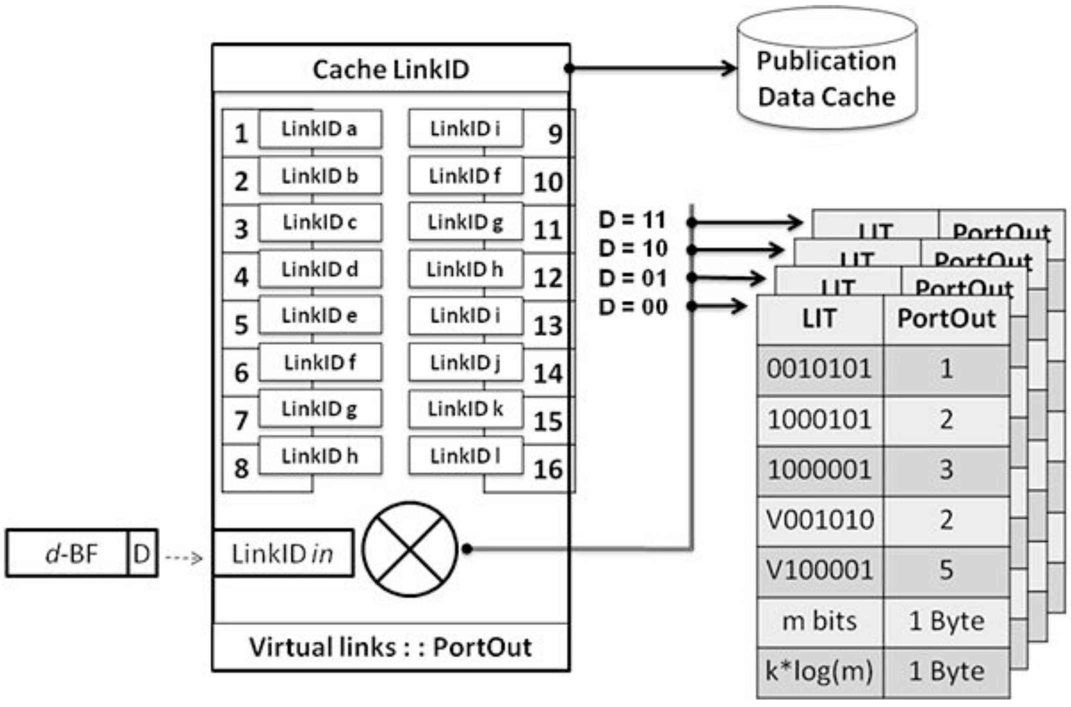

Figure 4.10 – LIT router model

Construction

The zFilter representing an actual delivery tree is computed from the participant list (publisher and subscribers) received from the rendezvous and the network graph collected by the topology management system [Zah2009]. When determining the actual forwarding tree, we can apply various policy restrictions (e.g., for link avoidance) and keep traffic engineering in mind (e.g., balancing traffic load or avoiding temporarily congested parts of the network). As a result, we get a set of unidirectional links to be included into the zFilter. The final step is ORing together the corresponding LITs of the included links, yielding a candidate BF. As each link has *d* different identities, we get *d* candidate BFs that are "equivalent" representations of the delivery tree. That is, a packet using any of the candidates will follow, at minimum, all the network links inserted into the BF.

Selection

Recall that a false positive will result in an excess delivery; i.e., a packet will be forwarded over a link that is not part of the delivery tree. To achieve better performance in terms of lower false positive probability, we first consider two relatively simple strategies:

- **Lowest false positive after hashing (fpa)**: The selected BF should be the one with the lowest false probability estimate after hashing: $min\{r_0\hat{}k_0, \ldots, r_d\hat{}k_d\}$, where r_i is the fill factor, i.e. the ratio of 1's to 0's.
- **Lowest observed false positive rate (fpr)**: Given a test set T_{set} of link IDs, the candidate BF can be chosen after counting for false positives against T_{set}. The objective is to minimize the observed false positives when querying against a known set of Link IDs active in the forwarding nodes along the delivery tree.

The *fpa* strategy is simple and aims at lower false positives rates for any set of link IDs under membership test. On the other hand, the *fpr* strategy yields the best performance of false positives for a specific test set at the expense of higher computational complexity.

To further enhance *fpr*, false positives at different places can be weighted, i.e., we can consider some false positives less harmful than others. For example, we can avoid forwarding towards non-peered domains, resource constrained regions, or into potential loops. We call such selection criteria as *link avoidance*, since they are based in penalizing those candidate BFs that yield false positives when tested against certain links. For example, the following kinds of criteria could be considered:

- **Routing policies**: T_{set} of links to be avoided due to routing policies.
- **Congestion mitigation**: *Static* T_{set} of links avoided due to traffic engineering (e.g., low capacity links) and *dynamic* T_{set} of congested links.
- **Security policies**: T_{set} of links avoided due to security concerns.

As a consequence, having *d* different candidates each representing the given delivery tree is a way to minimise the number of false forwarding decisions in the network as well as restricting these events to places where their negative effects are smallest.

Forwarding Nodes

Each forwarding node makes forwarding decisions for the incoming packets and the procedure is roughly as follows. For each link, the outgoing Link ID is ANDed with the zFilter found it the packet. If the result matches with the Link ID, it is assumed that the Link ID has been added to the zFilter and that the packet needs to be forwarded along that link.

With Bloom filters, matching may result in some false positives. In such a case, the packet is forwarded along a link that was not added to the zFilter, causing extra traffic. This sets up a practical limit on how many link names can be included into a single zFilter.

Our approach to the Bloom filter capacity limit is twofold: Firstly, we use recursive layering [Day2008] to divide the network into suitably sized components. Secondly, the topology management system may dynamically add *virtual links* to the system.

A virtual link is, roughly speaking, an unidirectional delivery *tree* that consists of a number of links. It has its own Link ID, similar to the real links. The functionality in the forwarding nodes is identical: the Link ID is compared with the zFilter in the incoming packets, and the packet is forwarded on a match.

Multicasting

While the presented forwarding mechanism is based on including link IDs in the FId, it can be easily seen that the mechanism supports multicasting by nature. For a simple multicast scenario, where the packet is delivered to two different paths from a forwarding node, it is only required that both of these outgoing link IDs are added to the zFilter in the packet header. Once the forwarding node receives the packet, it delivers it to both of these outgoing interfaces.

As the results show [Jok2009], in a simple scenario, without virtual trees, the system supports sparse multicasting inside a metropolitan area network. For dense multicast, however, it is required that the topology management creates some virtual trees to avoid too full Bloom filters in the packet headers that result in too high false positive rate.

Stateless vs. Stateful Functionality

So far, we have considered only stateless operations, where each forwarding node maintains only a static forwarding table storing the LITs. We now carefully introduce state to the network in the form of virtual links and fast failure recovery. While increasing hardware and signalling cost, the state reduces the overall cost due to increased traffic efficiency when facing large multicast groups or link failures.

Virtual links

Our goal is to maintain a forwarding efficiency at the level of 95% (i.e. 5% of false positives). This can be achieved in sparse multicast trees with the basic zFilter based forwarding mechanisms. However, in case of dense multicast trees, we have to consider other methods to maintain a decent forwarding efficiency.

In case of dense trees, especially when multiple trees share multiple consecutive links, it becomes efficient to identify sets of individual links with a separate Link ID and associated LITs. We call such sets of links as *virtual links*. The abstraction introduces the notion of tunnels (or link aggregation) into our architecture -- a notion more general than traditional one-to-one or one-to-many tunnels, being able to represent any link sets, including partial one-to-many trees, forests of partial trees, many-to-one concast trees, etc.

A virtual link may be generated by the topology layer, whenever it sees the need for such a tree. The creation process consists of selecting the individual links over which the virtual link is created, assigning the link a new Link ID, and computing the LITs. Once the virtual link has been generated, the topology layer needs to communicate the Link ID, together with the LITs, to the nodes residing on the virtual link.

Note that virtual link maintenance does not need to happen in line speed. There are always alternative ways of sending the same data. For example, if a virtual link is needed to support a very large multicast tree, the sender can send multiple packets instead of one, each covering only a part of the tree.

Once the virtual link creation process is finished, we can use a LIT of this virtual link in any zFilter instead of including all the individual LITs into it. This reduces the probability for false positives when matching the zFilter on the path. On the other hand, adding forwarding table entries into nodes increases the sizes of the forwarding tables. Given the typical Zipf-distribution of multicast receivers [Liu2005], the sizes of the forwarding tables will still remain small compared to the current situation with IP routers. Unfortunately, falsely matching to a virtual link will mean falsely tunnelling packets through the entire connected part of the denoted subgraph. However, this can be mitigated by careful naming of the virtual links (e.g., more 1-bits than in the case of physical links) and by explicitly avoiding false positives during BF-selection.

Fast recovery

Whenever a link or a node fails, all delivery trees flowing through the failed component break. In this section, we consider two approaches for fast re-routing, examining the simple case of single failures.

Our first approach is to replace a failed link with a functionally equivalent virtual link. We call this the *VLId-based recovery* approach. The idea is to have a separate virtual backup path pre-configured for each physical link ID, to be dynamically used in case of failure. This virtual backup path has the same Link ID and LITs as the physical link it replaces, but is initially inactive to avoid false forwarding.

The main advantage of this approach is that there is no need to change the packets. Basically, it is enough that the node detecting a failure sends an activation message over the replacement route, activating the backup route for both the failed physical link and any virtual links flowing over the physical link, and then starts to forward the packets normally. When receiving the activation message, the nodes along the backup path reconfigure their forwarding tables, starting to forward packets along the replacement path. As a result, the packets flow, unmodified, over the replacement path.

Another approach is to have a pre-computed zFilter for the replacement route, encoding a valid representation of the backup path. In this method, when a node detects a failure, it simply needs to add the appropriate LIT(s) representing the replacement route into the zFilter in the packet. This method does not add any additional signalling or state to the forwarding nodes, but it increases the probability of false positives by increasing the fill factor of the zFilter.

Both of the mechanisms are capable of re-routing the traffic with zero convergence time.

Loop prevention

In some cases false positives can result in loops; for instance, consider the case where a zFilter encodes a forwarding path *A->B->C*, but, due to a false positive, the zFilter also matches with a separate link *C->A*, which is used to forward packets from *C* to *A*. Without loop prevention, this will cause an endless loop of *A->B->C->A*. Obviously, we can construct a set of links that may cause a loop and use the *fpr* method to select only loop-less candidate BFs. However, this does not guarantee loop freeness as the network changes.

As an alternative solution, we start with each node knowing the neighbouring nodes' *outgoing* Link ID and LITs towards the node itself. We call these as the *incoming* Link ID and LITs. Now, for each incoming packet, the node checks the *incoming* LITs of its interfaces, except the one from where the packet arrives, and compares them to the zFilter. A match means that there is a possibility for a loop, and the node caches the packet's zFilter and the incoming Link ID for a short period of time. In case of a loop, the packet will return over a different link than the cached one. Our early evaluation is based on this approach and suggests that a small caching memory does not penalize the forwarding performance.

As a third alternative, at the inter-AS level we can divide the links into up, transit, and down ones, and utilise the valley-free traffic model. As a final method, it remains always possible to use a packet TTL similar to what IP uses today.

4.6.2 Inter-domain

Although the generic requirements also apply for the inter-domain case, we have some specific requirements that we need to take into account when designing the inter-domain forwarding mechanism. The solution presented in the previous section for intra-domain forwarding cannot be directly applied to inter-domain forwarding. There are both technical, e.g., scalability related, as well as administrative issues when the networks are operated by different instances. To overcome these issues, we need to design a different mechanism for inter-domain forwarding. In this section, we present our basic assumptions, design goals, and technical considerations for the inter-domain forwarding function of the PSIRP architecture.

Some Basic Assumptions

The domain structure of the Internet and PSIRP network is explained in Section 4.5.2, and it forms the basis of the inter-domain forwarding work.

Another basic assumption behind our work is that popularity of RIds follows a typical Zipfian distribution. This is supported by, for example, [Cha2007] and [Gill2007], where Zipf distribution parameters ranging from -0.56 to -2.5 were reported for user generated video content. Also, the usage statistics from user-generated video services like YouTube suggest a very heavy *write-once, read-never* characteristic, i.e., large amounts of content is generated (published) but never consumed (subscribed to). With this, we can assume that there are few massively popular RIds, relatively small number of RIds of medium popularity, and a large bulk of content that has zero or few simultaneous subscribers at any point in time.

Design Goals

We can define the following major design goals for an inter-domain forwarding solution in PSIRP:

- The main requirement for forwarding structures being formed is **efficiency** as measured in latency and bandwidth of the communication between subscribers and publishers. Efficient routes in this sense are likely to be correlated with their goodness in number of AS level hops, locality, reliability, and costs to the parties involved.
    - Resulting forwarding node algorithms should be simple enough to allow a **fast hardware implementation**, i.e., enable forwarding at almost light speed. It follows that the more complex decisions related to routing, policies, security, and topology management should be done in specialized nodes with more logic. These *slow path* components are used to set up the simpler forwarding nodes that form the basic forwarding fabric.
- Packet header overhead should be optimized for small size, optimizing bandwidth.
- Only packets along forwarding trees that were successfully created by the rendezvous function should generally travel longer distances in the network and each stakeholder in the forwarding process should have clear **incentives** to participate in the operation.
    - FIds should be temporary and be renewed for long-term connections. That is, senders are not allowed to send packets to other domains without the consent of a receiver in that domain. This protects the network from DDoS attacks because circumventing valid forwarding trees is not possible and communication along forwarding trees is scalable and access controlled.
- The forwarding algorithm should be able to **scale** to global use with only standard hardware used. The price of unit bandwidth per end-user to cover the costs of managing the network and to reach any location in the network should be reasonably close to constant for typical traffic matrix if the operators behave rationally.
- The forwarding algorithm should support arbitrarily large **multicast** groups.
- The forwarding architecture should be **extensible** for many uses and communication semantics on top of them.
- The forwarding architecture should enable flexible *peering markets*, in which inter-domain routing underlies different economic scenarios that are likely to be dependent on the type of information items or information networks (see Section 2) that are being transported.

Basic Operation

The forwarding function assumes an input from the **rendezvous** (see Section 4.4) and *inter-domain topology formation* function (see Section 4.5) that constructs, when successful, a new *forwarding identifier* (FId) and related meta-data that allows the subscribers to join an existing or a newly formed forwarding tree reaching the publisher of the RId and receive updates from the publisher. Basically, forwarding trees are source-based multicast trees that follow policy compliant routes. Packet routes should have a minimal stretch with mostly low latency operations performed on the packets at line-speed.

The information of the mapping from the FId to the link layer resources can be partly encoded in the FId itself and partly stored as a state in the network. For example, a FId can denote a forwarding graph embedded in the network connecting the publishers and subscribers in a tree structure. In addition to basic routing, the network resources denoted by a FId can include space for caching data and cycles for simple processing of the data.

The mapping from an FId to network resources can be abstract and probabilistic in nature, because it is important that the inherent redundancy in Internet topology can be utilized for fault tolerance with minimal delays to the end user. This means that a FId should fix the use of specific nodes in the network only when it is absolutely necessary. Ideally, the applications could choose a suitable trade-off between the amount of state and bandwidth needed in the network and the transparency of failures in the latency perceived. Also, the FId should not necessarily be thought as being a connected set of resources, but it can, for example in a mobile setting, determine disconnected structures at each network and these structure are sometimes joined while the underlying connection exists.

We also assume that in the rendezvous process, each RId has an associated owner or publisher that controls the use of the RId and manages its life cycle and related resources. Typically, in a forwarding tree configuration, the owner of the RId resides at the root of the tree.

Modular Design

When a subscription is issued for a publication, the Rendezvous and Topology formation functions are responsible for locating the publication and creating a forwarding ID for the data. However, the global (inter-domain) topology formation function (see Section 4.5.2) cannot always have the required information about the forwarding inside a single domain. This may be for example due to the network operator's unwillingness to reveal its network topology, or due to the dynamic nature of the internal network in which case it would be too slow to inform constantly the global topology formation function about the changes inside the domain. For this reason, a dialogue between local (intra-domain) topology management functions and the particular inter-domain topology formation function is envisioned (see Section 4.5.2).

Taking into account the previous, we need to create a system that can separate the inter-domain forwarding from the intra-domain forwarding. The global topology formation function, being aware of the higher level topology of the global network, can only create forwarding IDs that are used to deliver the data closer to the actual destination using e.g. domain-level forwarding paths.

Trade-offs Between Routing Information Approaches

There are different alternatives to handle the multi-level forwarding IDs. It is possible to use stacking of different levels' FIds in the packet header (i.e., tunnelling), or performing switching between FIds on the packets closer to the subscribers' domains.

Using a tunnelling style solution steadily increases the packet header and moves the required state information from the network to the packet header. On the other hand, using FId switching in the network creates some state inside the network, while it keeps the packet forwarding header compact.

Directly using the RId for switching between FIds at the edge of a subscriber's domain would immediately create a scalability problem since the number of active subscriptions can be huge, thus placing an immense processing and memory burden onto the edge routers. In order to be able to perform FId switching in a scalable way, we have to investigate the possibilities for this. One potential way of doing it could be ,e.g., distribution of the mapping to different nodes inside the domain and the edge node just making the decision to which internal mapping node the incoming publication is forwarded.

It is also possible to use a combination of the aforementioned methods.

Multicast Forwarding

Another issue is that inter-domain forwarding mechanism, when combined with the intra-domain solution, needs to support multicast in a scalable and efficient way. Here we have to consider different cases; subscribers at the leafs, subscribers both at the leafs and at the transit domains, different kinds of transmission speeds in different networks and what are the effects of this to multicast forwarding and so on.

Transport Semantics Considerations

In one-to-one communication the transport level semantics can be implemented in (trusted) end-points only (for example, TCP congestion control), but when there are multiple subscribers or publishers simultaneously exchanging information, the set of possible semantics become more complex. If we divide the routers along the forwarding path into *branching forwarding nodes* that duplicate the received packets into multiple output ports and *core forwarding nodes* that implement only unicast routing, then the end-to-end principle holds only between tree end points and branching routers, but branching routers may perform more complex operations like caching and queuing for the packets based on the metadata of the RId.

Forwarding Incentives

It is possible to think that each subscriber brings their own resources to the forwarding tree when they join it. This way the available resources for the tree grow along the number of the subscribers and each domain has an incentive to forward multicast messages.

Forwarding trees with low branching factor should always have optimal stretch and are routed only through fast core routers. It is possible, however, that stretch is gradually worsened as the number of simultaneous subscribers in the forwarding tree grows.

Feasibility Analysis

Different approaches to the forwarding problem, taking into account our consideration in this section, need to be evaluated along multiple dimensions including:

- latency
- bandwidth
- scalability
- manageability
- cost
- security
- flexibility

Optimizations

It is possible for multiple RIds to share the same forwarding graph to minimize the amount of state needed in the network if the set of RIds are related to each other and most subscribers want to either receive all of them or none at all. For example, all RIds inside the same scope could share the same forwarding structure and subscribers might then receive some extra messages.

4.7 Network Attachment

As mentioned in Section 3.4.5, tasks handled by the network attachment component can include attachment point (AP) discovery and selection, authentication and authorization of users and nodes, and node configuration. In this section we describe how a network attachment mechanism that enables such operations can be designed.

4.7.1 Background

In [Kja2009], a pub/sub-based mechanism that is designed to work on top of a blackboard-based communication infrastructure similar to the PSIRP component wheel is presented. The attachment protocol included in that design is based on a low-level single-packet pub/sub service similar to the one described in Section 3.3.1. In order to support different types of attachment operations, the protocol enables message exchanges over two-way communication channels between attachment points and nodes that join networks. Design alternatives and security considerations are also presented. [Kja2009] mainly focuses on asymmetric attachment scenarios, where single client nodes join existing networks.

Thus we can use the abovementioned mechanism as our starting point, but in this architecture we utilize the concepts of scopes, rendezvous, and forwarding, and also want our attachment mechanism to be suitable for symmetric attachment scenarios.

4.7.2 High-level View of an Attachment Procedure

First of all, an attachment procedure can be initiated if one node subscribes to data published by another one. In addition, nodes seeking wireless network access must usually also be able discover and select APs on their available links as a first step. Consequently, a need for a scheme that makes is possible for two nodes to find each other, and also enables bootstrapping of pub/sub-based communication between them, can be seen.

In order to provide information to others about themselves, nodes can spontaneously publish advertisements that periodically get sent out on their link interfaces. Alternatively, nodes that want to get such information can publish solicitations (i.e., subscriptions) that other nodes can respond to. In either case, these initial messages can contain various information about attachment points and networks[4], including subscription data for establishing two-way control channels with selected nodes. Moreover, these advertisements and solicitations must be published with identifiers that other nodes know and are willing to subscribe to. Pre-distributed or well-known Ids can be used for this purpose.

As indicated above, the initial information message should contain parameters for generating identifiers that the publisher has subscribed to. In this way other nodes can initiate attachment procedures with selected APs. Furthermore, by including similar subscription data in those messages, two-way communication can be established. Examples of node-specific operations that require this kind of communication model are authentication and authorization.

As a result of the steps above, a control channel over which data can be exchanged has become available. An initial attachment procedure can be then be run, for example in order to negotiate about service/compensation contracts and provide nodes with configuration data.

[4] *In fact, in some cases where further negotiations are not needed, all required configuration data can simply be put in this message.*

The result of this phase is that network connectivity is enabled between the nodes attached to each other. Finally, the channel can remain and used for data updates, if needed.

An additional issue that can be considered is re-attachment to other attachment points. Presumably, some initial messages again need to be exchanged between nodes. However, an already established control channel can also be used for informing other nodes in a network about an upcoming attachment point change. This can partly enable re-attachment in a make-before-brake fashion. This area is obviously close related to the mobility issue, for which additional considerations are presented in Section 5.3.

4.7.3 Attachment Protocol Example

Next we outline what an attachment protocol could look like (see Figure 4.11 below).

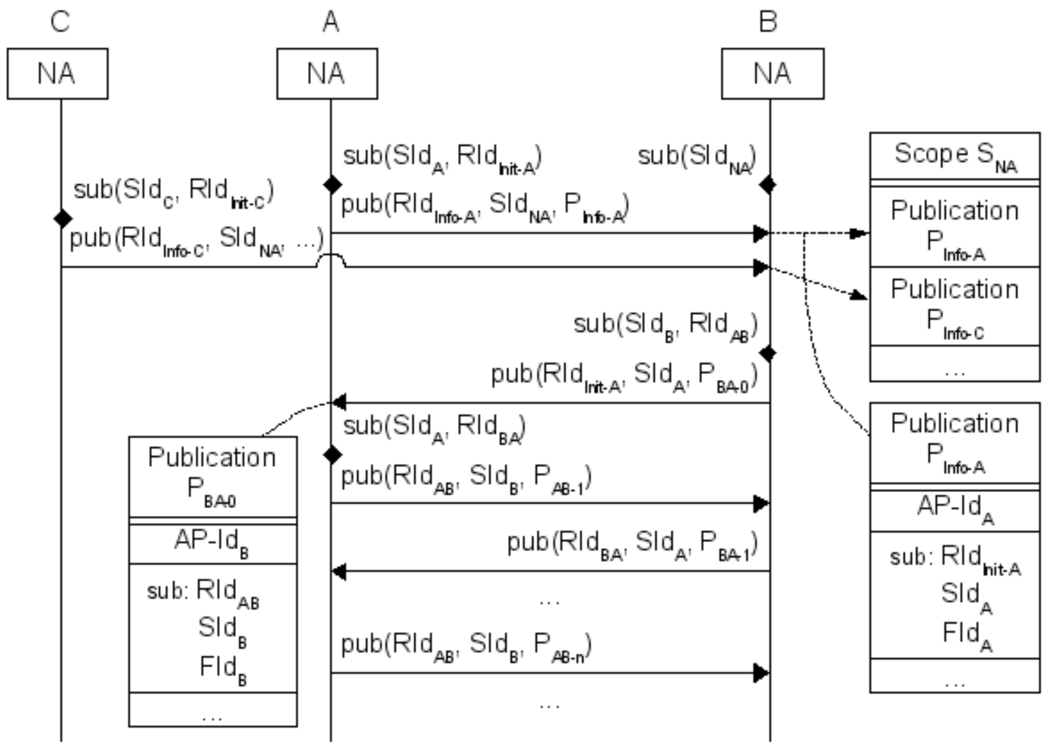

Figure 4.11 – Attachment example

We can begin by looking at node B which first starts collecting information about other attachment points by subscribing to scope S_{NA}. S_{NA} is assumed to be a commonly known scope that is used for bootstrapping network attachment. It is identified by SId_{NA}. This subscription actually implies that B is willing to receive information labelled with a corresponding FId, and that the network attachment component can be notified when new APs become available. Consequently, B receives publications $P_{Info\text{-}A}$ and $P_{Info\text{-}C}$ from nodes A and C respectively. The figure also shows that such publications can contain, for example, attachment point Ids, in addition to subscription information (e.g., an explicit {RId, SId, FId} set or data for generating them as needed) for attachment initiation messages. Nodes A and C have previously subscribed to those Ids. Furthermore, many other information elements can, of course, be included in these messages.

In this case, B chooses to attach to node A. It does this by publishing an initiation message ($P_{BA\text{-}0}$) with $RId_{BA\text{-}0}$ in scope SId_A. Similarly as above, subscription data for A-to-B control traffic is included in this message, along with other information, such as an authentication request, for example. A responds with message $P_{AB\text{-}1}$ which can include a specific subscription for B-to-A messages, and thus the two nodes have established the two-way control channel discussed previously.

The figure also shows that after exchanging a number of messages, the initial attachment phase is complete when A publishes P_{AB-n}. At this point, network communication to nodes in networks between the two nodes should be completely enabled. If we, for example, assume that B is a client node and A and access provider, C might have received a cryptographic token proves that a contract for using network services has been established[5]. In addition, B could have received additional identifiers needed for contacting essential network elements, such as rendezvous points. Moreover, both nodes now know the characteristics about the link between A and B, as well as the agreed service level, which can be possibly be utilized by the topology and forwarding functions. Configuration of these components, as well as other communication between them and the NA component, is of course assumed to take place through the component wheel.

As mentioned above, the control channel, that is, the subscription state for conveying attachment data between two nodes, can remain after the initial attachment exchange, and be used for updating various data, including the subscription state itself.

[5] *Such a contract can, e.g., be used for authorization on individual publish/subscribe operations, or in fast attachment point changes.*

5 Design Considerations

While we focussed in Section 3 and 4 on the design for the overall architecture and its core components, the following section addresses design considerations that are not specific for a particular component but crosses many areas in our architecture. These considerations specifically concern mobility, security, and service discovery.

5.1 Mobility

The publish/subscribe paradigm has been regarded as especially suitable for the support of mobile communications. As a product of the indirect character of communication, anonymity and asynchrony of pub/sub networks allows them to adapt quickly to frequent connections and disconnections, making them advantageous for mobile environments [Hua2004]. Furthermore, the shift towards an information centric paradigm means that multicast will become the norm rather than the exception, as it is the most appropriate mechanism for the efficient delivery of content to groups of subscribers. Hence, multicast assisted mobility [Fes2007] re-emerges as a promising research direction, albeit in a new context.

Typically, mobility support is realized by handoff management mechanisms. A simple way to cope with a handoff on the subscriber and publisher level is for subscribers to re-subscribe to those publications they are subscribing to, and for publishers to re-establish their publications at the new location. The rendezvous service is responsible for maintaining the proper set of rendezvous points and multicast forests. This requires that the routing and forwarding topology is updated accordingly during and after handoffs. The main problem then becomes how to optimize the rendezvous system for this operation. Apart from resuming communication from a new point of attachment to the network, it is also desirable to achieve reliable, mobility-safe communication [Tar2007]. This basically entails the in-order delivery of all publication items to all their subscribers. Per publication sequence numbers are considered to be a convenient technique for guaranteeing in-order delivery. On the other hand, publication loss is prevented by buffering publications during a handoff procedure.

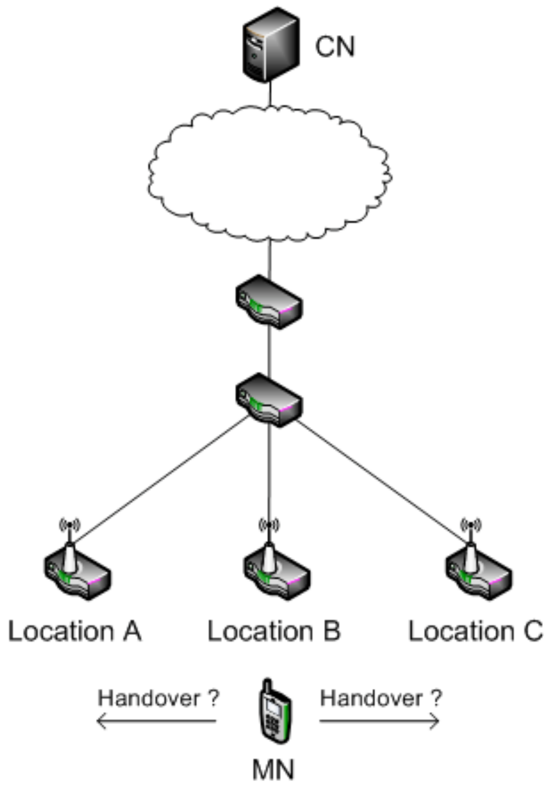

Figure 5.1 – Multicast assisted mobility (*prefetching*)

Apart from these above high-level mechanisms, an extended set of enhancements will be considered in supporting mobility in PSIRP. As already mentioned, the multicast nature of the forwarding substrate is expected to assist mobility. In multicast assisted mobility, termed as prefetching [Bur2004] in pub/sub networks, multicast trees are created in order to distribute data around the area that a mobile user resides in (see Figure 5.1). The purpose of this distribution is to reduce the duration of the handoff procedure, since data will already be available in the new network by the time a mobile subscriber reaches this network or forwarding paths may have already been established for a moving publisher. The scope of this distribution can be either statically determined, for example, data are forwarded to all neighbouring areas, or based on the expectation of possible movements of the mobile node. The cost of this optimization consists of the resources consumed for the proactive handoff preparation in multiple areas and/or for the establishment of reliable prediction mechanisms (e.g. statistics collection). Furthermore, routing and forwarding updates may be efficiently handled by exploiting the multicast nature of the routing substrate. Though dependent on the topological characteristics of the multicast trees, re-attaching to an already established forwarding tree may prove beneficial for the provision of fast handoffs.

As another simple optimization of a handoff mechanism, we observe that if some RIds are already subscribed or published in the new network of a mobile node, the re-establishment of forwarding paths is avoided. This requires the network to be able to filter and merge existing with new subscriptions/publications.

5.1.1 Router and Network mobility

Similar mechanisms can be used for router and network mobility, although in this case function of re-subscription and re-publishing becomes a function of aggregating a set of publications and subscriptions. With this, the mobile router that receives a subscription from its mobile network becomes a rendezvous point for the nodes it serves. When the mobile router moves, it simply updates the publish and subscribe requests to the rendezvous system, which updates the forwarding tables in the network to deliver data to the new location.

5.1.2 Overlay multicast assisted mobility

In order to investigate the intrinsic characteristics of publish/subscribe and multicast network architecture with respect to mobility support we have initially focused on an overlay variant of the PSIRP paradigm based on Scribe [Cas2002a]: an overlay publish/subscribe system providing multicast routing. In this first approach we have also attempted to take advantage of specific characteristics of the underlying Distributed Hash Table (DHT) scheme [Row2001] and further explore the potentials of an overlay realization of the PSIRP vision.

Overlay routing

Scribe is based on Pastry [Row2001], an efficient and scalable DHT substrate. Unlike other DHT schemes, Pastry attempts to employ proximity metrics, such as the number of IP hops or the round trip time towards other nodes, when choosing among the potentially large number of DHT nodes that may relay the data in question. Due to the use of proximity metrics the average distance a message travels does not exceed 2.2 times the distance between the source and the destination in the underlying network [Cas2002b]. Scribe maps the name of each multicast group to an identifier and makes the node responsible for that identifier the rendezvous point for that group. A subscriber to a group issues a Scribe JOIN message towards the RV of the multicast group. As the JOIN message propagates, reverse path routing state is established until a node already in the tree is found, thus forming a multicast tree rooted at the RV point. When the RV receives a publication it forwards it to all nodes that have expressed their Interest for this publication.

One of the main reasons for choosing this particular overlay multicast mechanism is that multicast routing state is maintained in a completely decentralized fashion: each node in a tree is only aware of its immediate ancestors and descendants, eliminating that way the signalling traffic needed in order to maintain global state information. Hence, in a highly dynamic environment, with a multitude of mobile nodes and where group membership (i.e. subscriptions) not only changes due to content-related reasons but also due to mobility, state management is simplified in favour of scalability. Furthermore, according to the Route Convergence (RC) property of Pastry, the distance travelled by two messages sent to the same key before their routes converge is approximately equal to the distance between their respective source nodes in the proximity space. This property is of particular importance in mobile environments as it will be argued and analytically demonstrated in the following sections.

Mobility Support

In this architecture every router participates in Pastry and can serve Scribe multicast trees as well. Every router is assigned a unique ID and hence, a unique position in the identifier space. On the other hand, (mobile) end nodes neither participate in overlay routing nor carry an IP address. This clearly reflects the target of breaking the end-to-end semantics of today’s communication. Every node is directly connected to an overlay access router (OAR), which is the router providing access to the overlay network and the multicast communication substrate. Mobile nodes are connected to OARs through their currently associated access point (AP). APs may act as simple bridges to the wired part of the network, may form groups connected to a single OAR so that link layer mobility (roaming) is provided in certain parts of the network or they can even act as OARs themselves. For simplicity, we will consider the first option in the reminder of this section.

Whenever a (mobile) node wishes to act as a publisher, it simply delivers its publication to its OAR, which is then responsible to deliver it to the proper RP. Whenever a (mobile) node wishes to subscribe to a publication it sends a subscribe(RId) message to its OAR. When an OAR receives a subscribe(RId) message, for the first time, it issues a Scribe JOIN message in order to join the multicast group of the publication. What actually changes with mobility is that as mobile nodes (MNs) move from one AP to another it is possible that they will change their OAR as well, performing a handoff. In this case they must inform their new OAR about the publication they are interested in. Of course, there is always the case that an OAR is already a member of the multicast tree for a publication a MN is interested in. This may happen because another end node that resides on that OAR has already subscribed for the same publication or because, due to the overlay nature of Scribe, this OAR has become part of this multicast tree in order to serve another OAR as a forwarder. In that case the OAR has only to forward the publication towards MN.

The way an OAR leaves a multicast group is by sending a LEAVE Scribe message after a period of time from the moment it anticipates the need to leave the multicast tree. This anticipation is based on the state of the association of each, currently attached MN with the wireless access point. In practice, an OAR decides to schedule the LEAVE Scribe message for a specific group when the last mobile member of that group has disassociated from the AP. The reason why an OAR does not immediately leave the multicast group is two-fold. First, whenever an OAR leaves a multicast group the multicast tree for this publication is (partially) destroyed so it will not be possible for a moving MN to take advantage of the multicast nature of Scribe and the route convergence property of Pastry, unless, of course another mobile node in the same area has subscribed to the same tree.

The support for mobility in the described overlay architecture is primarily based on the following two characteristics. First, by employing multicast as the main routing mechanism, routing information updates can be localized. This means that when a MN moves from a OAR to another, traffic can be diverged to the new point of attachment at the lowest common ancestor (CA) of the two visited OARs in the tree. Second, due to the route convergence property of Pastry, this CA is expected to be close enough to the new point of attachment so that the routing update can be performed without large delays. The intuition here is that when a MN moves from one access point to another, these access points are logically expected to be close enough in the proximity space, so the re-subscribe message of the MN will meet the CA after travelling a short distance in the proximity space.

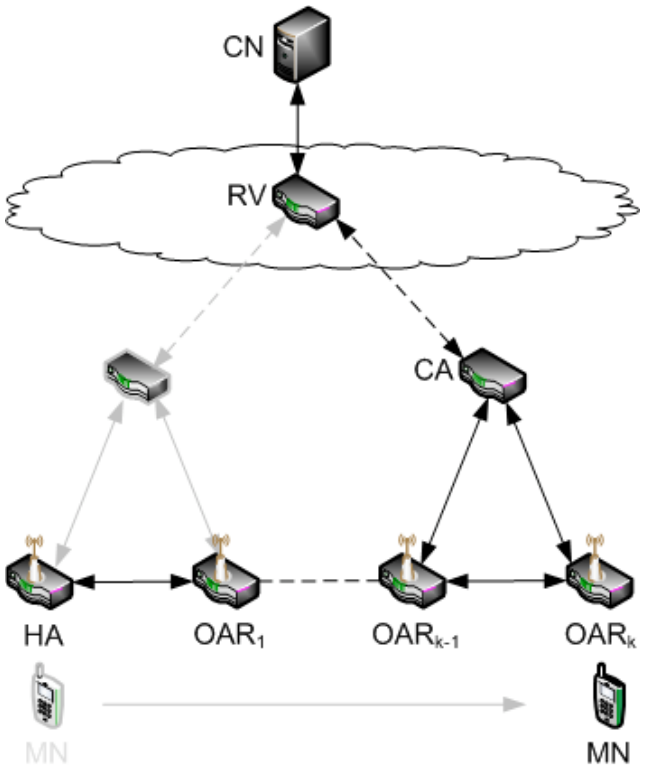

Figure 5.2 – Overlay multicast assisted mobility example

5.2 Security Design

Our security goals include confidentiality, integrity, availability, and accountability. Although we must take these goals into account at multiple levels, this section mostly concentrates on network level security. Potential targets of the attacker include:

- End-nodes like a publisher and subscriber
- The underlying network infrastructure including physical links, forwarding nodes, and nodes which handle network services, like rendezvous points
- Network services in general, including forwarding, rendezvous, caching, network attachment, and compensation.

The system should be resilient and secure against internal attackers. We must assume that any entity within the network can be a potential attacker, including forwarding and rendezvous nodes, which are a basic part of our architecture.

Our goal is to investigate multiple security choices for PSIRP. Different parts of the PSIRP system like rendezvous and forwarding may use different security solutions.

5.2.1 Direct and Indirect Cryptographic Associations

It is important to notice that the security mechanisms applied depend on the creation time of content. Therefore, identifiers associated to content may be generated in different ways depending on whether the content exists in the system or not at the time when the related meta-data is accessed. For example, it is possible to compute a cryptographic hash over a movie clip and use that hash for the identifier of the clip. On the other hand, it is not feasible to compute in advance a hash over a full content of a real time voice call. Still, we have to offer meta-data identifiers (i.e. container handles) for the subscribers to the channels and to full content in order to support accessibility and reachability, independent of the usage scenarios.

We can divide the association between identifiers and data objects into two groups based on the type of identifiers. The groups are: direct cryptographic and indirect cryptographic association. The direct cryptographic associations can be used with fixed data objects while the indirect association is useful, e.g., in the case of data channels.

In the direct association case, there is a direct mapping between RIds and content. Typically, when the content is modified the corresponding identifier is also changed. For example, cryptographic one-way hash values that are computed over a block of data, like SHA-256, provide this kind of property. We note that direct cryptographic associations can only be established when the content is created before it is accessed. The benefit obtained is that we can guarantee the integrity of the original data object. However, the security of the approach relies heavily on a trusted rendezvous mechanism.

In the case of real-time media, we can use indirect cryptographic associations, like public-key techniques. A public key can be used as part of a RId or a SId. The trust is based on the association between a public-key pair and a priori known legal entity, like a person or a company. An essential difference compared to the direct association is that the identifier is not, by default, changed when the content is modified. In a way, it is possible to use the same public key pair together with additional labels to name different kinds of contents. To uniquely identify a SId (a container) or a RId, we can associate it to a public key pair. Basically, it is possible to concatenate a public key with some other label that also may or may not have cryptographic properties. This evidently results in certificates that associate the public key with the label.

Using certificates as primary identifiers allows us to support different kinds of trust models in the system. Basically, it is possible to allow each host to self-sign its certificates. Another alternative is to deploy third parties that sign the certificates and authorize other hosts to use the signed identifiers.

Information Freshness

When the same content is stored at multiple caches, it is challenging to keep all caches up to date. Version control is an interesting research area from the security point of view, because old and outdated information in the system may result in loss of time and money. One way to overcome the challenges in the version control would be to define a home location for each information object. However, this easily leads to a solution where the home is a single point of failure, and this may open the system for DoS attacks. On the other hand, when the information is replicated to multiple locations in the network, it becomes unclear who is the owner or acts as a trust anchor for the original content.

One viewpoint to the version control problem is to consider that a SId (and the related public-key pair) can be used to associate together a set of different versions of a document. It may also be possible to consider that a new version of a document is just a delta compared to the original one. The approach requires usage of algorithmic identifiers to associate subsequent RIds of a document together. An algorithmic identifier denotes that the subsequent identifiers can be associated together in a provable way. For example, Lamport's one-way hash chains provide a mechanism that can be used for algorithmic identifiers.

Through-the-Stack Security Solution

The access technology and maximum transmission units (MTUs) set the limitation for the size of chunks that can be transmitted over physical communication media. The content must be divided into data chunks that are transmitted in packets over the network between hosts. The data chunks must be strongly associated with the RId of the original full content and SId to provide integrity protection. In addition, each data chunk should have its own RId to support network coding, distributed caching and per packet integrity protection.

It seems that the security mechanisms that are applied to SIds and full content can also be used with data chunks. This means that the same public-key pairs can be used to sign individual data chunks and full data objects. On the other hand, it may be feasible to compute a hash of each data chunk and associate the subsequent data chunks with each other using algorithmic identifiers. At this level, the algorithmic identifiers can be used together with public-key signatures to authenticate packets, e.g., belonging to the same voice call or a movie clip.

A Rendezvous Mechanism based on Direct Cryptographic Association

These direct and indirect cryptographic associations are also coupled with the rendezvous architecture security design. The rendezvous system may act as a trust anchor and a trusted third-party both for publishers and subscribers. Another alternative is that the publisher and subscriber have a trust relationship between them and the RId is acquired via some out-of-band mechanism. However, the out-of-band mechanism must also be protected.

The main advantage of the direct cryptographic association is that it does not always require a trust relationship between the content subscriber and publisher. Basically, the approach makes it possible to verify the integrity of the original content at the subscriber's end. In this case, delivering the RId to the subscriber requires an integrity-protected channel between the publisher and the rendezvous system and between the subscriber and the rendezvous system.

Both the publisher and the subscriber trust the rendezvous system to map the subscriptions to the publications in an accurate way. Basically, the rendezvous adds an indirection layer to the architecture. Note that each in-direction layer must be protected with a security mechanism.

A Rendezvous Mechanism based on Indirect Cryptographic Association

When a public-key pair is associated to content the resolution mechanism is responsible for mapping the subscriptions to the right public keys. Also in this case, the resolution mechanism adds an additional indirection layer to the system. From the subscriber's point of view, the situation is a bit different compared to the previous one because the signature is not directly associated to the content. Instead, the publisher may sign any kind of content using the same public-key pair. This was earlier called an indirect cryptographic association. In a way, trust is based on the assumption that the private key is possessed by a well-behaving host. In other words, the information object has an 'owner'. Earlier, with the direct cryptographic association, the subscriber did not need to care about the publisher's trustworthiness. In a way, the object did not have an 'owner'.

Technical solutions and their consequences for security

The basic security operations, integrity and confidentiality protection, can be implemented with different kinds of cryptographic mechanism. These mechanisms are here divided into public key, one way hash and watermarking techniques. However, the watermarking techniques are still for further study.

The public key techniques can roughly be divided into public key and threshold cryptography. In both cases, the length of the public key and certificate material carried in packets cause tradeoffs in the protocol design. Basically, there is a tradeoff between the length of the carried security header and the payload size. To maximize the payload length in the carried messages it is possible to use ECC techniques where the length of the public keys and signatures is shorter compared to traditional RSA public key material. In addition, it is possible to increase the computational speed of public key cryptographic operations by using specific FPGA cards at hosts.

Currently, Packet-Level Authentication (PLA) is an ECC-based approach that signs each packet transmitted in the network. It may also be possible to protect the integrity and confidentiality of packets using traditional symmetric cryptography. Without an explicit host identifier namespace, it is not obvious how to authenticate publishers and subscribers to each other. On the other hand, each private key that is associated to a data object is stored at some host. Therefore, the public key technology may generate a shadow host namespace.

It is also evident that the rendezvous mechanism requires authorization in one form or another. This kind of requirement results easily in non-scalable PKI solutions. To avoid this kind of architectural design choice, it may be possible to replace traditional authorization certificates with threshold cryptography in several parts of the architecture. In a way, the private key material is distributed into multiple places in the system and thus increases the robustness of the system. In public key based approaches, the owner of the private key works as a trust anchor and owner for the corresponding data object. Using threshold cryptography, it is possible to design an architecture where the content has multiple trust anchors.

In addition to public keys, one-way hash values are useful for generating primary identifiers for data content. Lamport's one-way hash chains can be used to bind subsequent data objects together. It may possible to apply the main principle of TESLA to bind different versions of a mutable object together or to provide lightweight integrity protection of packets at the link layer.

Finally, the freshness of the data is a hard problem from the security techniques point of view. When data and related meta-data are securely time-stamped, the whole system depends on synchronized clocks. It may turn out that this is too tough a requirement. However, to what extent this time stamping becomes part of the resolution system is not yet determined.

5.2.2 Network Packet Authentication and Access Control

To meet the outlines security requirements we require the ability to determine that the packets travelling across the network are both authentic, and have not been tampered with by unauthorised parties. Authentication can be performed by parties at the edge of the network, although this has several weaknesses:

- Signatures over the payload of the packet cannot protect the integrity of the routing information in the network header. Potential attackers may tamper with routing information or replay previous packets to alternative destinations
- Malicious packets may only be detected after they have traversed the network. This enables Denial-of-Service attacks to be launched on both network and end host resources.

By providing authentication on the packet level as part of the PSIRP architecture we allow any legitimate party in the network to test whether a packet is authentic and take appropriate action (e.g., dropping the packet) if it fails such checks.

To check authenticity, we do not need to disclose the identity of the originator of the packet. Instead, it is sufficient to determine that the packet has been sent by a party authorised by another party with whom we have a trust relationship. This can protect confidentiality (or privacy of living persons) while allowing different forms of accountability. In this manner we may authenticate that the packet originates from a publisher who has an account with their ISP, or alternatively authenticate that the packet comes from a user within an organisation.

Such authentication is applied to all packets on the network, whether they come from publishers and subscribers, or from network components and services (for example packets between rendezvous service providers).

After accepting that authentication on the packet level has a role in network security, the question becomes as to what properties of a genuine packet we are trying to authenticate. In order to answer this question, we need to consider some of the actors in the publish/subscribe network and some of their relationships.

Subscriber

A subscriber to information wants to ensure that any packets it receives are genuine packets in relation to its subscription (that is published to the same identifier and containing content corresponding to that identifier). It also wants to ensure that it is not subject to denial of service attacks (using genuine content and identifiers in order to overwhelm its resources). To do so the subscriber needs to be able to check that:

- The identifier relates to a current subscription
- The content and identifier have not changed since publication
- The packet is not a replay of an earlier packet (to the current or previous subscribers)
- The publisher is trusted to create content for that identifier

This last point can be considered in two parts:

- Is the publisher trusted by the subscriber?
- Does any controller of the identifier trust the publisher?

The subscriber may know of, and directly establish trust with the publisher. Alternatively, the subscriber may trust an intermediary such as the ISP of the publisher. Such an intermediary may be able to vouch that the publisher has not abused other subscribers (by sending spam or performing Denial-of-Service attacks), and thus can perform a powerful additional role.

We also see that the notion of the identifier controller may not be the same as either the publisher or subscriber. For example, one publisher may create and control an identifier to which it delegates publication rights to other publishers. Alternatively, the controller many be one of a number of subscribers, or simply a third party management system. In cases where the identifier controller is simply either the publisher sending the packet or the subscriber desiring the check, the subscriber simply need to trust the publisher.

Since the subscriber is the ultimate destination for the packet, preceding parties such as network forwarding providers can undertake these checks on behalf of the subscriber. This can be advantageous to both the forwarding network and the subscriber since there will be financial benefits to not sending a malicious packet across its own network, or potential further benefits from not instructing onward networks to deliver the packet.

Forwarding Network

The forwarding network may inherit the concerns of subscribers, particularly if the subscribers are its direct customers (i.e., the forwarding network is an ISP). It also has its own separate concerns over the authenticity of the packet.

In particular the forwarding network is concerned that it will receive appropriate remuneration for the forwarding of the packet. Such remuneration may come from direct cash settlement, or from reciprocal schemes and other incentives. Thus in forwarding the packet, the forwarding network needs to be sure that someone is accountable for the packet, whether this is the original publisher, or some intermediary who has inherited the accountability. In essence, it needs to ensure that someone is accountable for the packet.

The forwarding network also needs to protect its own resources. It is conceivable that publishers can launch Denial-of-Service attacks to overwhelm the forwarding and bandwidth resources of the forwarding network. To do so, an attacker may publish to existing subscribers or may also engineer subscribers to direct traffic patterns. Thus, the forwarding network needs to be able to trust:

- That the publishers are not deliberately attacking network resources
- That the subscribers are not deliberately attacking network resources

In a similar manner to subscribers, the forwarding network may place its trust in upstream network providers to perform these checks on its behalf, potentially gaining the advantage of the combined reports of multiple forwarding networks.

Rendezvous system

Rendezvous systems can be considered as a specialised case of subscribers. They use the forwarding network in order to receive subscription requests (from subscribers) or subscriber resolution requests (from publishers or their proxies).

As such, they share the concerns of subscribers (although for more specialist content such as subscription messages), but also share commercial and security concerns similar to those of the forwarding networks. In this regard, the rendezvous service needs to know that someone is accountable for the requests it receives, and also that publishers and subscribers (or any intermediating party) are not conducting Denial-of-Service attacks on its resources.

Denial-of-Service attacks on the rendezvous system can be conducted to directly attacking the communication, processing, or storage capabilities of the rendezvous system, or in order to attack peered rendezvous services or underlying forwarding networks through the propagation of control messages. Attacks on the rendezvous system may also have the motivation of damaging a publisher or subscriber who uses a particular identifier or rendezvous system. These concerns also hold for other network components such as the inter-domain topology resolution.

Publisher

While it is certainly true that the publisher of packets to the network may also elect to subscribe to information (such as flow control requests), there may also be a limited amount of packets that are sent back to the publisher through other network actions. For example, network attachment will result in some initial network information being transmitted to the potential publisher.

Another example of packets received by the publisher is in response to an advertisement to the rendezvous system. This advertisement allows subscription information to be pushed back to the publisher in order to control their sending of publications into the network. Such feedback may simply consist of a signal, that informs the publisher if there are no subscribers, or may provide more detailed information (such as a subscriber count, or potentially even subscription metadata such as flow rate requests). In terms of the architecture, the advertisement method can be viewed as a subscription to subscriber information for a particular Rendezvous Identifier. The rendezvous system is the publisher of information to the related 'subscriber info' Rid.

In these cases, the advertiser expects to receive genuine subscriber updates from rendezvous systems that participated in the advertisement process.

5.2.3 Packet Level Authentication (PLA)

Packet Level Authentication (PLA) [Lag2008, Can2005] is a novel way to secure networks on the network layer by providing availability and protecting the network infrastructure from several kinds of attacks, like denial-of-service (DoS) attacks. PLA is based on the assumption that per packet public key cryptographic operations are possible at wire speed because of new cryptographic algorithms and advances in semiconductor technology.

The main principle of PLA is to allow benevolent traffic while giving the ability to detect and stop malicious traffic as quickly as possible. Unlike traditional network layer security solutions, PLA provides the ability to nodes in the network to immediately detect attacks by checking the authenticity and integrity of every packet. In comparison, when traditional end-to-end security solutions like IPSec [Ken1998] are used, only the end point of the connection can verify the authenticity of the packet. In contrast to link level solutions, PLA allows every node to verify the packet independently without having to trust intermediate nodes that have previously handled the packet on the path from the originator. PLA aims to complement existing security solutions instead of completely replacing them.

A good analogy to the principle of PLA is a paper currency. Anyone can independently verify whether the bill is authentic simply by checking security measures inside the bill like watermark and hologram. There is no need to contact the bank, which has issued the bill. Using the same principle, PLA gives every node a possibility to check whether the packet has been modified, duplicated, or delayed without a previously established trust relation with the sender of the packet.

PLA header

PLA relies on cryptographic signature techniques, and it adds a separate PLA header to the packet. This PLA header contains all the necessary information for verifying the packet's authenticity and integrity. The structure of the PLA header is shown in Figure 5.3. Fields of the PLA header are explained below.

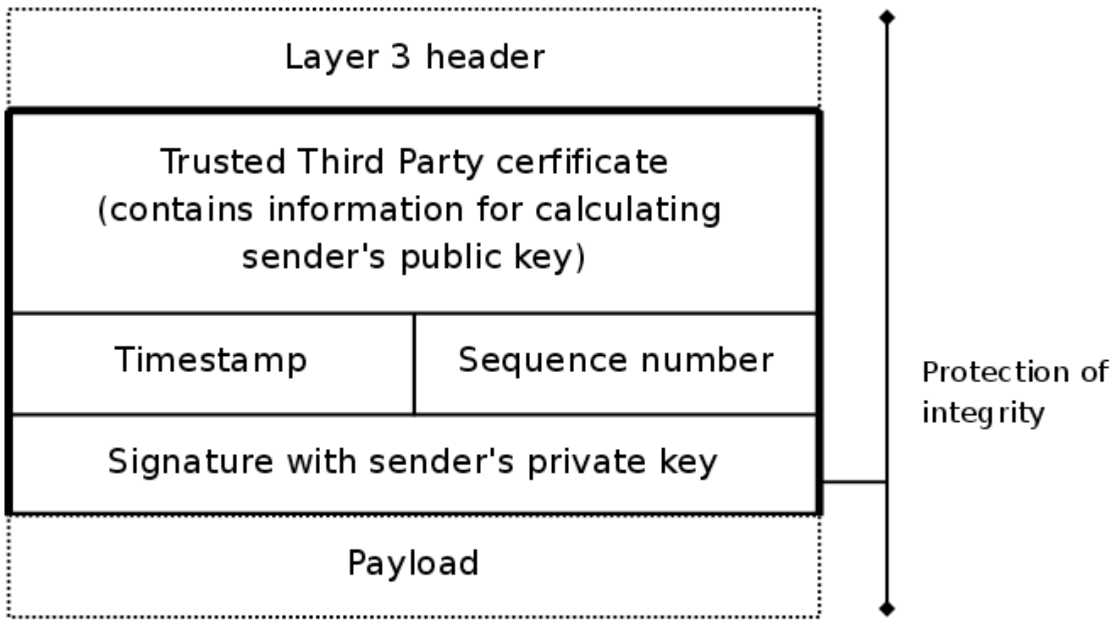
Layer 3 header
Trusted Third Party cerfificate (contains information for calculating sender's public key)
Timestamp
Sequence number
Signature with sender's private key
Payload
Protection of integrity

Figure 5.3 – The structure of the PLA header

The trusted third party (TTP) certificate corroborates the binding between the sender's identity and its public key. It also guarantees that the sender is a valid entity within the network and is authorized by some trusted third party. To reduce computational and bandwidth overhead, PLA utilizes identity-based implicitly certified cryptographic keys [Bru2007]. As a result, the sender's public key can be calculated from the TTP certificate. The sender's public key also introduces accountability, i.e., the sender cannot deny sending the packet.

The timestamp field makes possible to detect delayed packets. Such packets can be a sign of replay attacks.

The sequence number field contains a monotonically increasing number. It is used to detect duplicated packets.

Finally, the PLA header contains a cryptographic signature over the whole packet ignoring some IP header fields like hop limit, since these change during the lifetime of the packet. The signature protects the integrity of packet and guarantees that any modifications of the packet can be detected.

Cryptographic solutions and performance

To reduce bandwidth overhead produced by signatures and public key information, PLA uses elliptic curve cryptography (ECC) [Kob1987, Mil1985]. A 163-bit EEC key used with PLA offers the same cryptographic strength as a 1024-bit RSA key [Men1995]. As a result, the total length of the PLA header is only about 1000 bits and therefore PLA produces less than 10% bandwidth overhead with standard 1500 byte packets.

Public key cryptography is very computationally intensive, and this is the main reason why per packet public key cryptography has not yet been implemented on a large scale. This problem can be alleviated by using a dedicated hardware for accelerating signature verifications and generations. An FPGA-based proof-of-concept hardware accelerator for cryptographic operations has been developed for PLA [Jar2007, For2008]. While the FPGA is very suitable for development because of its flexibility, its performance is very low compared to the application specific integrated circuit (ASIC). However, Altera offers Hardcopy [Alt2008] technology to convert an existing FPGA design to a dedicated ASIC. According to simulation, such an ASIC built on 90 nm manufacturing process would achieve 850,000 signature verifications per second, which approximately equals to 10Gbps throughput with 1500 byte packets. An optimized ASIC on a modern manufacturing process would achieve a significantly better performance. As a result, PLA is scalable to high-speed core networks, as long as a dedicated hardware is used for accelerating cryptographic operations.

PLA and PSIRP

We consider PLA to be one candidate for securing the PSIRP networking functions. PLA is a logical choice since it is based on public-key cryptography and at least some PSIRP labels are derived from cryptographic keys.

One example is utilization of the PLA's trusted third party certificate mechanism for providing and enforcing access control to scopes. First, we assume that the scope is owned by an entity possessing a private/public key pair, and a scope identifier is derived from this public key through, for example, hashing. The user wishing to access the scope, will authenticate itself through some means, and as a result will receive a certificate from the scope. This certificate has the same structure as the PLA's TTP certificate and therefore contains scope's public key. Basically it will replace the TTP certificate present in a standard PLA header. Forwarding nodes will forward data traffic to the scope only if such certificate is present in packet's header, effectively preventing the abuse of resources. The basic PLA security mechanism will protect this certificate from tampering.

We plan to investigate further how PLA can be used to secure other functions of PSIRP.

5.2.4 Tesla

Timed Efficient Stream Loss-Tolerant Authentication (TESLA) [Per2002] is a protocol designed to authenticate broadcast and multicast traffic. Authors of TESLA consider per-packet cryptographic signatures to be too computationally intensive to use on a large scale. Therefore, TESLA bases its operation on loose time synchronization between sender and receiver, as well as through the use of one-way chains. A one-way chain is a cryptographic primitive in which every element S_n in the chain can verify all the elements S_k where $k > n$. The first element of the chain, for example, can verify all other elements, and is thus the last element revealed to end-users.

At the initiation phase, the TESLA client and sender loosely synchronize their clocks. Then, the sender splits the time into equal size intervals and assigns an element of the one-way chain to each interval. For each time interval, the sender computes a message authentication code (MAC) using the corresponding element of the one-way chain as a cryptographic key. Finally, the sender broadcasts the packet and reveals the value of the one-way chain after a known delay. The receivers must buffer packets until they receive the requisite values in the one-way chain that can be used to validate them. Even if the receiver looses a disclosed key, it can recover by using the keys that it will receive afterwards. Moreover, receivers discard any packet, which contains a MAC that was computed with a key that was already revealed since these packets may be forged. In this manner, a repeat attack is avoided.

Tesla with instant message authentication

TESLA requires that each receiver in the multicast or broadcast tree will have to buffer packets waiting for authentication and integrity verification. Thus, it multiples buffer demands on the one hand. On the other hand, assuming an adversary model that enables an attacker to inject fake packets (having an out-of-chain HMAC signature) during a disclosure interval, then these packets are also buffered for validation, and in this case for dropping. If the attacker is permitted to inject multiple packets, then receivers' buffers will be eventually overflowed. This scenario shows that pure TESLA is vulnerable to local and time-constraint DoS attacks, in the case where a bogus packet flooding occurs. The density of these attacks depends on the adversary capabilities, the buffer size, and the disclosure threshold. Since multiple receivers are involved in the packet forwarding tree, optimizing these parameters sounds tricky.

Some recently proposed methods are found to be more efficient for fast packet authentication. For instance, instant message authentication proposes a way to exchange receiver buffering with sender buffering. The sender stores the hash value of a later packet, to an earlier packet and hence as soon as the earlier packet is authenticated by the receiver, the later packet can be authenticated with the usage of the hash value, as well. To construct the packet for the message chunk M_j during the time interval Ti the sender appends the hash value of the message chunk M_{j+vd} which is $H(M_{j+vd})$,the MAC value (where MAC = (K_i,D_j)), and the key that decrypts earlier packets Ki-d where d is represents the predefined disclosure period. The following figure illustrates the construction of two consecutive packets. The earlier packet is P_j and the later packet is P_{j+vd}.

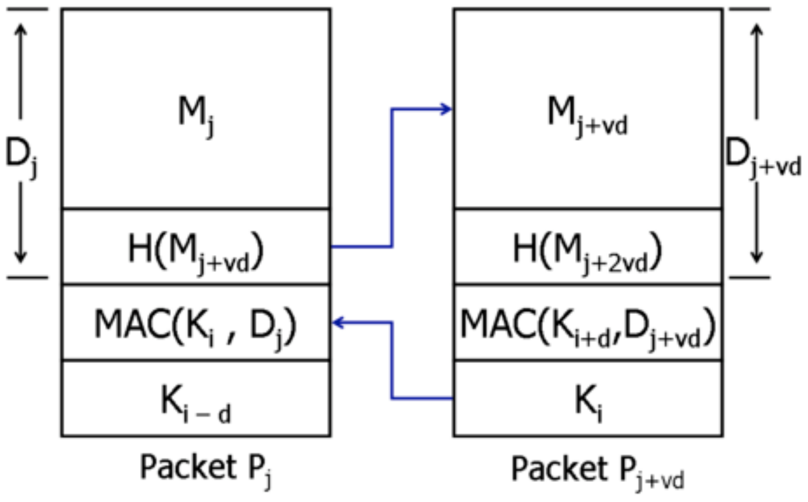

Figure 5.4 – Tesla

When packet P_{j+vd} arrives the receiver uses the key 'K_i' in order to authenticate packet P_j (this packet was buffered by the receiver). If packet P_j is successfully authenticated, the receiver can conclude that $H(M_{j+vd})$ is also authentic. The fact that $H(M_{j+vd})$ is authentic permits instant message authentication of the chunk M_{j+vd}. This TESLA variation adds flexibility for dynamic sending rate. This is achieved if the sender attaches as many as needed hash values of multiple future packets. Moreover, since there is no or less buffering needs as far as the receiver is concerned, this scheme is less vulnerable to DoS attacks.

5.2.5 Security and Algorithmic IDs

In Section 4.1.5, we introduced algorithmic identifiers (algIds) as a concept to build information collections (see Section 2) for both network and application functions, such as fragmentation, flow and error control as well as others. Producing and testing algorithmic IDs is a crucial part of their usage, e.g., for testing parent or child relations.

From a security perspective, we can consider that the functions used to produce and test algIds may take cryptographic keys (e.g. public and private key-pairs) as input. This may offer two advantages. For the purposes of integrity, a public key can be used to determine if a specific party was involved in producing the algId.

For the purposes of confidentiality, a public key may be involved during the generation of the algIds. This would mean that only a party possessing the corresponding private key would be able to test the relationship between the new algId and its parent identifier. The use of multiple keys could thus determine the ability of a recipient to navigate the graph of algIds and test different relationships. This can be used to protect the privacy of the algId creator (e.g., publisher or subscriber) against attacks that correlate different information identifiers in the network.

5.3 Service Discovery

Discovery of various kinds of entities is a fundamental feature for a network architecture. We distinguish between three different types of discovery, namely discovery of protocol suite modules and their options, network attachment, and discovery of services offered in the distributed environment.

The ability to discover protocol suite modules involves reflection on the part of the component wheel and the PSIRP APIs. The components of the wheel and helper functions need to be able to discover other modules, their dependencies, and possibly introduce configuration changes to the suite. Care needs to be taken in any possible reconfiguration of the component wheel to prevent undesired side effects. Nevertheless, dynamic configuration by utilizing reflection is a desirable feature for a protocol suite that aims to offer flexibility.

Network attachment includes the basic ability of a node to attach to the PSIRP network and to gain the communications capability. After the basic network attachment, the node needs to be able to discover services in the network in various scopes.

The discovery of services involves either directly querying for a service given its scope and name, or by enumerating a larger candidate set to find the correct services. Service discovery utilizes rendezvous points for obtaining information about scopes and services offered in them. Scope and application/service identifier enumeration is an optional feature that can be offered by RPs. The enumeration is requested by using a special field in the metadata of a request to the RP.

Service invocation and activation differs depending on the implementation. A service can issue periodic updates that can be subscribed using the RId obtained from an RP. Rendezvous system will coordinate the formation of the multicast structure that connects the subscribers and the service instance. When the process is completed, the client will be able to receive data from the service. Now, the service needs to inform the client how requests can be sent to the service, which is in the basic case done by periodic advertisements.

A service can also be contacted directly by publishing a message to a service specific RId that the service has previously subscribed. These two approaches can be directly implemented with the pub/sub API. Given that a service name and description can be resolved by the RP to a RId, the client can, depending on the mode of discovery, either publish a service request message or subscribe periodic advertisements from the service itself.

The former model is desirable because it avoids the overhead of periodic beacons. On the other hand, this approach is susceptible to various denial of service attacks. A number of clients can flood the public RId of the service. Given that there are multiple services under the same RId (destinations of the multicast structure), there needs to be a way to process the response messages. This approach may become challenging when a peer-to-peer service is used in which there are millions of nodes in the network that offer the same service.

The problem with large numbers of recipients can be solved either by aggregating responses using RPs or by using anycast communication primitive. Both require changes to the network architecture. Anycast is easy to realize given that the multicast graph can be used; however, with anycast it is not obvious what metric should be used to select the path that the anycast message will travel.

One way to avoid above problems is to use the first model, i.e., the periodic advertisements and combine that with advertisement caching. The RP can act as a proxy for periodic service advertisement messages and cache them according to time-to-live value defined in metadata. In the first phase, the subscription request to the service RId is not processed as a regular subscription but rather the RP simply terminates the subscription and returns the cached advertisement profile that includes the public RId of the service.

In any case, for any interaction to happen between the client and service, the client needs to have a response RId that the service can use to publish messages to the client.

6 Application Considerations

It is important to consider application design in the context of the PSIRP architecture. While we do not focus on the development of applications per se, this section addresses considerations for the design of applications in the context of our architecture. For this, we address issues of mapping application-specific identifiers onto our information concepts, discuss the issue of API from an application developer's points of view (in contrast to the architecture view of Section 3.1) and present a few examples (including our current demonstrator) to illustrate what example applications could look like. But we do recognize that many more considerations in the application and service space need consideration with a paradigm change like the one proposed by the PSIRP architecture.

6.1 Application Programmer Interfaces

Although PSIRP builds on entirely different foundations than the current Internet, it is a useful exercise to understand how application developers might view it as an increment on the current functionality—a major increment, that is. Here the architecture underlying that API is not of primary interest. Rather, the focus is on the functionality that will be available to application developers at roughly the level of the existing TCP/IP socket API.

Only a few of the most important communication services are addressed below. Protocols required mainly for control-plane functions are neglected. Moreover, many details are left unspecified, such as whether to support access control based on the identities of the communicating participants.

A common thread is the use of a self-certifying topology-independent identifier to join the communication session, whatever the semantics of the communication service. The self-certifying identifier (which may be a rendezvous ID) may have to be qualified by an additional identifier (a scope ID) for this purpose. The following discussion of different types of services assumes that this pair of identifiers has been combined into a single composite identifier.

6.1.1 1-to-N Message Stream

This service enables the owner of an ID to publish a stream of messages, to which others can subscribe using the ID. A subscriber joining an ongoing message stream receives messages from that point onwards. Message delivery is neither reliable nor guaranteed to preserve ordering.

This service is comparable to that offered by UDP over IP multicast. A receiver receives only those message streams to which it subscribes and, in addition, the owner of the ID can control who can receive these messages via the scoping mechanism. The service is intended to support end-user applications such as streaming media and multiplayer games, which typically use UDP, and often RTP. These applications tend to be complex, requiring additional protocols alongside the real-time message stream.

PSIRP may use cryptographic techniques to ensure that the received messages were in fact published by the owner of the ID.

The behaviour when the owner of the ID publishes messages under that ID at several different hosts is to be determined.

6.1.2 1-to-1 Bidirectional Connection

This service provides for connection-oriented communication in the form of a bidirectional reliable byte stream. It is intended to be largely compatible with TCP and to support applications that require request-response or conversational client-server communication. It differs from TCP in that the target of the connection is a topology-independent ID identifying a service rather than a host.

PSIRP will use cryptographic techniques to ensure that each connection made to an ID is a secure communications channel to a service endpoint created by the owner of that ID.

It would be highly desirable to support connection-oriented anycast when multiple hosts listen to the same ID, to allow for scalability, reliability, and DoS resistance.

6.1.3 1-to-N Document Distribution

This service provides for highly scalable distribution of content, either immutable or versioned. The owner of an ID can publish a byte string of arbitrary length under that ID, ranging from a small image on a web page to an entire OS image, followed by any updates to that byte string. Anybody subscribing to that ID will receive the most recent version of the published data.

PSIRP will use cryptographic techniques to ensure that the data received by anybody subscribing to an ID was in fact published by the owner of that ID.

The API may allow a subscriber to additionally receive subsequent updates to the data. Alternatively, a separate API might be provided for receiving notifications of updates, so that the subscriber can pull new versions adaptively (skipping versions when updates occur in rapid succession).

The behaviour when the owner of an ID publishes different data at different hosts is yet to be determined. An appropriate choice, considering these hosts as redundant origin servers that presumably replicate data at an application level, would be such that a subscriber receives the most recent version published at a host selected by the network, either arbitrarily or possibly based on topological criteria, e.g. avoiding inter-domain links where possible.

As an optimization, it might be of interest to allow publishing of immutable data under an ID derived from the data itself.

6.2 Mapping Application IDs onto Rendezvous IDs

As already mentioned in Section 4.1, application IDs can be resolved into network rendezvous IDs in various ways. While for some applications the AId to RId mapping process is immaterial, as long as a resolution mechanism such as a search engine exists, for many applications the various pieces of content exchanged are related in such a manner that it makes sense to translate such AIds to algorithmically derived RIds, so as to expose these relationships to the network.

As a first example, consider an application producing a stream of data, such as packets of consecutive voice samples (as in VoIP). In this case, the AId of each packet could be derived from the AId of the voice stream plus a sequence number. These AIds could be transformed to a set of algorithmically derived AIds using a function taking as parameters the RId of the stream plus the sequence number of the packet. The subscriber could then subscribe to the stream RId implying that it should receive all the derived RIds. The network in turn could take advantage of this exposition of the relationship between the data for the various purposes discussed in Section 4.1.

As a second example, consider a dynamic web page consisting of some main text and multiple embedded objects and links. In this case, the entire page could be identified by a single AId and each component could be identified by AIds derived from the main AId by adding sequence numbers. By mapping these AIds to an algorithmic set of RIds as above, the subscriber could subscribe to the entire page in order to receive all components, without having to first receive the main page, parse it, and then retrieve each object. It is also possible to generalize this to multiple levels by using a tree of RIds as in Section 4.1, to identify a set of pages, the individual pages, sections of each page, and finally the page components themselves.

As a third example, consider a bulk distribution application based on BitTorrent, where the content to be distributed consists of a set of separately identified (and verified) blocks, each of which is transmitted as a set of packets, not necessarily in sequence. In BitTorrent there already exists a hierarchical convention for the AIds of pieces and blocks, so this could be directly mapped to a tree of algorithmically derived RIds, with the RId of the entire publication at the root, the RIDs of the pieces at the first level and the RIds of the blocks at the second level (and maybe the RIds of individual packets at the third level).

6.3 Application Examples

The following sections present two application examples that show the usage of the application-level APIs in the context of web and BitTorrent like applications.

6.3.1 World Wide Web Functionality

Support for new communication protocols and URL schemes can be added to modern web browsers such as Mozilla Firefox as protocol extensions. As such an extension is installed into a web browser, the browser will continue working as before except that content such as web pages and images can now be fetched using the new protocol and URL scheme the extension added support to.

For this, the web functionality is implemented on top of PSIRP with the help of "psirp:" URL scheme. The format of psirp: URL scheme is as follows:

- psirpURL = "psirp:" [scope identifier] : [rendezvous identifier]

where [scope identifier] and [rendezvous identifier] are as defined in Section 4.1 of this document.

This URL scheme may be implemented as a Firefox extension, which works by subscribing to the content pointed to by the URL from PSIRP. Through the use of advertisement functionality within the rendezvous system the web server will detect the presence of subscribers and deliver the data of the publication to the Firefox web browser. Alternatively without such advertisements the web serve needs to subscribe to delivery requests (equivalent to HTTP GET messages). It is assumed that each content item such as a HTML page or an image on a web site will form its own publication, i.e., can be referred to by a "psirp:" URL. The "psirp:" URLs can be used the same way as any other URL. They can, for example, be typed into the browser address bar, they can be used as a "src" attribute in an image tag or as a "href" attribute in an anchor tag.

Although backwards compatibility for web applications can be provided by using the "1-to-1 Bidirectional Connection" API, the "1-to-N Document Distribution" API might be the most interesting of all PSIRP APIs for web developers. This "1-to-N Document Distribution" API allows for subscribing to not only the current data but also to the upgrades of it, i.e., it could be possible to create self-updating web pages using this feature without resorting to constant polling with Javascript.

6.3.2 BitTorrent-like Content Distribution

BitTorrent is a peer-to-peer content distribution protocol, in which content downloaders help the content originator to distribute the content while they are downloading the content themselves. The fact that the burden of content distribution is distributed among the downloaders makes BitTorrent scale to large numbers of downloaders. A content originator can thus publish content on BitTorrent without worrying about server bandwidth availability, even if the published content becomes surprisingly popular.

To a first approximation, BitTorrent-like scalable content distribution will be straightforward in a PSIRP network, even if it contains caches. Content can be published using the PSIRP "1-to-N Document Distribution" API. In PSIRP, the same way as in BitTorrent, a content originator can publish content without worrying about the load that the publication is going to cause on the publisher's own server, since caches will make PSIRP scalable the same way as BitTorrent. The first difference between PSIRP and BitTorrent is that in PSIRP not only subscribers, but also caches, can act as additional sources for the data, while in BitTorrent only the downloaders can act as additional sources. The second, and more important, difference is that the use of the "1-to-N Document Distribution" API will allow publications to be forwarded in multicast mode to many subscribers at once, regardless of the source used, unlike BitTorrent where data is forwarded in unicast mode.

In BitTorrent, user participation in content distribution is rewarded by an incentive mechanism called *tit-for-tat*, which tries to ensure that uploading to others results in higher download speeds than mere free-riding. PSIRP is currently lacking such an incentive mechanism that would reward the subscribers for acting as sources of the parts of the publications they have already received. The issue is that the use of the "1-to-N Document Distribution" API breaks the direct relationship between the endpoints upon which tit-for-tat is based, therefore the subscribers cannot be "punished" if they do not act as sources. Thus, implementing an incentive mechanism in PSIRP publish-subscribe environment remains an interesting topic for future research.

Implementing BitTorrent-like functionality is however more complicated for very large files, especially without caches. Transferring a very large file as a one single publication using one-to-many communication might not be the best option. Instead the file might be split into a number of sub-publications that would correspond to "pieces" in traditional BitTorrent. One master publication containing the list of identifiers of the sub-publications and other metadata would then act as ".torrent" file equivalent. Once a downloader receives a piece as a sub-publication, it may then re-publish the piece under the same rendezvous id. The rendezvous process would then connect future subscribers seeking this piece to the downloader, assuming that there are no other publishers nearby. Again, the download of each piece can be made more efficient by using the one-to-many forwarding capability of the PSRIP network. Where flow signalling would normally be used in a one-to-one network, for multicast delivery each piece may be further split into low flow rate layers, allowing each client to receive as many simultaneous layers as it desires. This option can of course also be combined with caching, with the caches serving as sources for any pieces they have already cached.

6.4 Current Demo

There are currently two demonstrations available as a result of the work within the implementation work package 3. This work implements the node architecture, based on the component wheel (see Section 3.4), that is outlined Section 4.2. As basic components, the demonstrations implement the publisher and subscriber functions as well as the (intra-domain) forwarding function. With that, basic PSIRP scenarios can be implemented, as shown in the following subsections.

6.4.1 Demo Scenario I

In the current demo, we have three different nodes that are: publisher, forwarder and subscriber. This is illustrated in Figure 6.1. The publisher may publish any kind of files, even binary content. The 'publish' helper application uses our native pubsub API. The application creates a RId for the content and publishes related meta-data in the default scope. In our case, the meta-data is forwarded through a forwarding node to the subscriber.

Once the subscriber receives the meta-data, it initializes a virtual memory object, i.e. an empty container, for the content. This empty data container is visible like a normal file in the host's directory (/pubsub/...). The directory hierarchy is divided into two levels below the /pubsub mount point. The second level directory names the SId and the third level directory the RId assigned to the specific scope. Finally, the content is readable from the file that is named by default as 'data'.

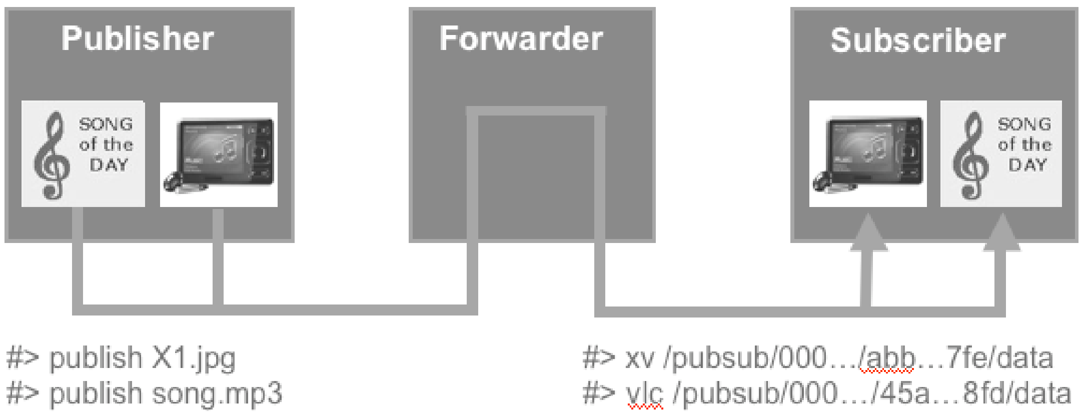

Figure 6.1 – Demonstration scenario I

Legacy (unmodified) applications are able to open the 'data' file as any normal file. The main difference compared to normal file system is that the pubsub file system uses a new kind of pager. Normally, the pager takes care of paging memory pages between swap-file system and the DRAM. In our case, the paging takes place between DRAM and our FUSE daemon. Now, when the application (e.g., the xv media player) opens a file in the pubsub file system and tries to access a memory address, a page-fault is created. This page-fault in turn triggers a subscription for the missing memory pages. In FreeBSD, each memory page is 4kb. However, to support smaller MTUs on the wire we subscribe the pages in 1kb chunks. In practice, the page-fault causes the pager to fetch multiple pages that results subscribing multiple chunks per page-fault. In other words, we have coupled the virtual memory management and paging with on-demand content subscription. This is illustrated in Figure 6.2.

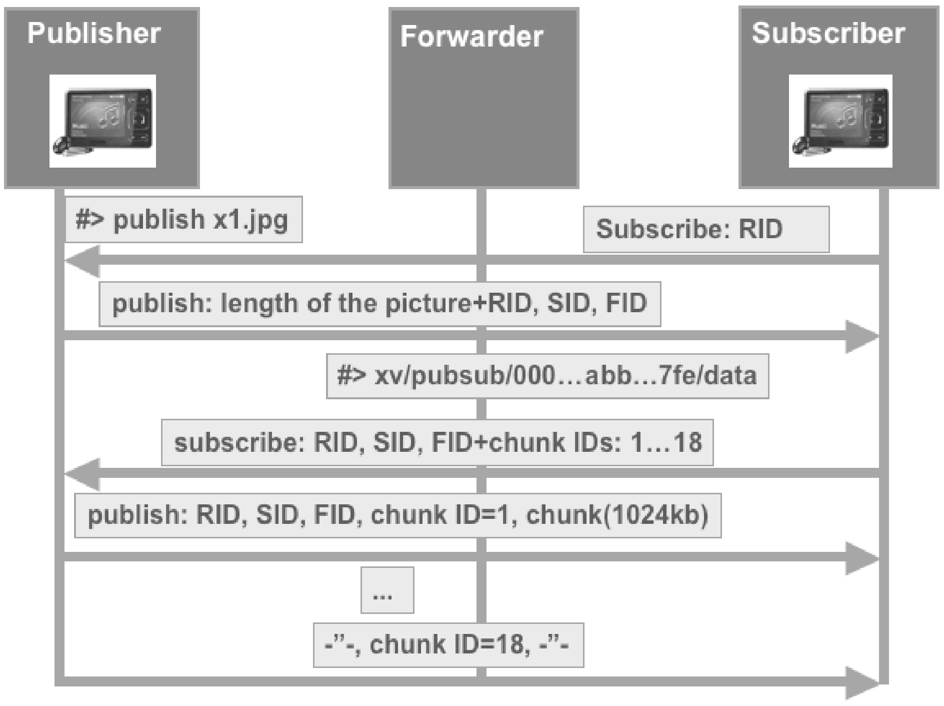
Publisher
Forwarder
Subscriber
#> publish x1.jpg
Subscribe: RID
publish: length of the picture+RID, SID, FID
#> xv/pubsub/000…abb…7fe/data
subscribe: RID, SID, FID+chunk IDs: 1…18
publish: RID, SID, FID, chunk ID=1, chunk(1024kb)
…
-"-, chunk ID=18, -"-

Figure 6.2 – Paging in our demonstration scenario

It is also important to note that the current prototype supports the presented zFilter forwarding (see Section 4.6.1). However, we do not yet have a working topology manager which makes it difficult to demonstrate the zFilter functionality with the current implementation.

6.4.2 Demo Scenario II

Typically, when an unmodified application opens a file it reads the content into memory at once. From the subscriber viewpoint, all the chunks are stored locally before the application views a picture or plays a music file. However, our second scenario demonstrates the on-demand memory access policy with URLs.

For this, we implemented an apache web-server at the subscriber side. The apache server directly accesses the /pubsub directory. The files in the directory can be expressed as URLs that can be given to the music player 'vlc'. The 'vlc' application understands that the content is reachable over the network and starts playing the content before it gets all the content. In practise, the apache-server reads the file in the /pubsub directory based on the received requests.

The following figure illustrates how we are able to stream music between PSIRP and IP networks when there is a apache proxy in-between the networks.

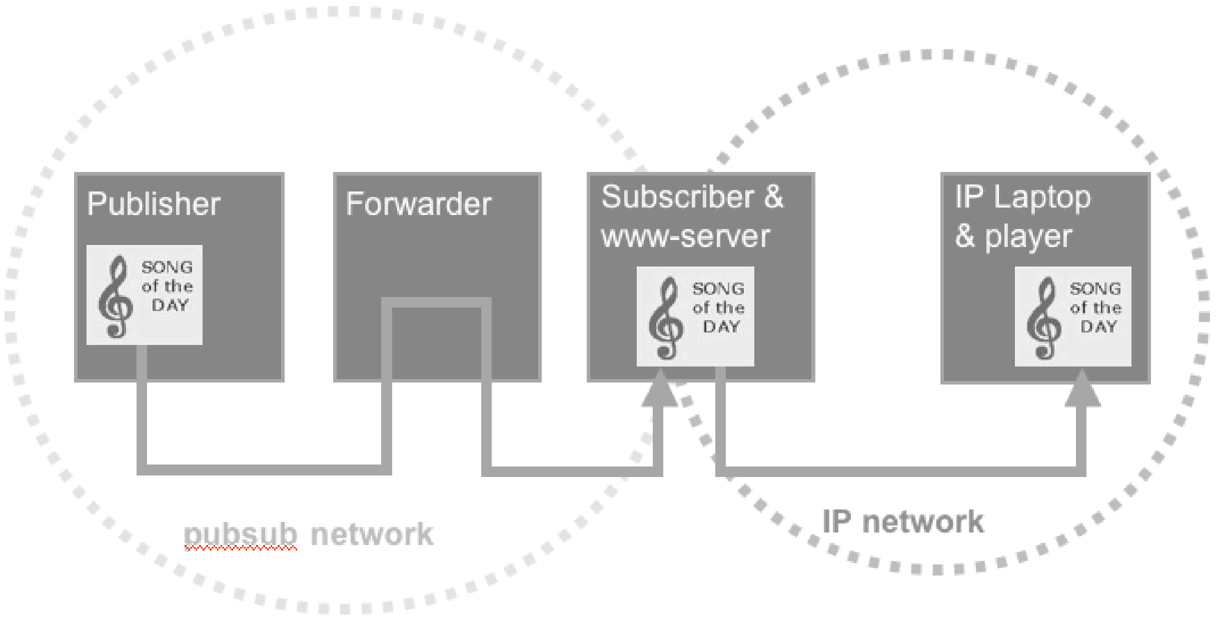

Figure 6.3 – Bridging PSIRP and IP networks

We are able to demonstrate music streaming with our current prototype. However, the prototype does not support any TCP like congestion control functionality in the pubsub network. To overcome this problem we limit the rate of the outgoing packet flow to a constant rate of 50ms.

7 Conclusions

This report extended our architectural work beyond the initial considerations that were presented in D2.2 [PSI2008d]. Based on our design methodology, which foresees a combination of bottom-up and top-down approaches, we combined our lessons learned in the implementation of a first PSIRP prototype with the continued work on extending the conceptual architecture and its main components. This approach has led to a clarification of concepts, presented in the design considerations and solutions that you can find in this report.

The concepts that were initially presented in D2.2, e.g., data and metadata, scoping, and the pub/sub communication paradigm, have manifested in clearly formulated information concepts, a node and network architecture with well-defined components as well as a refined service model. Feasibility of these solutions is represented in an evolving prototype implementation of some of the concepts but also in the detailed descriptions of the technologies, such as for inter-domain rendezvous, that can now be undertaken a rigorous evaluation with respect to scalability and economic feasibility.

The future work will concentrate on the development of solutions in various areas for which considerations have already started. Examples are inter-domain forwarding and topology formation as well as solutions for traditional transport functions like error and flow control. The solutions for these areas are expected to find entry in the final architecture deliverable of this project.

8 Terminology

Application identifier	Any higher-level identifier that applications may use. It is usually mapped onto a set of rendezvous identifiers for the data items relating to such application identifier.
Algorithmic identifier	An identifier that is determined algorithmically. It is used for rendezvous but also scope identifiers to enable *information collections*, i.e., sets of information items.
Caching	Process of temporarily storing data that is expected to be useful in the future in intermediate network elements.
Component wheel	The PSIRP layer-less network "stack" design, in which a number of network components communicate internally within a node using the publish/subscribe paradigm.
Domain	A managed network that is analogous to an autonomous system.
Forwarding identifier	An identifier used to associate a rendezvous identifier with a forwarding path.
Information item	The lowest entities of data being used in the PSIRP network. An information item is associated with a rendezvous identifier (RId) and assigned to one or more scopes.
Information collection	A set of information items. An *information network* forms a specific kind of information collection, also called *scope*. Other information collections can be formed via metadata information items, which include pointers to other information items. Also *algorithmic identifiers* can be used to assemble information collections, the collection being defined by the algorithm being used to determine the individual rendezvous identifiers of the information items in the collection.
Information network	A collection of information items under a single scope. An information network is identified by a scope identifier
Inter-domain routing	Inter-domain routing pertains to data delivery in the global network, typically spanning several domains. The inter-domain routing system is configured through the rendezvous process and takes into account any inter-domain policies in effect.
Intra-domain routing	Intra-domain routing pertains to data delivery within an administrative domain. Intra-domain routing is concerned with local policies.
Metadata	Data may also have associated metadata, which includes scoping information and other useful information either for ultimate receivers or network elements. This metadata in itself is data, i.e., it is associated with another rendezvous identifier. Metadata may be provided within a publication or as a separate data element with a separate rendezvous identifier.
Network	The process of obtaining connectivity with a PSIRP network.

attachment	
Publication	A self-contained data unit that has been made available using the publish primitive.
Publisher	An entity that uses the *publish* primitive to make data available.
Rendezvous	Rendezvous is the process of matching publishers and subscribers according to given rendezvous identifiers and initiating the transfer of data over the network within a given scope.
Rendezvous identifier	An identifier given to an entity by the rendezvous system in order to mediate between high-level identifiers, namely application identifiers, and lower-level identifiers, namely forwarding identifiers.
Scope	Scope defines an information network, assembling a collection of information items under that scope. A scope is labelled with a scope identifier (Sid). It determines the part of the rendezvous system that is used by the network. The three typical low-level cases are link local, intra-domain, and inter-domain. But scope can also be used to map higher-level concepts, like social networks, onto particular parts of the rendezvous system.
Service model	The model that is used by network elements to interface with the network. The PSIRP service model includes a low-level API, a channel model and higher-level service models
Subscriber	An entity that uses the *subscribe* primitive to request certain pieces of data.
Tussle	Tussle is defined as a conflict of interests, these interests being brought forward by players within a given communication scenario, either expressed as explicit constraints, requirements or other concepts of concerns. Design for Tussle [Cla2002] offered a novel insight for the proper design of such systems, i.e., providing guidance as to how a system ought to be designed so as to withstand and even incorporate a wide range of tussles.

9 References

NOTE: Some of the following references have been included as supplementary references from Deliverable D2.2 and may not have been cited in the body of Deliverable D2.3. With this, we intend to provide the reader with a comprehensive collection of relevant related work.

[Ahl2000] R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung, "Network Information Flow," *IEEE Transactions on Information Theory*, vol. 46, issue 4, pp. 1204-1216, 2000.

[Ake2004] A. Akella, S. Chawla, A. Kannan, and S. Seshan, “On the scaling of congestion in the internet graph,” *SIGCOMM Computer Communications Revue*, vol. 34, issue 3, pp. 43-56, 2004.

[Alt2008] Altera, HardCopy Structured ASICs: technology for business [online], 2008, available at: http://www.altera.com/products/devices/hardcopy/hrd-index.html [Accessed 21st May 2008].

And2007] D. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker, “Holding the Internet Accountable,” *HotNets-VI*, pp. 7–12, Nov. 2007.

[Bal2004] H. Balakrishnan, K. Lakshminarayanan,S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish, “A Layered Naming Architecture for the Internet,” *ACM SIGCOMM Computer Communications Review*, vol. 24, issue 4, pp. 343-352, Oct. 2004.

[Ban2001] A. Banerjee, J. Drake, J.P. Lang, B. Turner, K. Kompella and Y. Rekhter, “Generalized Multiprotocol Label Switching: An Overview of Routing and Management Enhancements,” *IEEE Communications Magazine*, vol. 39, issue 1, pp. 144-150, Jan. 2001.

[Bel1976] D. Belsnes, “Single-Message Communication,” *IEEE Transactions on Communications*, vol. 24, issue 2, pp. 190-194, 1976.

[Ber2001] T. Berners-Lee, J. Hendler, and O. Lassila, “The Semantic Web,” *Scientific American Magazine*, May 17, 2001, available at: http://www.sciam.com/article.cfm?id=the-semantic-web [Accessed on 15 July, 2008].

[Bri2004] R. Briscoe, “The Implications of Pervasive Computing on Network Design,” *BT Technology Journal*, vol. 22, issue 3, pp. 170-190, July 2004.

[Bri2005] R. Briscoe, A. Jacquet, C. Di Cairano-Gilfedder, A. Salvatori, A. Soppera, and M. Koyabe, “Policing Congestion Response in an Internetwork Using Re-feedback,” *ACM SIGCOMM Computer Communications Review*, vol. 35, issue 4, pp. 277-288, Sept. 2005.

[Bru2007] B. Brumley and K. Nyberg, “Differential properties of elliptic curves and blind signatures,” Proc. of Information Security, 10th International Conference - ISC '07, vol. 4779 of Lecture Notes in Computer Science, pp. 376-389, Springer-Verlag, 2007.

[Bur2004] I. Burcea, H.-A. Jacobsen, E. de Lara, V. Muthusamy, M.Petrovic, “Disconnected operation in publish/subscribe middleware,” *IEEE Computer Society Mobile Data Management*, 2004.

[Cae2006] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, I. Stoica, and S. Shenker, “ROFL: Routing on Flat Labels,” *ACM SIGCOMM Computer Communication Review*, vol. 36, issue 4, pp. 363-374, Sept. 2006.

[Cal2007] K. L. Calvert, W. K. Edwards, and R. E. Grinter, “Moving Toward the Middle: The Case Against the End-to-End Argument in Home Networking,” *6th Workshop on Hot Topics in Networks*, 2007.

[Can2005] C. Candolin, “Securing military decision making in a network-centric environment,” doctoral dissertation, Helsinki University of Technology, Finland, 2005.

[Cao2005] F. Cao and J. P. Singh, “MEDYM: Match-Early and Dynamic Multicast for Content-based Publish-Subscribe Service Networks,” *25th IEEE International Conference on Distributed Computing Systems Workshops*, Colombus, OH, June 2005, pp. 370-376.

[Car1996] B. Carpenter, *Architectural Principles of the Internet*, IETF RFC 1958, June 1996.

[Car2001] A Carzaniga and A.L. Wolf, "Design and Evaluation of a Wide-area Event Notification Service", *ACM Transactions on Computer Systems (TOCS)*, vol. 19, issue 3, pp. 332-383, Aug. 2001.

[Car2003] A. Carzaniga and A. L. Wolf, "Forwarding in a Content-based Network," in *Proc. ACM SIGCOMM 2003*, Karlsruhe, Germany, Aug. 2003, pp. 163-174.

[Cas2000] C. Castelluccia, "HMIPv6: A Hierarchical Mobile IPv6 Proposal," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 4, issue 1, pp. 48-59, 2000.

[Cas2002a] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "SCRIBE: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 100–110, 2002.

[Cas2002b] M. Castro, P. Druschel, Y. Charlie, and H. A. Rowstron, "Exploiting network proximity in peer-to-peer overlay networks," tech. rep., Microsoft Res., 2002.

[Cha2007] M. Cha, H. Kwak, P. Rodriguez, Y. Ahn, and S. Moon, "I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System". In ACM SIGCOMM IMC'07. Proceedings, pages 1-14, 2007

[Che2000] D. R. Cheriton and M. Gritter, "TRIAD: A New Next-Generation Internet Architecture," July 2000, available at: http://www-dsg.stanford.edu/triad/ [Accessed on 15 July, 2008].

[Cla1988] D. Clark, "The Design Philosophy of the DARPA Internet Protocols," *ACM SIGCOMM Computer Communication Review*, vol. 18, issue 4, pp. 106-114, Aug. 1988.

[Cla2002] D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden, "Tussle in Cyberspace: Defining Tomorrow's Internet," in *Proc. ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, New York, NY, Aug. 2002, pp. 347-356.

[Cla2003] D. Clark, R. Braden, A. Falk, and V. Pingali, "FARA: Reorganizing the Addressing Architecture," in *Proc. ACM SIGCOMM Workshop on Future Directions in Network Architecture*, Karlsruhe, Germany, Aug. 2003, pp. 313-321.

[Cla2007] D. Clark and M. Blumenthal, "The End-to-End Argument and Application Design: The Role of Trust," in *Proc. Conference on Communication, Information, and Internet Policy (TPRC)*, Arlington, VA, Sept. 2007.

[Cro2006] J. Crowcroft, "100% NAT - a DoS proof Internet," *IRTF End-to-End (E2E) mailing list*, Feb. 2006.

[Day2008] J. Day, *Patterns in Network Architecture: A Return to Fundamentals*, Prentice Hall, 2008.

[Egg2004] L. Eggert and J. Laganier, *Host Identity Protocol (HIP) Rendezvous Mechanisms*, IETF Internet draft, work in progress, Oct. 2004.

[Eug2003a] P.T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov and A. M. Kermarrec, "Lightweight Probabilistic Broadcast", in *ACM Transactions on Computer Systems (TOCS)*, vol. 21, issue 4, pp. 341-374, 2003.

[Eug2003b] P. T. Eugster, P. A. Felber, R. Guerraoui, and A. M. Kermarrec, "The Many Faces of Publish/Subscribe," *ACM Computing Surveys (CSUR)*, vol. 35, issue 2, pp. 114-131, 2003.

[Fea2004] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. v. d. Merwe, "The Case for Separating Routing from Routers," *in Proc.ACM SIGCOMM workshop on Future Directions in Network Architecture (FDNA 2004)*, New York, NY, 2004, pp. 5-12.

[Fes2007] A. Festag, H. Karl, and A. Wolisz, "Investigation of multicast-based mobility support in all-ip cellular networks: Research articles," Wireless Communications & Mobile Computing, vol. 7, no. 3, pp. 319–339, 2007.

[Fie2004] L. Fiege, A. Zeidler, A. P. Buchmann, R. Kilian-Kehr,and G. Mühl, "Security Aspects in Publish/Subscribe Systems," *Third Intl. Workshop on Distributed Event-based Systems (DEBS'04)*, Edinburgh, Scotland, UK, May 2004.

[For2004] B. Ford, “Unmanaged Internet Protocol: Taming the Edge Network Management Crisis,” *SIGCOMM Computer Communication Review*, vol. 35, issue 1, pp. 93-98, 2004.

[For2008] J. Forsten, K. Järvinen, and J. Skyttä, “Packet level authentication: Hardware subtask final report,” technical report [online], 2008, available at: http://www.tcs.hut.fi/Software/PLA/new/doc/PLA_HW_final_report.pdf [Accessed 25th July 2008].

[Gan2004] P. Ganesan, K. Gummadi, and H. Garcia-Molina, “Canon in G Major: Designing DHTs with Hierarchical Structure,” *ICDCS*, March 2004.

[Gao2001] L. Gao, “On Inferring Autonomous System Relationships in the Internet,” *IEEE/ACM Transactions on Networking*, vol. 9, issue 6, pp. 733-745, Dec 2001.

[Gill2007] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "YouTube Traffic Characterization: A View From the Edge". In ACM SIGCOMM IMC'07. Proceedings, pages 15-28, 2007.

[Gir1991] M. Girault, “Self-Certified Public Keys,” *Advances in Cryptology (EUROCRYPT)*, pp. 490-497, 1991.

[Han2006] M. Handley, “Why the internet only just works,” *BT Technology Journal*, vol 24, issue 3, pp. 119-129, July 2006.

[Hag2007] Haggle partners, “Deliverable D1.2: Specification of the CHILD-Haggle,” *HAGGLE*, Aug. 2007, available at: http://www.haggleproject.org/deliverables/D1.2_final.pdf [Accessed on 15 July, 2008].

[Ho2003] T. Ho, R. Koetter, M. Médard, D. R. Karger and M. Effros, “The Benefits of Coding over Routing in a Randomized Setting,” in *Proc. IEEE International Symposium on Information Theory (ISIT) 2003*, Yokohama, Japan, 2003, p. 442.

[Hua2004] Y. Huang and H. Garcia-Molina, “Publish/subscribe in a mobile environment,” Wireless Networks, vol. 10, no. 6, pp. 643–652, 2004.

[Hue2003] R. Huebsch, “Content-Based Multicast: Comparison of Implementation Options,” Technical Report, 2003.

[Jac2006] V. Jacobson, “If a Clean Slate is the Solution What Was the Problem?, *Stanford "Clean Slate" Seminar*, Feb. 2006, available at: http://cleanslate.stanford.edu/seminars/jacobson.pdf [Accessed on 15 July, 2008].

[Jar2007] K. Järvinen, J. Forsten, and J. Skyttä, “FPGA design of self-certified signature verification on Koblitz curves,” Proc. of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2007, Vienna, Austria, September, 2007, pp. 256-271, Springer-Verlag LNCS 4727.

[Jok2009] P. Jokela, A. Zahemszky, S. Arianfar, P. Nikander, C. Esteve, "LIPSIN: Line Speed Publish/Subscribe Inter-Networking", PSIRP project technical report TR09-0001, available at http://www.psirp.org [Accessed on 24 February, 2009], Jan 2009.

[Kat2006] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft, “XORs in The Air: Practical Wireless Network Coding,” *ACM SIGCOMM Computer Communication Review*, vol. 36, issue 4, pp. 243-254, 2006.

[Ken1998] S. Kent and R. Atkinson, Security architecture for the Internet protocol, IETF RFC 2401, November 1998.

[Key2006] P. Key, L. Massoulié, and D. Towsley, “Combining Multipath Routing and Congestion Control for Robustness,” in *Proc. 40th IEEE Conference on Information Sciences and Systems (CISS)*, Princeton, NJ, Mar. 2006, pp. 345-350.

[Kja2009] J. Kjällman, "Attachment to a Native Publish/Subscribe Network", Master's thesis, Helsinki University of Technology, Finland, February 2009.

Kob1987] N. Kobliz, “Elliptic Curve Cryptosystems,” Mathematics of computation, vol. 48, pp. 203-209, 1987

[Kom2005] M. Komu, S. Tarkoma, J. Kangasharju, and A. Gurtov, “Applying a Cryptographic Namespace to Applications,” in *Proc. 1st ACM Workshop on Dynamic Interconnection of Networks*, New York, NY, 2005, pp. 23-27.

[Kop2007] T. Koponen, M. Chawla, B. G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, “A Data-Oriented (and Beyond) Network Architecture,” *ACM SIGCOMM Computer Communication Review*, vol. 37, issue 4, Oct. 2007, pp. 181-192.

[Kri2007] D. Krioukov, K. C. Claffy, K. Fall, and A. Brady, “On Compact Routing for the Internet,” *SIGCOMM Computer Communcation Review*, vol. 37, issue 3, pp. 41-52, 2007.

[Lag2008] D. Lagutin, “Redesigning Internet - The packet level authentication architecture,” licentiate's thesis, Helsinki University of Technology, Finland, June 2008.

[Lak2006] K. K. Lakshminarayanan, I. Stoica, S. Shenker, and J. Rexford, “Routing as a Service,” Technical Report, University of California Berkeley, 2006.

[Leg2005] S. Leggio, J. Manner, A. Hulkkonen, and K. Raatikainen, “Session Initiation Protocol Deployment in Ad-Hoc Networks: a Decentralized Approach,” in *proc. International Workshop on Wireless Ad-Hoc Networks (IWWAN2005)*, London, UK, May 2005.

[Liu2005] H. Liu, V. Ramasubramanian, and E. G. Sirer, "Client behavior and feed characteristics of rss, a publish-subscribe system for web micronews", *in Proc. of ACMInternet Measurement Conference*, 2005.

[Mad2005] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TinyDB: An Acqusitional Query Processing System for Sensor Networks", in *Proc. ACM TODS*, 2005.

[Men1995] A. Menezes, “Elliptic curve cryptosystems,” CryptoBytes, vol.1, no. 2, 1995.

[Mil1985] V. Miller, “Use of elliptic curves in cryptography,” Proc. of the Advances of Cryptology – Crypto '85, Santa Barbara, USA, August 1985.

[Moo2002] T. Moors, “A Critical Review of ‘End-to-end Arguments in System Design’,” *International Conference on Communications*, vol. 2, pp. 1214-1219, 2002.

[Mos2008] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson, *Host Identity Protocol*, IETF RFC 5201, Apr. 2008.</span>

[Nek2003] M. Nekovee, A. Soppera, and T. Burbridge, “An Adaptive Method for Dynamic Audience Size Estimation in Multicast,” in *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, vol. 2816, 2003.

[Nik2004] P. Nikander, J. Arkko, and B. Ohlman, "Host Identity Indirection Infrastructure (Hi3)", in *Proc. Second Swedish National Computer Networking Workshop (SNCNW 2004)*, Karlstad, Sweden, 2004.

[Per2002] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "The TESLA Broadcast Authentication Protocol," CryptoBytes, vol. 5, no. 2, 2002, pp. 2-13.

[Per2004] R. Perlman, “Rbridges: Transparent Routing,” in *proc. 23rd Annual Joint Conference of the IEEE Computer and Communication Societies (INFOCOM 2004)*, Hong Kong, March 2004, vol. 2, pp. 1211–1218.

[Pie2004] P. R. Pietzuch, “Hermes: A Scalable Event-Based Middleware,” doctoral dissertation, Computer Laboratory, Queens' College, University of Cambridge, Feb. 2004.

[PSI2008a] PSIRP, *PSIRP Homepage*, available at: http://wiki.psirp.org [Accessed on 15 July, 2008].

[PSI2008b] D. Trossen (ed.), "PSIRP Vision Document", available at http://www.psirp.org/ [Accessed on 15 July 2008].

[PSI2008c] P. Jokela (ed), *PSIRP Deliverable 3.1: Prototype Platform and Applications Plan and Definition*, 2008.

[PSI2008d] S. Tarkoma (ed.), "PSIRP Deliverable 2.2.: Conceptual Architecture of PSIRP: including subcomponents descriptions", available at http://www.psirp.org/ [Accessed on 15 February, 2009].

[Raj2008] J. Rajahalme, M. Särelä, P. Nikander, S. Tarkoma. "Incentive-Compatible Caching and Peering in Data-Oriented Networks", Re-Arch'08 workshop at ACM CoNext 2008 conference, 2008.

[Ram2001] K. K. Ramakrishnan, S. Floyd, and D. L. Black, *The Addition of Explicit Congestion Notification (ECN) to IP*, IETF RFC 3168, Sept. 2001.

[Ram2004] V. Ramasubramanian and E.G. Sirer, "The design and implementation of a next generation name service for the Internet," *ACM SIGCOMM'04. Proceedings,* October 2004.

[Roa2002] A. B. Roach, "Session Initiation Protocol (SIP)-Specific Event Notification", IETF RFC 3265, June 2002.

[Rat2001] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Application-level Multicast Using Content-addressable Networks," *Lecture Notes in Computer Science*, 2233, 2001, pp. 14-29.

[Ros2001] D. Rosenblum, "A Tour of Siena, an Interoperability Infrastructure for Internet-scale Distributed Architectures," *Ground System Architectures Workshop*, Feb. 2001.

[Ros2002a] J. Rosenberg, *A Session Initiation Protocol (SIP) Event Package for Registrations*, IETF RFC 3680, Mar. 2004.

[Ros2002b] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, *SIP: Session Initiation Protocol*, IETF RFC 3261, June 2002.

[Row2001] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems," in *Proc. of Middleware 2001*, Heidelberg, Germany, Nov. 2001, pp. 329–250.

[Sal1984] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-End Arguments in System Design," *ACM Transactions on Computer Systems*, vol. 2, issue 4, pp. 277-288, Nov. 1984.

[Sar2008] Mikko Särelä, Teemu Rinta-aho, Sasu Tarkoma, "RTFM: Publish/Subscribe Internetworking Architecture," in *Proc. ICT-Mobile Summit*, Stockholm, Sweden, June 2008.

[Sch2000] H. Schulzrinne and E. Wedlund, "Application-layer mobility using SIP," *SIGMOBILE Mobile Computer Communications Revue*, vol. 4, issue 3, pp. 47-57, 2000.

[Soo2008] M. Sooriyabandara, T. Farnham, C. Efthymiou, M. Wellens, J. Riihijärvi, P. Mähönen, A. Gefflaut, J. A. Galache, D. Melpignano, A. van Rooijen, "Unified Link Layer API: An generic and open API to manage wireless media access", 2008.

[Sop2003] A. Soppera, T. Burbridge, B. Briscoe, and M. Rizzo: "GAP: The generic announcement protocol for event messaging," in proc. *London Communication Symposium (LCS'03)*, UCL, London, UK, Sept. 2003.

[Sto2002] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure," in *Proc. 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Pittsburgh, PA, Aug. 2002, pp. 73-86.

[Su2007] J. Su, J. Scott, P. Hui, E. Upton, M. How Lim, C. Diot, J. Crowcroft, A. Goel, E. de Lara, *Haggle: Clean-slate networking for mobile devices*, Technical Report, University of Cambridge Computer Laboratory, Jan. 2007.

[Tar2007] S. Tarkoma and J. Kangasharju, "On the Cost and Safety of Handoffs in Content-based Routing Systems," *Computer Networks*, vol. 51, no. 6, Apr. 2007.

[Tar2008] S. Tarkoma, D. Trossen, M. Särelä, "Black Boxed Rendezvous Based Networking," *The 3rd ACM International Workshop on Mobility in the Evolving Internet Architecture (MobiArch 2008)*, Aug. 2008.

[Tro2007] D. Trossen and D. Pavel, "NORS: An Open Source Platform to Facilitate Participatory Sensing with Mobile Phones", in *Proc. Mobile and Ubiquitous Systems: Networking & Services (MobiQuitous 2007)*, Philadelphia, PA, Aug. 2007.

[Wal2004] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker, "Middleboxes No Longer Considered Harmful," *in proc. 6th USENIX OSDI*, San Francisco, USA, Dec. 2004.

[Yan2007] X. Yang, D. Clark, and A. Berger, "NIRA: A New Inter-Domain Routing Architecture," *IEEE/ACM Transactions on Networking*, vol. 15, no. 4, pp. 775-788, Aug. 2007.

[Yeu1999] R. W. Yeung and Z. Zhang, "Distributed Source Coding for Satellite Communications," *IEEE Transactions on Information Theory*, IT-45, pp. 1111-1120, 1999.

[Zah2009] A. Zahemszky, A. Csaszar, P. Nikander, and C. Esteve, "Exploring the pubsub routing/forwarding space" *In International Workshop on the Network of the Future 2009*, 2009.