Virtual Network Stacks

Ghazi Bouabene, Christophe Jelger, Christian Tschudin
Computer Science Department,
University of Basel Switzerland
ghazi.bouabene@unibas.ch, christophe.jelger@unibas.ch,
christian.tschudin@unibas.ch

ABSTRACT

In this paper, we get inspiration from peer to peer file sharing networks to provide a new way of inter-networking. In our proposal, nodes having access to multiple network types can share their networking resources with other peers residing in networks with different protocols and (potentially) different addressing schemes. Such neighbor nodes will form a peer to peer overlay backbone; the purpose of it being to offer to applications and protocols access to remote network stacks that their running hosts do not implement or have no direct access to. This creates RPC-like access to foreign network stacks well in line with a federation approach that avoids introducing a global overlay for integrating heterogeneous networks.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Distributed networks*

General Terms

Design.

1. INTRODUCTION

Networks have been used to develop all sorts of distributed applications such as distributed file systems, databases, grid computing, etc. Also, many interesting techniques have used networking services in order to offer hosts access to remote functionality available through the network. Mechanisms such as Remote Procedure Calls (RPC), Remote Method Invocation (RMI) and CORBA, offer the hosts implementing them the possibility to extend their functionality. Using such mechanisms, a host can invoke remote services provided by network peers almost as if the services were available locally. Although such mechanisms have been widely used to deploy numerous types of software in a distributed way, there has been little effort in that sense for core networking software.

PRESTO'08, August 22, 2008, Seattle, Washington, USA.

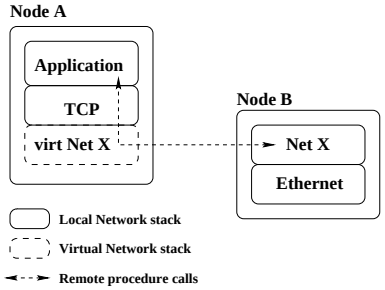

Figure 1: Conceptual view: A virtual (and distributed) network stack

This notion of distributing networking software that blurs the distinction (in layers) between application and network logic might be confusing. Indeed, a network is by definition a distributed system, as it is a collection of remote entities collaborating to provide a common service. However, the software i.e., the networking stack, allowing a host to interact within a certain network, must always be fully implemented within the host. Our intention in this paper is to apply the functional distribution and virtualization methods provided by techniques such as RPCs and CORBA, to networking stacks. Hosts that do not implement a specific networking protocol will be able, via such mechanisms, to "transparently" access distant services provided by remote network peers.

Our proposal basically allows networking hosts to complement their networking functionnality in order to better support application portability and also offers a new way of inter-networking in a heterogeneous network environment.

1.1 Basic operation

Our proposal allows applications and protocols to access networking stacks residing in different hosts. The goal is to extend the panel of network services a host can use. This access should be the most transparent possible to applications and protocols as well as to the remotely accessed network stacks. Our idea is to offer the remote network stack services through a local `virtual networking stack`. Such a virtual stack would present the same interface as the missing networking stack would. Internally, it will handle the necessary operations to relay the requested commands to the remote stack. In our proposal, the entity providing the virtual networking stack will be inter-operating with the remote networking stack with RPC-like mechanisms.

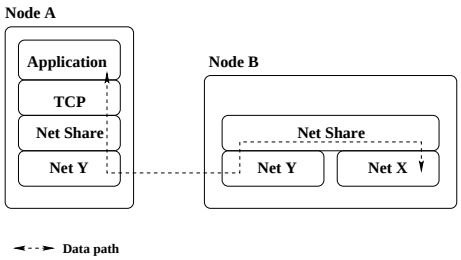

Figure 2: Implementation view : the Net share architecture

Figure 1 illustrates our proposal with a basic example. In this example we suppose that the administrator of node B is willing to share the local network X stack, where X stands for any networking technology (IPv4, IPv6, Bluetooth, etc.), with other hosts. We then assume that an application on node A requires a TCP/X networking stack in order to operate. However, node A does not implement any X networking module and has no direct access to the X network. As shown on the figure, our proposal is to offer a virtual X layer to the TCP module on node A. This virtual layer will use the available networking resources on node A in order to access the X stack on node B in an Remote Procedure Call (RPC) like fashion detailed in the next subsection. Note that this remote procedure execution poses evident security issues since a host might want to restrict the access to its network stacks. Although security issues are not covered in this paper, they could be handled for example by maintaining Access Control Lists allowing only trusted hosts to access the shared networking stacks.

1.2 Proposal details

Figure 2 shows in more details how we implement the remote procedure calls in order to virtualize remote networking layers. In this figure, we suppose that both node A and B support the technology and protocols for network Y (here again, Y stands for any networking technology) and that they can reach each other. The RPC-like access arrow shown in Figure 1 is in fact implemented by two `Network Sharing (NetS)` modules running on nodes A and B. These modules use the services of the Y network in order to communicate with each other. The `NetS` module on node A is the one virtualizing the remote X stack in the context of node A. It offers to the higher layer modules on node A the same interface a local X stack would. This way the application interacts (transparently) with the NetS module as if it was a local X stack. The Network Sharing module on node A forwards the requests received from the application to its peer on node B via the Y network.

The NetS module on node B is responsible for presenting the services of the X stack to the users of the Y network. Indeed without this module, the X stack on node B has no interaction with the Y stack and can not be reached by Y network users. It is this Network Sharing module that receives the remote procedure calls from its peer in node A, through the Y stack, and relays them into local X primitive calls.

1.3 Features

Bridging the gap between distributed systems and network architectures, our proposal of virtual network layers offers three attractive features. First, it has the advantage of favouring applications portability. Indeed, the ability of accessing remote networking stacks allows a host to invoke additional virtual network stacks to complement its local network functionalities. When running an application requiring a networking stack unavailable locally, the host can offer a virtual stack with the needed interface to the application.

Another feature of our proposal is that it permits to distribute the networking functionality (stacks). In the example of Figure 1, the logical `Application/TCP/X` stack is divided in two halves running on two separate nodes. Such a distribution allows to factorize networking stack functionality i.e. identifying the parts common to all host local stacks and centralize them in one point of the network. For example, node A in figure 1 could be a sensor mote accessing a common IPv4 stack provided by a gateway (node B). Such a factorization frees the sensor motes, present in the same network as B, from implementing an IPv4 networking stack. This factorization has the advantage of easing the management and update of deployed networking stacks. Indeed, modifying a shared networking stack will automatically impact all the hosts using the services of the shared stack. For example, updating the IPv4 module in the sensor network gateway example will impact all the motes using its services remotely.

Finally, our virtual stacks proposal offers a new internetworking method. Indeed, when applications *transparently* access remote networking stacks, they are in fact *transparently* accessing remote networks. In fact such mechanisms can be extended to offer access to remote networks residing "many network hops away". As we will show in section 5, these network stack sharing mechanisms create a global inter-network that does not hide all concatenated networks with a global addressing overlay (as is the case for IP).

2. ARCHITECTURAL ABSTRACTIONS

Our proposal is currently being developed in the context of the Autonomic Network Architecture (ANA) project [1]. To support the re-configuration and adaptation of networking components, ANA introduces a flexible framework for composing "network stacks" on demand in a dynamic way. This framework is based on a small set of networking abstractions and a communications API which allow the various components of ANA to interact in a generic manner.

In this section, we introduce the core abstractions and the communications API of ANA: these concepts are a prerequisite for understanding the design of our proposal. The generic nature of ANA is indeed essential for supporting all possible and unforeseen interactions between distributed networking components.

2.1 Basic components of ANA

Functional Blocks.

The *functional blocks* (FBs) are the result of ANA's decomposition of current network stacks: they basically serve as abstractions for any protocol entities generating, consum-

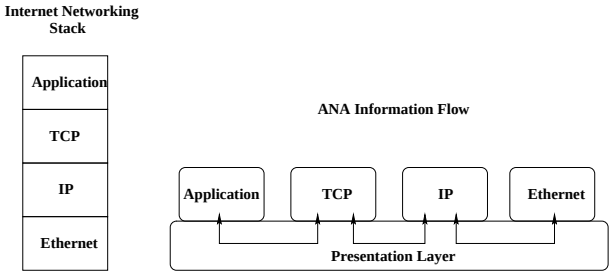

Figure 3: ANA common presentation layer

ing, processing and forwarding information. For example, an encryption function, an IP stack, or a TCP module, can be abstracted as a functional block.

Information Dispatch Points.

The *information dispatch points* (IDPs) play a fundamental role in providing the generic communication hooks between functional blocks. Somehow similar to Unix file descriptors, IDPs are ANA's abstraction of *access means* to networking resources. In practice, IDPs are identified by flat labels that have a meaning only locally to an ANA node. Once a channel to a remote peer or local FB is obtained, an IDP is assigned and *bound* to the communication or the FB. All further communication is then performed via the local IDP. Note that the binding of an IDP is dynamic and can change over time as the "network stack" is re-configured.

Common Presentation Layer.

This layer plays an important role in providing generic data representation between functional blocks. As shown in figure 3, in ANA all functional blocks are using this layer and all communications between the functional blocks are going through it. The presentation layer basically defines a protocol (internal to an ANA node) that allows the functional blocks to exchange commands and arguments. This protocol is similar to RPC's XDR [2] protocol in the sense that it provides a minimal semantic description of the commands and arguments being exchanged by functional blocks.

Although this level of indirection in the communication between network modules can affect the performance of the system, we consiously trade-off performance for the greater flexibility offered by such an indirection (cross-layer optimizations, dynamic adaptation, evolvability, etc.).

Network Compartments.

In ANA, communication channels across ANA nodes are provided by *network compartments*. The concept of network compartment is similar to the notion of *context* as proposed in Plutarch [3] where "a context describes a region of the network that is homogeneous in some regard with respect to addresses, packet formats, transport protocols, naming services, etc". For example in ANA, typical network compartments are: an Ethernet segment, the public IPv4 Internet, a private IPv4 subnet, the DNS, peer-to-peer systems like Skype, and distributed web caching networks like Akamai.

Typically in each network compartment, protocol entities collaborate in order to provide communication services to other compartments and applications. The access to the communication services of a given compartment is provided on each ANA node by the corresponding *builder functional block* which supports the generic communications API described in the next section.

Node compartment.

The ANA architecture has the additional particularity of pushing networking abstractions inside the network hosts. We indeed consider a networking host to be itself a network composed by the functional blocks running on the host. As a result, every ANA node is organised as a *node compartment* which supports the generic communications API described below. In particular, this permits functional blocks to discover each other and interact inside the node compartment in the same manner as with any other network compartment.

2.2 Generic communications API

In ANA, every network compartment builder uses the presentation layer to support a common and generic API for accessing communication services. The generic API is ANA's understanding of the biggest common subset of services offered by today's networks. The API has the following primitives described below with some C-style function prototypes.

- IDP_s `publish(`IDP_c`, CONTEXT, SERVICE)`
- `int unpublish(`IDP_c`,` IDP_s`)`
- IDP_s `resolve(`IDP_c`, CONTEXT, SERVICE,` IDP_r`)`
- `void* lookup(`IDP_c`, CONTEXT, SERVICE,` IDP_r`)`
- `int send(`IDP_s`, DATA)`

Similar to Plutarch [3], the API follows a publish/resolve communication model in which a *service* is *published* within a certain compartment's *context*. A published service can then be *resolved* into a communication channel (identified by an IDP) that can further be used to *send* data to the resolved service. Beside resolution, one can also *lookup* a service to obtain further reachability information but not instantiate a communication channel to the service. For example, looking up a name via the DNS compartment typically returns an IP address (and not a communication channel).

In the prototypes we just introduced, the IDP_c identifies the functional block providing access to the network compartment services. The SERVICE is typically what is being published or looked up, while the CONTEXT defines some scope inside the compartment. The IDP_s is used to refer to a published or resolved service, while the IDP_r specifies the IDP on which the response to some request is expected (remind that communication via the presentation layer is via message passing).

The CONTEXT argument defines the publication or resolution scope "inside" a compartment. For example, in an IP compartment one uses IP addresses as CONTEXT values, while in a DNS compartment the CONTEXT is used to specify record types (e.g., A, MX). ANA also mandates that all network compartments understand two generic CONTEXT values. The generic CONTEXT `"*"` specifies the largest possible scope as interpreted by the compartment, while the `"."` CONTEXT restricts the scope to node-local operations.

The SERVICE argument is a description of the service to be published, and is typically not interpreted by the compartment provider FB. The type of the SERVICE argument

is not fixed: this argument can be a port or protocol number, a string (e.g., URL or filename), an IP address, a hash value, etc.

As previously stated, all compartments in ANA must support this generic communications API: it basically provides the "glue" for all possible and unforeseen interactions within ANA. In particular, the API and the generic data representation permit to deploy advanced network services in a very simple way.

3. REMOTE STACK INTERACTIONS

In this section we will demonstrate how to make use of the architectural abstractions and more specifically the common generic API provided by the ANA architecture to implement the stack virtualization mechanisms described in section 2. Figure 4 maps the example case depicted in figure 2 into the ANA architecture.

Here the services of the Y and X networks are provided by the corresponding Functional Blocks present on node A and B. These functional blocks support the generic API described in the previous section. We will now detail the steps allowing the user application and TCP functional block present on node A to interact with the remote X functional block as if it was available locally on node A.

3.1 Accessing the remote stack

Upon startup, as it is intended to work over the X network, the TCP FB will query its node compartment for a functional block fitting the X description[1]. This can be done by addressing a generic `resolve` command to the node compartment. Since the node A does not run an X functional block, this resolve command would normally fail. At this point, two possible ways for accessing the remote network stack are possible.

The first method is that the TCP module explicitly asks the Network Sharing funtional block on node A to instantiate a virtual X stack. This access method does not change the way the TCP module would interact with the remote stack but still requires a change of behaviour from protocols and applications. A second and more implicit way of providing the remote access is that the NetS functional block on node A intercepts the `resolve` command addressed by the TCP module to the node compartment. It then offers an IDP to reach the virtual remote stack as if the answer came from the node compartment. This way the TCP module will be accessing a remote stack exactly as if it was available on its node (no change of behaviour required).

Independently of the chosen access method, once the NetS FB on node A receives the resolve command for a remote X FB, it transmits it to its NetS peer in node B via the Y network. Using the node compartment services (by addressing a `resolve` command), the NetS functional block on node B can discover the local X functional block. At this point, both NetS funtional blocks use the services of the Y network to build a communication channel leading from IDP `t` in node A towards IDP `p` in node B.

The NetS FB on node A will then return the IDP label `t` to the TCP FB as a response to its resolve request. Note that at this level the TCP functional block will be accessing the remote X FB exactly as if it was available locally.

[1]This description is indicated in the SERVICE argument of the resolve command addressed to the node compartment

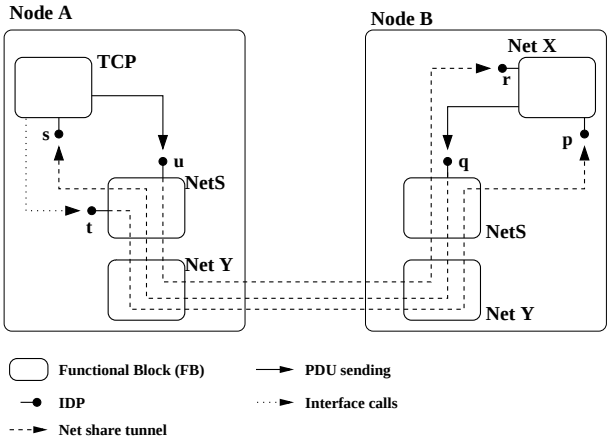

Figure 4: Remote stack access mechanisms

3.2 Remote interactions

Offering services through a remote stack.

Now we assume that an application wants to make a service, reachable locally via IDP `s` attached to the TCP FB, available to X users. For doing so, The TCP FB uses the generic `publish` command and addresses it to the IDP label `t` leading to the virtual X stack. Here again we insist that the TCP functional block is behaving exactly in the same way as if the X FB was running on its host. The network sharing functional block on node A then forwards the publish command to its peer through the Y communication channel. Before relaying the publish command to the X functional block, the NetS FB on node B creates a communication channel leading towards the IDP `s` in node A through the Y network. This channel is accessible in the context of node B via the IDP `q`. The NetS FB on node B then relays the publish command to the X FB by substituting the IDP label `s` with `q`.

At this point, all data received by the X FB and destined towards the published service will be forwarded locally to the IDP `q` and then automatically tunneled through the Y network towards IDP `s` in node A.

Accessing remote services via a remote stack.

Suppose an application asks the TCP functional block on node A to establish a communication channel towards a particular X host. This can be done by addressing a generic `resolve` command to the X stack (i.e. to IDP `t`) with the target X address as a CONTEXT. The network sharing functional block on node A receives the command and transmits it to its peer on node B that in turn relays it to the X FB on node B. The X functional block then instantiates a communication channel towards the requested peer and returns the resulting label `r` to the NetS FB on node B. Now, both NetS functional blocks on node A and B collaborate to instantiate a communication channel leading from the newly created IDP `u` on node A towards IDP `r` on node B. The NetS functional block on node A then returns the IDP label `u` to the TCP FB as a result of the resolve request. At this point, all data sent by the TCP FB on IDP `u` will be tunneled through the Y network and then forwarded to the X FB on its IDP `r`. The X functional block then adds the

received data into an appropriate X packet and sends it to the destination address.

4. NETWORK SHARING OVERLAY

In this section we show how to construct an overlay compartment (network), that will help protocols and applications reach networks that they do not have direct access to or that their nodes do not implement. To relate to the previous section, we will describe here how to build a network based on Network Sharing functional blocks similar to the ones described before. The goal is to offer access to remote network stacks residing in different networks from the host.

4.1 Components of the overlay

The network sharing overlay is composed of two entities:

- Core members: the components forming the core of our overlay are network sharing functional blocks similar to the one on node B in figure 4. Core members are the nodes sharing networking resources and exchanging network capabilities information with other core peers. Given that the network sharing functional block on node B relies only on generic API calls to operate, the same module can be used for sharing all sorts of networks. However, in order to adapt to different (new) networks, our NetS core functional blocks should not depend on any pre-established structure. Therefore we prefer a decentralized and incrementally growing structure for our network sharing overlay. In our proposal, the NetS core members will build the core of the overlay on a peer to peer basis.

- User members : users of the overlay are nodes similar to the NetS block on node A of Figure 4. These members are needed for handling the tunneling interactions with the core members and offering a virtual stack to applications. They do not perform any sharing activity and have therefore the advantage of simplifying the client code that hosts willing to use the overlay have to support.

4.2 Core formation

On startup, the core network sharing functional blocks try to discover the networking capabilities of their local hosts. This can be done with a generic `lookup(n, ".", "compartment", r)`, where n is the IDP label of the node compartment and r the IDP label to handle the response. Doing so, a core NetS functional block obtains a list and description of all the host-local Functional Blocks providing access to networks.

The next step in the bootstrap process is to discover and interact with other NetS core peers reachable via the available networks. Therefore, for every FB `x` providing networking functionality, the core NetS functional block will execute the two following commands :

- `resolve(x, "*", "net share", r)` : where x is to be subsequently replaced by the IDP labels of the previously discovered FBs. As a result, the compartment provider FB will return an IDP to reach all possible NetS peers in its interpretation of the `"*"` CONTEXT.

- `publish(x, "*", "net share", s)` : where x is again to be replaced by the IDP labels to reach the network providing functional blocks. This will make the core NetS FB visible in the context of the network i.e. discoverable by other NetS core peers and overlay user members.

Note that the described bootstrap procedure relies only on generic API primitives, which allows the NetS core FB to interact within all networks providing the API.

4.3 Addressing and routing

Once it can interact with its peers, a NetS core functional block will request to join the core of the overlay. Typically, this means obtaining an identifier (name or address) and routing information. Note that this identifier is transparent to the applications taking advantage of the network sharing overlay. Indeed, user applications keep their local network identifiers and need only to provide a description of their target network. At the status of our proposal, we do not specify any naming (addressing) nor routing conventions. Our only concern (requirement) in this matter is that the naming and routing procedures can be fully decentralized and automated (i.e. no human intervention). In fact, many existing techniques with e.g. flat routing and DHTs [4, 5] can be applied here.

4.4 Information sharing

Once a NetS core functional block has joined the overlay, it can start exchanging network capability information with its peers. It will use a global information sharing mechanism to announce the description of the networks available on its host. Note that this networking capability information can be tuned by policies or a human administrator to include only a subset of the functionnality that a host is willing to share.

Here again, at the current state of the proposal, we do not specify a particular information sharing mechanism. However, considering our requirement for decentralization, peer to peer information sharing mechanisms [6, 7] are good candidates for this task.

5. RELATED WORK AND DISCUSSIONS

To summarize, we propose in this paper a set of mechanisms allowing a host to access remote networking stacks and virtualize their functionality to local applications. Beside enhancing applications portability and factorizing functionality, these techniques also support a new inter-networking model. This model consists of a peer-to-peer overlay destined for sharing network stack functionality. This overlay allows hosts residing in different and distant networks to communicate without imposing global networking semantics. Indeed, our inter-networking model does not introduce a new global namespace federating all interconnected networks as it is the case for IP. Also, our deployed overlay does not “hide” the underlying networks by integrating them (as for IP), but rather offers a backbone network to make them globally visible. This ability of accessing a remote network stack has the advantage of confining service access/publication to a specific part of the global network. For example, offering a service to a peer in a private network becomes possible by accessing a remote networking stack residing in the private network and publishing (creating the access point) the service in the context of the private network only.

Our proposed work can be confused with gatewaying and tunneling techniques. Indeed, a `Net Share` module linking two networking stacks such as the one on node B in Figure 2 resembles service adaptation (interface adaptation) gatewaying techniques. A major difference of our proposal is that our network sharing modules do not translate the services of a network N into those of N' as it is the case for service adaptation gatewaying [8, 9]. Instead, the NetS modules simply use the services of the network N in order to transport remote procedure calls towards a stack implementing the network functionalities of N'.

Our proposal can also be confused with protocol tunneling techniques. Indeed, the example of Figure 2 can be interpreted as a mechanism for tunneling X packets through a network Y. However, a major difference of our proposal from protocol tunneling techniques is that we do not encapsulate the packets of a network N' inside those of a network N as it is the case for protocol tunneling techniques [10]. We rather relay the user data (or PDU) from a packet N towards a packet N' (and vice versa) at the border of the networks. Indeed, in our example of Figure 2, there are no X packets circulating in the traversed Y network. Only PDUs are relayed between X and Y packets at the level of the NetS module in node B (border between the two networks).

Solutions similar to our proposal actually exist. In [11], a remote access to X.25 network services is offered for nodes only having an Internet connection. This solution uses RPC to tunnel X.25 API calls and data over an IP network. In [12], communication capabilities of several hosts in a lab are merged using inter-process communication mechanisms over a Cambridge Ring. While these techniques are similar to our proposal, they are very specific to particular networking technologies. In contrast, our proposal is fully generic: first, our network sharing overlay can create peerings on top of any underlying network technology and second, the overlay can be used to share any kind of network stack as long as it supports our generic communications API.

6. FUTURE WORK

A central assumption of our work is that, to be remotely accessible, a network stack must support our proposed communications API and presentation layer. However while looking at the very few primitives of the API, one may suspect that the API is too restrictive and may not permit to "express" advanced communication services. Actually like others [3, 13], we believe that the richness of an API does not depend on the number of primitives it offers. In our proposal, the expressiveness of the API is actually tied to the SERVICE and CONTEXT arguments which permit to specify *what* is being published or resolved and *where/how* this should be done.

In practice, we are currently assessing how existing "network types" can be mapped into our proposed API. Our current approach is to use human readable strings to encode the SERVICE and CONTEXT arguments. This for example permits to encode simple contexts such as `"10.1.2.3"`, `"224.0.0.9"`, `"2001::1"`, or `"MX"`, as well as simple services like `"tcp:80"`, `"www.example.com"`, or `"some_song.mp3"`. Our current work is actually to establish a clear mapping between the most popular existing protocols and our communications API, and to understand if each derived semantic is rich enough to cover all the features of the corresponding protocol. For example, an IPv4 source route could be expressed with a CONTEXT containing a list of IP addresses (e.g., `"10.1.2.3|10.1.3.4|10.1.4.5"`), but one might also want to extend this semantic to enforce only a partial source route (e.g., to choose an outgoing path in a multi-homed network).

Finally, while continuing the implementation effort on the ANA architecture and the network sharing mechanisms, focus will be put on identifying information sharing, routing and naming schemes for our overlay that offer the best tradeoff between global communications efficiency and flexibility. The impact of this tradeoff will be measured via performance evaluation comparing our architecture to already deployed networking solutions as well as to other novel network architectures.

7. ACKNOWLEDGEMENT

Still running until the end of 2009, this work is carried out in the context of the ANA project [1] (FP6-IST-27489) funded by the European Commission under the proactive initiative on "Situated and Autonomic Communications" (SAC).

8. REFERENCES

[1] Autonomic Network Architecture - EU Project (2006-2009). `http://www.ana-project.org`.

[2] R. Srinivasan. RFC-1832 - XDR: External Data Representation Standard, 1995.

[3] J. Crowcroft, S. Hand, R. Mortier, T. Roscoe, and A. Warfield. Plutarch: an argument for network pluralism. In *FDNA '03: Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, pages 258–266, New York, NY, USA, 2003. ACM.

[4] Matthew Caesar, Tyson Condie, Jayanthkumar Kannan, Karthik Lakshminarayanan, and Ion Stoica. ROFL: routing on flat labels. *SIGCOMM Comput. Commun. Rev.*, 36(4):363–374, 2006.

[5] Matthew Caesar, Miguel Castro, Edmund B. Nightingale, Greg O'Shea, and Antony Rowstron. Virtual ring routing: network routing inspired by DHTs. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 351–362, New York, NY, USA, 2006. ACM.

[6] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, 2001.

[7] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. pages 329–350, 2001.

[8] M. Gien and H. Zimmermann. Design principles for network interconnection. In *SIGCOMM '79: Proceedings of the sixth symposium on Data communications*, pages 109–119, New York, NY, USA, 1979. ACM.

[9] G. V. Bochmann and P. Mondain-Monval. Design principles for communication gateways. *Selected Areas in Communications, IEEE Journal on*, 8(1):12–21, Jan 1990.

[10] R. Gilligan and E. Nordmark. RFC-2893 - Transition Mechanisms for IPv6 Hosts and Routers, 2000.

[11] Advanced Relay and LayGO toolkit. `http://www.advancedrelay.com/`.

[12] R. Braden, R. Cole, P. Higginson, and P. Lloyd. A distributed approach to the interconnection of heterogeneous computer networks. *SIGCOMM Comput. Commun. Rev.*, 13(2):254–259, 1983.

[13] M. Demmer, K. Fall, T. Koponen, and S. Shenker. Towards a Modern Communications API. In *Proceedings of Fifth ACM Workshop on Hot Topics in Networks (HotNets-VI)*, November 2007. Atlanta, USA.