The Autonomic Network Architecture (ANA)

Ghazi Bouabene, Christophe Jelger, Christian Tschudin, Stefan Schmid, Ariane Keller, and Martin May

Abstract—**The objective of autonomic networking is to enable the autonomous formation and parametrization of nodes and networks by letting protocols sense and adapt to the networking environment at run time. Besides its dynamic aspects, a core requirement of autonomic networking is to define a structured framework and execution environment that enables algorithms to operate in a continously changing environment. This paper presents the major design principles of the *Autonomic Network Architecture* (ANA) and reports on a first implementation. The guiding principle of ANA is to strive for flexibility and genericity at all levels of the architecture. In our approach we explicitly avoid to impose a "one-size-fits-all" architecture (where communication protocols and paradigms are fixed by the architecture). To this end, ANA introduces generic abstractions, for example "information dispatch points" instead of addressable endpoints, as well as communication primitives that support network heterogeneity, adaptability, and evolution. These core abstractions allow for the coexistance of multiple and diverse networking styles and protocols. With the public release of the ANA prototype, we aim at federating autonomics related networking projects, enabling different actors to share, compare, and build upon each other's work. The ANA runtime can host clean slate network designs as well as legacy Internet technology and serves as a platform for demonstrating autonomic communication principles.**

Index Terms—**Network architecture, autonomic networking, future internet, clean slate network design.**

I. Introduction

END USERS might have the impression that the Internet is working just fine, which usually is confirmed through our daily experience. When looking under the hood, however, one realizes that considerable manual configuration effort is necessary to keep this system running. In addition, a considerable number of operational and protocol *patches* (e.g. CIDR, NAT) had to be introduced in order to sustain its growth and increasing complexity. However despite these changes, the Internet core suite of protocols has almost not evolved since the 70ies, with new features such as IP multicast, IP mobility, and IP version 6 being slow or having failed to be widely deployed.

A growing fraction of researchers and network operators are actually seriously concerned about the ability of the Internet to sustain more disruptive evolution steps. For many researchers including some of the early Internet designers [1], [2], the salient design principles (i.e. end-to-end principle and transparency, and global addressing) that contributed to the success of the Internet are also preventing any radical update of its core functionalities. As a result, the networking community has witnessed a renewed interest in the design of *clean-slate* network architectures (see e.g., [3], [4], [5], [6]) that could better support disruptive evolution and emerging network paradigms such as content-based, ad hoc, sensor, mobile, and delay-tolerant networking. A key motivation is to foster innovation by freeing researchers from providing backwards compatibility with the current Internet.

One direction of clean-slate research looking at adaptive and automated network operation is *autonomic networking*. The term "autonomic" (first coined in 2001 by IBM [7]) refers to the ability of a system to perform its operation via (so-called) self-* properties i.e., without requiring active human intervention. Current network protocols cover some part of the self-* properties, with for example routing and address autoconfiguration leading (to some extent) to the *self-organisation* and *self-configuration* of networks.

However, today these protocols are confined to narrow tasks (i.e. each protocol is designed in isolation to solve a specific problem) and they are not interlinked because of the exponential complexity of dealing with all possible interactions. If we want to develop more reliable and predictable communication systems, this disjoined development cycle must end and be replaced by a global and clearly structured development and execution framework. This would permit to sustain network evolution and at the same time reduce the overhead of human management, and it would mitigate the interaction complexity by allowing protocols and algorithms to operate in an autonomic manner.

To support these objectives, the *Autonomic Network Architecture* (ANA) project [8] is exploring novel ways of organising and using networks beyond legacy Internet technology. We are designing and developping a network architecture that can demonstrate the feasibility and properties of autonomic networking. The main guiding principle behind the architectural work in ANA is to strive for a maximum degree of flexibility in order to support *functional scaling* by design at all levels of the architecture. Functional scaling means that a network is able to extend both horizontally (*adding* more functionality) as well as vertically (different ways of *integrating* abundant functionality).

When designing ANA, we have favoured several tenets which we think are essential for an evolvable network architecture:

- A framework like ANA has to provide a set of generic abstractions which are used to model the network operation and which can also be turned into running code.

Manuscript received 22 December 2008; revised 31 May 2009.

Ghazi Bouabene, Christophe Jelger, and Christian Tschudin are with the Computer Science Department, University of Basel, Switzerland (e-mails: {firstname.name}@unibas.ch).

Stefan Schmid is with NEC Research Europe, Network Research Division, Heidelberg, Germany (e-mail: Stefan.Schmid@nw.neclab.eu).

Ariane Keller and Martin May are with the Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland (e-mails: {firstname.name}@tik.ee.ethz.ch).

Digital Object Identifier 10.1109/JSAC.2010.100102.

As described further in this paper, these are the notion of *compartment*, *information channel*, *functional block* and *information dispatch point* (Section II).

- Instead of defining rigid specifications like protocols and packet header fields, the framework has to identify and define generic communication methods to interact with the abstractions used to model the network (Section III).
- Finally, a common "communication core", which all components of a network instance can rely on to be present, has to be defined and implemented (Section IV).

Overall, ANA becomes a meta-architecture in the sense that it is a framework to host, interconnect, and federate multiple heterogeneous network instances. But unlike the Internet, which relies on a unique and globally shared addressing scheme, ANA is not another "one-size-fits-all" network waist. For example, ANA defines a set of primitives and related methods which are *address agnostic*, and hence do not impose a common address type and format.

In this paper, we describe the most prominent design features of ANA. Section II introduces the fundamental abstractions that protocols and algorithms manipulate with the communication primitives described in Section III. The low-level machinery of ANA is detailed in Section IV, and the overall operation of ANA is illustrated with examples in Section V. Section VI discusses the deployment and security of ANA, and Section VII introduces our prototype implementation. We finally discuss some related work in Section VIII conclude the paper in Section IX.

II. Architectural Abstractions

A network architecture is a set of design principles describing the scope, the objectives, and the abstract (i.e. high-level) operation of a communication system. It defines the atomic functions and entities that compose the network and specifies the interactions which occur between these various building blocks. In ANA we have defined the networking abstractions that we consider key in order to let autonomic functionality "play" with the various ways networks can be implemented and be operated. Our view is that ANA has to provide a minimal yet universal set of networking concepts which permits different networking styles to be expressed and allows users to access communication services in a generic manner.

The core abstractions of ANA are illustrated by Figure 1, which shows the modeling of a simple unicast communication. In the next subsections, we introduce these abstractions and describe their basic usage and purpose.

A. Network compartment and information channel (IC)

At the coarsest level, the ANA world is organized in *network compartments*. The concept of network compartment is similar to the notion of *context* as proposed in Plutarch [9] 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". In ANA, each compartment is free to *internally* use whatever addressing, naming, routing, networking mechanisms, protocols, packet formats, etc. However in order to offer a standard access to the communication services it provides, each

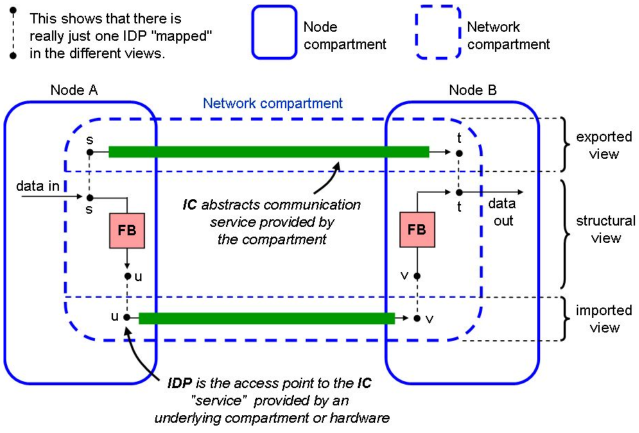
This shows that there is really just one IDP "mapped" in the different views.
Node compartment
Network compartment
Node A
Node B
Network compartment
s
t
data in
s
t
data out
FB
FB
IC abstracts communication service provided by the compartment
u
v
u
v
exported view
structural view
imported view
IDP is the access point to the IC "service" provided by an underlying compartment or hardware

Fig. 1. The fundamental abtractions of ANA.

compartment must support a generic "compartment API" that *wraps* its internal operation with generic constructs. In other words, each compartment operates as a black-box supporting a generic communication API (which is described in details in Section III-B). In practice, an Ethernet segment, the public IPv4 Internet, a private IPv4 subnet, peer-to-peer systems like Skype, and distributed web caching networks like Akamai, can each be modeled in ANA as a network compartment.

Typically in each network compartment, a distributed set of protocol entities collaborate in order to provide communication services to other compartments and applications. Whatever the nature of the communication service (i.e. unicast, multicast, datagram, reliable stream, etc), it is abstracted in ANA by an *information channel* (IC). This notion is illustrated by Figure 1 where the network compartment exports a communication service abstracted by the information channel (IC) $\texttt{s}\rightarrow\texttt{t}$. How this channel is instantiated is typically not exposed to entities not being part of the compartment which merely use the provided communication service and often do not care how this is implemented. Of course a compartment is an abstract construct which is really a collection of distributed systems. When interacting with a compartment with the generic API, an entity really interacts with a (node-local) software component implementing the operation of the compartment: the functional block (FB).

B. Functional Block (FB)

In ANA, any protocol entity generating, consuming, processing and forwarding information is abstracted as a *functional block* (FB). For example, an IP stack, a TCP module, but also an encryption function or a network monitoring module can be abstracted as a functional block. In contrast to OSI's protocol entities [10], functional blocks are not restricted to being abstractions of network protocols and the provided functionality can range from a full monolithic network stack down to e.g., a small entity computing checksums (like *elements* in Click [11]). In addition, a functional block (e.g. a passive monitoring module) does not necessarily provide access to some network compartment.

On Figure 1, the two functional blocks providing the information channel $\texttt{s}\rightarrow\texttt{t}$ are shown in the *structural view* of the compartment which shows how the compartment algorithms and protocols are implemented. Hence while the *export* view shows the information channel provided by the network compartment, the structural view shows the functional blocks implementing this communication service. Beside ICs and FBs, a fundamental feature of ANA is that an IC or a FB is always accessed via an indirection system that forms the core of ANA: the *information dispatch point (IDP)*.

C. Information Dispatch Point (IDP)

IDPs are inspired by network pointers [12] and by network indirection mechanisms [13], and they are somehow similar to *file descriptors* in Unix systems. Basically inside an ANA node, a functional block (FB) is always accessed via one or multiple IDPs attached to it: in other words, all interactions are carried out via an indirection level *built-in* in the network architecture. Like Unix systems enforce access to all system components (e.g. devices, sockets, files) via file descriptors, ANA mandates that all functional blocks are accessed via IDPs.

However unlike file descriptors and sockets, the binding of an IDP (i.e. the FB to which it is attached) is dynamic and can change over time as the "network stack" is reconfigured. The bindings between IDPs and FBs are stored in the core forwarding table of the ANA Node where each IDP is identified by a node-local label[1]. As detailed in Section IV, each IDP maintains some dynamic information such as the FB to which it is attached, a lifetime counter, some status information, etc.

The advantage of IDPs is two-fold: first, they act as generic communication *pivots* between the various functional blocks running inside an ANA node and, second, they provide the required flexibility allowing for the re-organisation of communication paths. For packet forwarding, IDPs permit to implement forwarding tables which are fully decoupled from addresses and names: i.e., in ANA the next hop "entity" (local or remote) is always identified by an IDP. This permits to easily add and use new networking technology and protocols as long as they *export* their communication services as IDPs.

D. Node compartment

The ANA architecture has the additional particularity of pushing networking abstractions inside the network hosts. We indeed consider a networking node 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 operates like any other (network) compartment except that it does not provide information channels. This permits functional blocks to discover each other and interact inside the node compartment in the same manner as with any other network compartment. Node compartments are illustrated on Figure 1 by solid-line rectangles with rounded corners.

E. IDPs, FBs, and ICs by example

As stated earlier, Figure 1 illustrates how the main abstractions of ANA are used to model two nodes communicating via a network compartment. On this figure, two functional blocks (shown as red squares) in two ANA nodes provide an information channel `s`$\rightarrow$`t` (shown as a green thick line) from node A to node B. This IC is abstracted by the IDP `'s'` (shown as a thick dot) in node A and leads to the IDP `'t'` in node B. The functional blocks involved are shown in the *structural view* of the compartment which shows how the compartment algorithms and protocols are implemented. The *import* and *export* views show the information channels that are respectively being used and provided by the network compartment shown. Note that the IC `u`$\rightarrow$`v` (used by the two FBs to communicate) is typically exported by another network compartment (not shown here). For example, the IC `u`$\rightarrow$`v` might be provided by an Ethernet compartment which is then used by an IP compartment to provide the IC `s`$\rightarrow$`t`.

All together, the core abstractions of ANA permit to model communication systems with a simple but structured framework. In particular and with the help of the communication API described in the next section, this permits to *wrap* all network compartments with a generic interface such that it becomes possible to access communication services in a "compartment-agnostic" way. This is critical in order to achieve autonomous behavior and adaptability without having to hard-code (and hence freeze) the interactions between the various (protocol) entities of ANA.

III. Communications API

A. Design rationale

As described in Section II, network compartments in ANA are free to *internally* use whatever addressing, naming, routing mechanisms, protocols, packet formats, etc. The objective is that ANA fosters variability and evolution by not imposing any restriction on the internal network operation of compartments. However, to foster and guarantee all possible and unforeseen interactions within ANA, all compartments must support a generic "compartment API" which provides the "glue" that permits to build complex network stacks and packet processing paths.

With respect to application[2] development, a key requirement is that the API supports generic function prototypes and arguments such that applications can use the same programming primitives when communicating to *any* compartment.

In other words, a long-term objective of ANA is that developers can write "compartment-agnostic" applications that do not need to be modified to support new network compartments. For example, with ANA one could write a web browser that takes any *opaque* string as input, finds the right network compartment for handling this address or name[3], requires from this compartment an information channel to the destination and eventually sends the data to the destination FB without, at any time, having to understand the syntax of the input string. This is different from the current situation

[1]The current implementation uses 32-bit unsigned integers for IDP values.

[2]The term "application" refers to some software using ANA and the API: this for example includes network stacks, routing protocols, network monitoring frameworks, as well as "end-user oriented" applications such as video-streaming servers and peer-to-peer systems.

[3]ANA uses regular expressions to match names and compartments and find out that e.g., "1.2.3.4" is handled by the ipv4 compartment. See Section V-A for details.

with sockets where a programmer must use and properly set the appropriate `sockaddr_` structure depending on the "network compartment" being accessed. This hard-coding assumes prior knowledge of "who uses what" and seriously hinders innovation (e.g. all applications had to be re-written to support IP version 6).

B. Main communication primitives

The ANA compartment API currently offers six fundamental primitives detailed below with some simplified C-style function prototypes.

- $\texttt{IDP}_s$ `publish(`$\texttt{IDP}_c$`, CONTEXT, SERVICE)`
- `int unpublish(`$\texttt{IDP}_c$`, `$\texttt{IDP}_s$`)`
- $\texttt{IDP}_r$ `resolve(`$\texttt{IDP}_c$`, CONTEXT, SERVICE)`
- `int release(`$\texttt{IDP}_c$`, `$\texttt{IDP}_r$`)`
- `void* lookup(`$\texttt{IDP}_c$`, CONTEXT, SERVICE)`
- `int send(`$\texttt{IDP}_r$`, DATA)`

Similar to Plutarch [9], 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 an information 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 an IDP/IC to the service. For example, looking up a name via the DNS compartment typically returns an IP address (and not a communication channel). When a resolved information channel is no longer needed, one can *release* the corresponding IDP in order to free some resources.

In the function prototypes, the IDP_c identifies the functional block providing access to the network compartment services. The IDP_s returned (and created) by the `publish` primitive identifies a published service that could later be resolved. Finally, the IDP_r returned (and created) by the `resolve` primitive identifies a communication channel which can be used to `send` data to some remote service.

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.

C. The CONTEXT and SERVICE arguments

The intuitive "design philosophy" behind the ANA API is that, for any kind of network applications and protocols, a user typically wants to find or reach something (the SERVICE) with respect to some scope (the CONTEXT) inside a network compartment. In other words, the SERVICE is what the user is looking for (or wants to communicate with, download, etc), and the CONTEXT defines where and how (in which scope) the network operation (e.g. resolve) must be done. Based on our current experience with the API, we believe that it is possible to fit most networking applications and protocols into the CONTEXT/SERVICE communication model.

For example, in an IP compartment one typically uses IP addresses as CONTEXT values with SERVICEs such as `"tcp:80"` or `"udp:53"`, while in a DNS compartment the CONTEXT could be used to specify record types (e.g., `"A"`, `"MX"`) with SERVICEs being FQDNs such as `"www.example.com"`. In a peer-to-peer file sharing system, one could use the CONTEXT to specify a search scope with boolean operators (e.g. `"title AND (rock OR blues)"`) while the SERVICE could specify keywords (e.g. `"satisfaction"`). Note that the SERVICE argument is a description of the service to be published, and is typically not interpreted by the compartment 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.

Finally, 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. For example, the `"*"` CONTEXT would map into the CONTEXTs `"FF:FF:FF:FF:FF:FF"` and `"255.255.255.255"` for respectively the Ethernet and IP compartments.

In the current implementation of ANA, the CONTEXT and SERVICE arguments of the API are simple `<pointer,length>` tuple so there is maximum flexibility to specify them. However, we currently favor printable (human readable) strings as a *public/external* format for both the CONTEXT and SERVICE arguments. The key motivation is that ASCII is a universal format that can be read and given by human users and can easily be manipulated with lexical tools and algebras (e.g. regular expression matching). Moreover, using strings as public formats avoids the need to know anything about the internal encoding used by compartments (e.g. no need to know that an IP address must be encoded in network-order).

D. Example of communication setup

To illustrate the use of the API, we present a simple communication example and describe how this is implemented with the basic primitives of the API.

Since each node is organized as a compartment, the communications *inside* a node are also setup via the compartment API. In particular, this allows each functional block to discover the services and network compartments available inside the ANA node in which it is running. For example, a functional block implementing the Ethernet protocol can publish itself inside the node compartment with the following primitive:

```
e ← publish(n, ".", "ETHERNET")
```

The IDP 'n' bound to the node compartment is provided to each functional block upon startup. In this example, the CONTEXT is set to `"."` which restricts all operations to the local node. Upon success, the publish primitive creates and returns the IDP 'e' (randomly generated) which is now bound to the Ethernet functional block. Now, a second functional

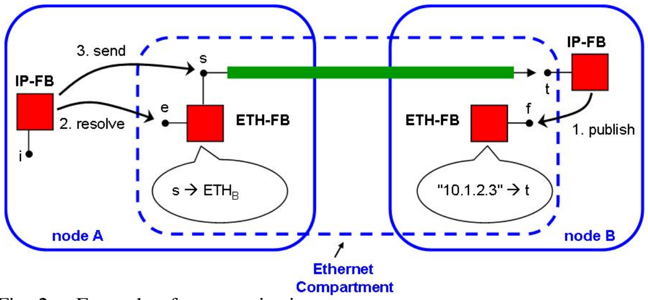

Fig. 2. Example of communication setup.

block which wants to use the Ethernet FB will be able to resolve its IDP with the following request (returning 'e'):

```
e ← resolve(n, ".", "ETHERNET")
```

This functional block (e.g., an IP stack), can now use the IDP 'e' to obtain an information channel via the Ethernet FB as shown later in this example. In the right part of Fig. 2, a functional block implementing an IP stack (IP-FB) publishes its IP address inside the Ethernet compartment via the Ethernet functional block (ETH-FB). This means that the IP-FB becomes reachable via the Ethernet compartment with the service name "10.1.2.3".

This publication is performed by the IP-FB by calling the primitive

```
t ← publish(f, "*", "10.1.2.3")
```

which, upon success, returns the IDP 't'. Note the use of the generic CONTEXT "*" which specifies the largest possible scope as interpreted by the compartment: for Ethernet, this typically means all attached hosts on the segment and means that the publisher wants to become visible in the entire Ethernet compartment.

The left part of Fig. 2 illustrates the resolve and send primitives. In this follow-up example, an "IP-stack" functional block inside Node A asks the Ethernet compartment (via IDP 'e') to resolve the service "10.1.2.3" in the "*" CONTEXT (i.e., the entire Ethernet segment). This is performed by calling the primitive

```
s ← resolve(e, "*", "10.1.2.3")
```

which, upon success, returns the IDP 's' which is used to send data to the resolved service. Inside Node A, the ETH-FB maintains some state that permits to transfer any data sent to the IDP 's' to the IDP 't' in Node B. For Ethernet, this information includes the MAC address of the destination node plus a dynamically allocated token value used for demultiplexing packets to IDP 't' at the destination node.

How the real resolution is performed is left to the compartment. In the case of the Ethernet compartment, a resolution request triggers the sending of a broadcast message (sent to FF:FF:FF:FF:FF:FF) containing the service name (here "10.1.2.3") being looked up. Any Ethernet FB that has published the service in the "*" CONTEXT replies to the request with a message containing the information needed to reach the service via the corresponding compartment. For the Ethernet compartment this request-reply phase is actually similar to today's operation of ARP (Address Resolution Protocol) but with dynamically assigned "ethertype" values for the Ethernet header. It is important to note that other compartments may implement a completely different resolution mechanism, for example involving a (local or distributed) database lookup or no action at all except creating an IDP (as is the case for a datagram service).

After having obtained the IDP 's', the IP-FB in node A can send data to the destination service with send(s,DATA), and it can later release the information channel by calling release(e,s) which notifies the ETH-FB that it can free the corresponding resources.

IV. Core Machinery

As already stated in previous sections, we believe that a network architecture can only be labeled autonomic if it supports the dynamic on-the-fly evolution and adaptation of networking elements. As a starting point towards autonomicity, IDPs provide a flexible and generic machinery upon which adaptable autonomic services can be designed. In this section, we develop some core aspects of the low-level machinery of ANA, including some detailed description of the operation of IDPs and an introduction to the built-in event notification system of ANA.

A. Information dispatch points (IDPs)

Down to implementation details, an IDP is identified by a numerical value (32-bit integer) with a node-local scope: access to an IDP is *always* via its numerical value. The bindings between IDPs and FBs are stored inside each ANA node in the *information dispatch table* (IDT) as <value,FB> pairs. The IDP object itself is a structure that contains various information such as its owner FB (which created the IDP), the callee FB (bound to the IDP), its status (e.g. ready to receive data), and its visibility i.e., *public* (all FBs can use this IDP) or *private* (only a particular FB can use this IDP).

A key feature of IDPs is the ability to bind and re-bind them dynamically to FBs in an on-demand manner. This is critical to permit the on-the-fly redirection of data flows when nodes modify their internal packet processing paths to cope with a changing networking environment. For example, it is possible to redirect the flow of packets to a reliable transmission FB dedicated for wireless links when a node switches its communications onto a radio network interface.

A critical condition is that any FB using some IDP must be able to continue using this IDP while a re-binding operation is carried out. That is, IDP re-bindings should not disrupt data flows. To fulfill this condition, ANA introduces *public* and *private* IDPs. In addition to preventing flow disruptions, there are two other reasons for doing so. First, functional blocks internally bind each IDP value to a particular operation to be carried out and this means that IDP re-binding must preserve IDP values. Second, while performing an IDP re-binding one may not always wish to redirect all the packets sent to a particular IDP. For example, one may wish to redirect only

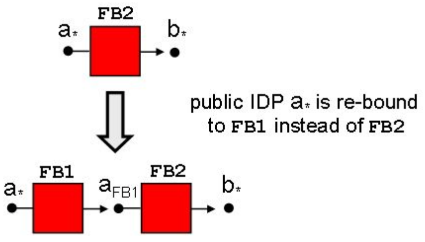

Fig. 3. Re-binding with public and private IDPs.

certain network traffic to an encryption FB while other packet flows are left unencrypted.

Figure 3 illustrates how a non-disruptive re-binding is actually performed. Public IDPs are identified by the subscript character * (such as e.g. a_*): such IDPs can receive data from any functional block. In contrast, private IDPs can only receive data from a particular FB indicated as a subscript suffix (such as e.g., a_{FB1}). In this example, the functional block FB2 continues to receive data via the IDP value 'a' after the re-binding. However, all data sent to the public IDP a_* is now first received by FB1 and then forwarded to FB2. At that point the IDT has two entries $<a_*$,FB1$>$ and $<a_{FB1}$,FB2$>$ and packets are dispatched to the appropriate destination according to the sender FB.

The use of public and private IDPs for performing selective redirection is illustrated by Figure 4 where only the packets sent by the funtional block FB1 are redirected to FB4 while the data sent by FB3 continues to be dispatched to FB2. Note that the header of the packets sent by FB1 still contain the same IDP value 'c' (there is no way to identify from its numerical value whether an IDP is public or private). However at that point, the information dispatch table of the node maintains two entries for IDP 'c' in its IDT $<c_*$,FB2$>$ and $<c_{FB1}$,FB4$>$ and packets are dispatched to the appropriate destination according to the sender FB.

A critical issue when performing IDP re-binding is to be able to migrate the state that a FB maintains for a given IDP. This is for example the case if an IDP bound to a FB providing a reliable TCP connection is re-bound to another FB: in that situation the state of the connection must also be migrated. To support this, ANA offers a mechanism to export/import import state to/from an IDP: this *IDPinfo* system is detailed in the Section V-C.

B. IDPinfo and information channels

Another fundamental role of IDPs is that they act as generic *information hooks* for information channels. That is, when an IDP is bound to an information channel (IC) (like IDP 's' in Fig. 2), it stores additional information to feature the characteristics of the IC. In particular, the IDP exposes the CONTEXT and SERVICE parameters of the information channel end-point so that all the users of the IDP/IC can easily access this information. On the example in Figure 2, one can query the IDP 's' for information and obtain the values of the remote CONTEXT (i.e. the Ethernet address of node B) and SERVICE (i.e. "10.1.2.3"). In addition, the IDP also stores additional information such as the local CONTEXT (i.e.

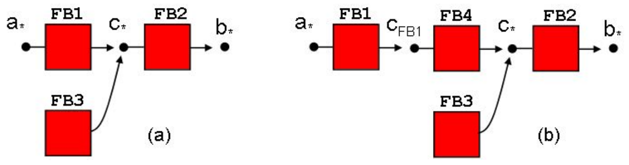

Fig. 4. Restricted IDP re-binding.

the Ethernet address of node A), the nature of the IC (i.e. unicast), the MTU (e.g. 1500 bytes), and the IDP value of the compartment functional block providing the IC (i.e. IDP 'e'). The information regarding IDP 's' can be retrieved using the following function:

```
void* getIDPinfo(s);
```

that returns a data structure containing the IDP information. Note that this function does not appear with the API primitives described in Section III-B as it is supported and implemented by the node architecture and not by the network compartment elements (as it is the case for the functions in Section III-B).

With the IDP information feature, there is no need for an FB to store the parameters that were used to perform a successful resolve request: everything is provided by the IDP/IC. Actually, this fully decouples the way one *accesses* an information channel (i.e. the IDP value) from the address(es)/name(s) of the end-point(s). This prevents the flawed situation where the user of a communication service locally binds to the service with the service name, and which prevents further “updates” of the service. This is what happens today with TCP using IP addresses as end-point identifiers i.e., a flaw which breaks IP mobility or prevents the transformation of a communication from unicast to multicast. With the end-point information being solely stored at the IC/IDP level, such defects are fully avoided. For example, IP mobility can be fully supported: in that case, the IC/IDP information can be updated (by the functional block providing the IC) each time the end-point moves to a new IP subnet. This is achieved in a transparent manner for all the FBs using the IC/IDP.

C. Event notification system

To further permit autonomic entities to react to networking changes, ANA also provides a built-in event notification system. The scope of this system is “node-local” and it allows functional blocks to subscribe to certain events in order to later be informed when such events occur. Depending on the events, either the core ANA machinery itself or a particular functional block is responsible for notifying the event.

The current implementation supports events that notify when an IDP is deleted, redirected, unpublished, busy (i.e. no longer accepting data), ready to receive data, and when its information (e.g. remote CONTEXT changed) is updated. The system also supports events that notify when a functional block is added or removed to/from the ANA node.

All together, these events allow functional blocks to automatically react to networking changes and dynamically adapt to the new conditions. This is essential to promote autonomic algorithms and reduce human interventions.

V. Concrete examples

In this section, we illustrate with simple examples some of the key flexibility features of ANA. These examples demonstrate the interaction and operation of the abstractions within a single ANA node as well as among multiple nodes.

A. Address-agnostic applications

A key feature of ANA is that it supports the development of *address-agnostic* applications i.e., applications which do not have any a priori knowledge of the types of addresses and names they use to communicate with remote peers. First, the compartment API uses generic arguments so that the same programming code can be used to access *any* compartment. This is different from the `socket` API which requires the programmer to use the proper `sockaddr_` structure for each "network type" (e.g. `sockaddr_in` for IPv4, `sockaddr_in6` for IPv6, `sockaddr_ll` for link-layer protocols, etc).

To free developers from such constraints, ANA includes a component that permits to map a name or address to the compartment handling the corresponding namespace. To do so, ANA advocates the use of (human-readable) strings to specify CONTEXT values. While this offers a huge flexibility in terms of expressiveness[4], it permits to use regular expressions in order to match CONTEXT values and compartments. For example, the regular expressions `^(\x{2}:){5}\x{2}$` and `^(\x{1,4}:){7}\x{1,4}$` respectively match Ethernet addresses and IPv6 addresses.

With this mechanism, it becomes possible to write address-agnostic applications very easily with for example the following code construct where the strings `ctx` and `srv` are passed as arguments and the IDP 'n' binds to the node compartment.

```
/* returns idp of compartment matching ctx */
idpc = lookup(n, ".", ctx);

/* returns IC to remote service */
idpr = resolve(idpc, ctx, srv);
```

Basically this code is said to be *address-agnostic* because it does not require any a priori knowledge of the type of the `ctx` CONTEXT argument. This means the same code works for `ctx="1.2.3.4"`, `ctx="2001::1"`, or `ctx="00:11:22:33:44:55"` as long as there is a matching compartment for each namespace.

B. Keyword-based description

As a second example, we show how the node local registry in addition to the keyword-based description of functional blocks can greatly enhance the adaptability of applications to new networks and hosts. Let us consider an example application able to communicate either on top of *IPv4* or *IPv6* networks. Upon startup, the application specifies this flexibility by requesting access to an underlying compartment with the following API command :

```
/* n is the IDP to reach the node compartment */
p = resolve(n, ".", "IPv4+IPv6");
```

where the `"."` (local) CONTEXT restricts the query to node local FBs. The '+' character in the SERVICE argument is the boolean *OR* operator. By emitting such a request, the application is asking for functional blocks matching either the `"IPv4"` or `"IPv6"` service description. The node compartment can then return an IDP 'p' leading to one of the two compartments.

The keyword-based description of functional blocks also allows applications to choose among the networking capabilities of the local host. Let us consider an example application that first requires to discover the network services available locally. In ANA, this can be easily achieved by enforcing all functional blocks offering the communication services of a network compartment to publish their presence in the node-local registry with a keyword like e.g. `"compartment"`. This way, our example application can easily discover all the available networks via the simple `lookup` call in the pseudo-code below.

```
/* n is the node compartment's IDP */
IDP_LIST := lookup(n, ".", "compartment");

For all c in IDP_LIST
    /* SERVICE is the description of the peers */
    resolve(c, "*", SERVICE);
```

This primitive call returns the list of IDPs attached to FBs providing network compartment services. The application can then invoke the `resolve` primitive with all the available compartments as shown above. Thanks to the use of generic abstractions (API primitives, `"*"` CONTEXT), this example code can be executed in an agnostic manner on any ANA host. This allows the application to discover its peers via any network compartment implementing the ANA abstractions. This flexibility constitutes an elementary feature towards the adaptability and evolution of autonomic communications software to heterogeneous environments.

C. State migration

As already introduced in Section IV-B, IDPs are used to store information related to the functional block and (if any) the information channel it is bound to. We refer to this information as the *generic part* of the IDPinfo data structure. In addition, each FB can bind/attach some of its internal state to the *specific part* of the IDPinfo. The format of this data is specific to the FB and it is typically used to export and import the FB's internal state when an IDP is being re-bound to another FB.

This mechanism is illustrated by Figure 5. On part (a) of the figure, the TCP-FB-1 exports its internal state related to some connection into the IDPinfo of IDP 's'. For a TCP module, this state typically includes the MSS values, window size, etc, of the connection. On part (b) of the figure, a new functional block TCP-FB-2 is bound to IDP 's' and, as part of the migration process, the new FB imports the IDPinfo data in order to initialize and resume the connection. Note that the application sending data to IDP 's' is not affected[5] by the migration because of the indirection feature offered by IDPs.

Of course, state migration can only work if the old and new versions of a FB use the same data format to encode their internal state. However this requirement, not specific to ANA with respect to state migration, only applies to the specific part

[4]strings impose no structure to data.

[5]except for some transient latency introduced by the migration process.

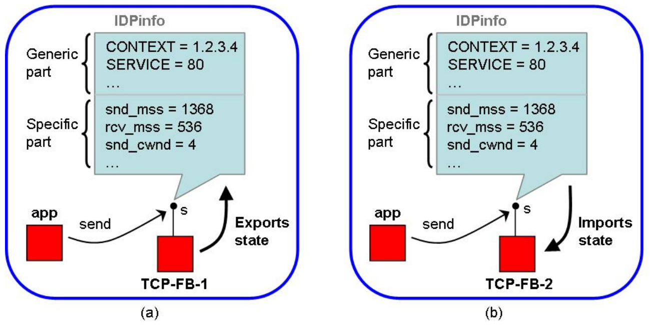

Fig. 5. State migration with IDPinfo.

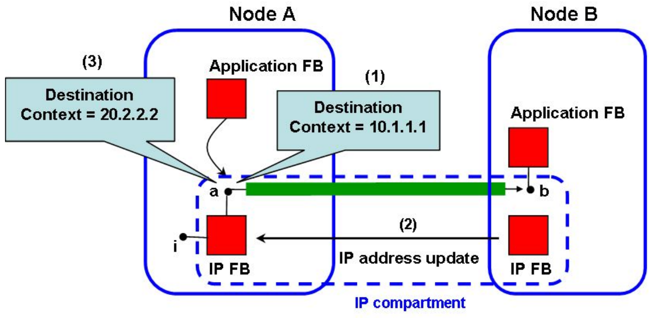

Fig. 6. Mobility support by state exchange at the indirection point

of the IDPinfo. In the case where this requirement cannot be satisfied, the generic part of the IDPinfo can always be used to at least re-setup the information channel.

D. Mobility support

In the following paragraphs, we will demonstrate how the IDP abstraction, along with the IDP information mechanism and the notification system, allow to easily handle node mobility within any network compartment, including IP.

Let us consider the case illustrated by Figure 6. The application FB on node *A* created an information channel towards its peer on node *B* by addressing the following API request to the IP FB on node *A* :

```
/* i is the IDP to reach the IP FB */
a = resolve(i, "10.1.1.1", "appli xyz");
```

where it indicates the IP address `"10.1.1.1"` of the peer to reach as *destination context*. As a result, IDP 'a' is created with the necessary state to forward packets to the destination as shown by action *(1)* in Figure 6.

Let us now suppose that node B changes its physical location which in turn induces a change of IP address. In that case, the IP FB on node *B* can inform its peer on node *A* of its new IP address `"20.2.2.2"` (action *(2)* in Figure 6) which results in an update of the state associated to the IDP 'a' as shown by action *(3)* in Figure 6. Note that this change of IP address is completely transparent to the application FB on node A as it continues forwarding messages via IDP 'a'. Supporting mobility with a level of indirection (i.e. IDP) is in fact very similar to existing mechanisms such as in *HIP* [14]. However, our method can be applied to all network compartments (in contrast with HIP which is specific to IP), and it does not conceal information from the application as it is the case for HIP. Indeed, in ANA, the application can subscribe to events regarding the IDP 'a', such as a change of state. As a result, an event handler is triggered within the application allowing it to extract the new state (IP address) via the IDP information facility described previously.

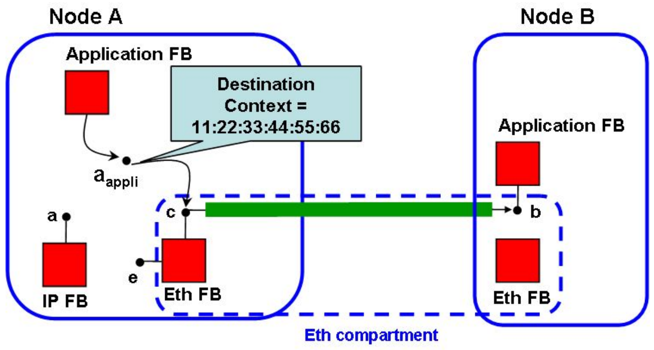

Fig. 7. Transfer of communication between compartments

Beside node mobility, the flexibility of indirection offered by the IDP abstraction also permits to transfer existing communications to other network compartments in a transparent way. To better illustrate this concept let us now suppose that the new physical location of node B lies within node A's LAN (Ethernet compartment). In such a case, transferring the application's communication to an Ethernet information channel would be less error-prone and more efficient as it gets rid of the no longer required IP layer as shown in Figure 7. This can be done with the following steps. First, a new Ethernet information channel leading to the application FB on node *B* is created. To do so, the following command is addressed to the Ethernet compartment (see Section III-C):

```
/* e is the IDP to reach the Ethernet FB */
c = resolve(e, "*", "appli xyz");
```

This results in the creation of IDP 'c' attached to the new information channel. The second step is then to redirect messages sent by the application FB on IDP 'a' towards the newly created IDP 'c', by adding a redirection point with the private IDP 'a$_{appli}$' as shown in Figure 7. At this point, all data sent by the application FB towards its peer on node *B* will be transmitted via the Ethernet information channel. Note that in a similar way to the previous example case, the application FB on node *A* can be notified of the change and extract the new IDP state (new compartment, destination context) using the event notification system and the IDP information mechanism.

Note that this scenario is just a sketch solution demonstrating the advantages of the IDP indirection. More complex mobility scenarios such as simultaneous mobility of the two nodes would require a more complex architecture (e.g. Rendez-vous points) and fall out of the scope of this paper.

E. Remote access to networking functionality

The indirection flexibility offered by the IDP abstraction allows not only to redirect messages and API request between IDPs within the same node but also across remote ANA nodes. In the example of Figure 8 the application running on the sensor mote *A* is communicating over the *IPv6* compartment using the *IPv6* functional block on node *B*. In other words the

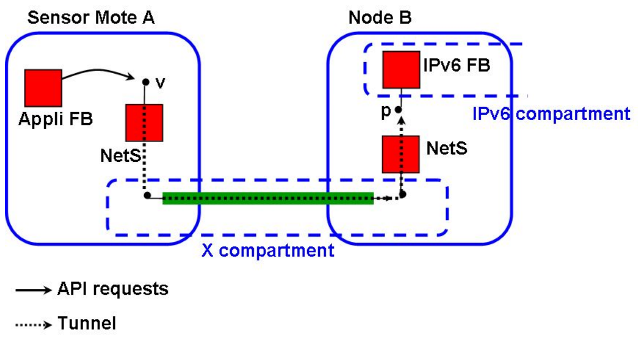

Fig. 8. Access to remote networking functionality

sensor mote *A* is remotely accessing (in an "RPC style") the *IPv6* FB on node B but does not implement any *IPv6* support locally. In this case the two *NetS* components present on both nodes *A* and *B* collaborate in order to *virtualize*, behind the IDP 'v' in node *A*, the activity of the *IPv6* FB on node *B*. The application's API requests (`publish`, `resolve`, `send`, etc.) addressed to the IDP 'v' are tunneled through the network compartment *X* and their results (new IDPs, ICs) are mapped within *A*'s node compartment.

Such a virtualization of remote networking functionality offers multiple advantages. First it allows to easily extend networking capabilities of hosts as they can simply virtualize an access to a remote FB implementing the needed protocols. In fact due to the generic property of the ANA compartment API, the same *NetS* FBs shown on Figure 8 can be used for all network compartments offering to hosts a wide variety of "virtual functionality". The remote access to network compartments also allows to *factorize* networking functionality. Indeed in the example of Figure 8, the *IPv6* FB on node B can be shared among all sensor motes reachable via the network compartment *X* avoiding to all of them the burden of implementing an *IPv6* FB.

Note that this mechanism can be seen as one of many inter-networking solutions allowing hosts within different compartments to communicate. Other solutions include the deployment of an overlay compartment, or solutions based on query flooding and reverse path routing. For further details regarding the tunneling mechanism, we refer interested readers to [15].

VI. Deployment and Security

In this section, we want to clarify the deployment model of ANA and also want to discuss the security issues related to ANA.

While ANA is a clean-slate initiative, our goal is not to replace the current Internet. Our motivation is to develop a flexible framework that can be used to federate existing and future "network instances", and allow researchers to develop algorithms and protocols that can easily interoperate in a heterogeneous environment. For example, the `socket` API can be wrapped with our compartment API, but the communications are still achieved with legacy IP packets and routing. ANA is also not designed to deploy software in a dynamic manner: while ANA offers a flexible framework to instantiate and bind functional blocks, it is not a platform for distributing software. Like most applications today, different versions of a given FB might exist at some time, with or without backwards compatibility, and with different levels of code maturity.

Regarding security and deployment, ANA relies on the same security model as existing software. Installing and executing functional blocks require local access to a machine, access to RAW sockets require root privileges, the resources (e.g. memory, number of IDPs) that can be used by each FB are limited, and (like processes) each FB has private memory and data (e.g. IDPs) that cannot be accessed by other FBs. Moreover, all communications inside ANA is via message passing so there is no risk of having malicious software creating deadlocks by never releasing mutexes for shared resources. Regarding runtime security, ANA does not have any mechanism to control the "network behaviour" (e.g. volume of data sent, etc) of functional blocks: ANA is designed for researchers setting up controlled experiments, not for the public, so there little risk of proliferation of malicious code. If this becomes an issue, ANA could be extended in the future with additional security measures.

VII. Prototype Implementation

All the abstractions and interfaces described above are publicly available as a prototype software [16]. The prototype is developed in `C` for `Linux` platforms and the same code compiles for either user or kernel space. The prototype basically consists of some kind of micro-kernel component called the MINMEX which implements the core functionalities of the ANA node: the management of FBs, the IDPs and their bindings, the event notification system, and the node compartment. Functional blocks (FB) can either be developed as plugins of the MINMEX (either .so extensions or Linux kernel modules) or as stand-alone applications which use standard IPC techniques (i.e. sockets or pipes) to communicate with the MINMEX. FBs can be started or stopped dynamically, and a system is currently being developed to bind the `C` code with a scripting language in order to unify the management of configuration options for all components of ANA. Although the prototype was primarily designed for Linux and Intel computers, it has also been successfully tested on small wireless devices (e.g. Linksys routers with OpenWRT), mobile phones (e.g. Nokia N810, Google Android HTC magic), and Apple systems (MAC OS X, iPhone).

As an example of communication protocols, our stable release incorporates an Ethernet and an IP compartment. Although these compartments were initially developed to demonstrate the modeling and implementation of classical networking using ANA, they do provide some innovative features. As an example, our Ethernet compartment does not rely on a static `EtherType` field to demultiplex incoming packets to higher level services. Instead, it uses the same keyword-set service description scheme, used for the node compartment, to identify higher level services. This allows any protocol or even an application to easily run directly on top of our Ethernet compartment without having to statically bind to a given protocol number (like `0x800` for IP over Ethernet).

The development version of the prototype also includes new network compartments that are still under construction.

These prototype networks cover domains such as field-based [17] and namespace routing [18]. Other approaches of inter-networking such as virtual network stacks [15] are also being explored. In order to be able to simulate different network topologies, a virtual link component called `vlink` was developed. This component permits to build UDP/IPv4 tunnels between different testbed machines and make them appear as virtual links inside the ANA framework. This way, researchers can dynamically simulate different network topologies to test their protocols. Note that our prototype software also permits to run multiple ANA nodes (virtual hosts) on a single network host and make them appear as belonging to different virtual networks. All together, these features offer a wide variety of testing options to help researchers experiment their prototype networks and protocols.

VIII. Related work

Looking back in the literature, the concept of developing a "network architecture that (...) can assemble itself given high level instructions, reassemble itself as requirements change, automatically discover when something goes wrong, and automatically fix a detected problem or explain why it cannot do so." was described by Clark *et al.* in their paper about the *knowledge plane* for the Internet [19]. Clearly the goals of the knowledge plane seem pretty similar to autonomic networking: however unlike ANA, this position paper has remained a visionary exercise and has not materialized into a prototype.

On the management side, the FOCALE [20] and ASA [21] architectures provide frameworks for managing and coordinating the operation of legacy communication systems in an autonomic way. The objective of FOCALE is to orchestrate the behavior of networking components in order to fulfill policies and objectives defined with a high-level language. With similar objectives, the Autonomic Service Architecture (ASA) proposes a framework to manage services and network resources in a uniform manner. Like FOCALE, ASA is focused on the operational complexity of modern communication systems and aims at managing services and network resources in a uniform manner. In contrast to these two frameworks, ANA introduces an abstraction level and machinery for manipulating communication entities in a generic manner at the *system* level. This actually appears like a very interesting match as ANA could be used to develop management systems like FOCALE and ASA as it provides the abstraction layer and flexible machinery upon which such advanced autonomic services could operate.

From past research in network architectures and flexible network stacks, ANA borrows, combines, and extends multiple concepts from various work. For example, the notions of information channel (IC) and functional block (FB) are very close to the notions of n-channel and protocol entity in OSI [10], while the core ANA concept of information dispatch point (IDP) has some similarities with other indirection mechanisms like i3 [13]. Basically the originality of ANA is in the overall integration of these concepts where for example the IDP becomes the core building block of both the API and the low-level packet forwarding machinery. Finally and compared to past work in flexible stacks like xkernel [22] and Click [11], ANA differentiates itself by merging the concepts of packet forwarding with protocol binding by using a single forwarding table where a unique label identifies the next operation (e.g. send packet to either output interface or next functional block) to be performed.

IX. Conclusion

In this paper, we have described the Autonomic Network Architecture (ANA), i.e. a framework and an execution environment for the development and testing of autonomic communications and protocols. To facilitate the interactions of network components, ANA introduces generic networking abstractions and communication primitives that can be manipulated in a generic manner by algorithms and protocols. This is a fundamental requirement for autonomic entities which must dynamically adapt to changing environments and networking components. In addition, ANA is designed as a *meta-architecture* in order to promote the evolution, adaptation, and federation of novel networking schemes and protocols: the key is to offer a generic development framework that can be used to model, develop, and deploy forward-looking communication systems and protocols.

In 2009, the ANA project is currently integrating the different autonomic components into the final software prototype that will be publicly released at the beginning of 2010. Among others, these components include e.g. self-organizing intra and inter-domain routing protocols, self-adaptive addressing protocols, a network monitoring framework, and a system for dynamically composing "protocol stacks" from the pool of available components. In addition, the project is currently finalizing the development of an embedded scripting facility that will allow developers to specify the operation of ANA components in a flexible way. Future work include the development of an improved system for specifying and searching the capabilities of each network component (in order to compose network stacks more efficiently) and the specification of the architecture and its abstractions and communication primitives in a more formal manner. In addition and with the experience gained during the integration and testing phase in 2009, we want to further refine the operation of the low-level machinery of ANA in order to come up with a mature and final specification of ANA in early 2010.

Acknowledgement

The 4-year (2006-2009) ANA Project (FP6-IST-27489) [8] is funded by the European Commission under the proactive initiative on "Situated and Autonomic Communications" (SAC). Some of the ideas presented in this paper result from work and discussions with other members of the ANA project. The authors of this paper hence do not claim exclusive credit with respect to the content.

References

[1] M. Blumenthal and D. Clark, "Rethinking the design of the Internet: The end to end arguments vs. the brave new world," *ACM Trans. Internet Technol.*, vol. 1, no. 1, pp. 70–109, August 2001.

[2] D. Clark, K. Sollins, J. Wroclawski, and R. Braden, "Tussle in Cyberspace: Defining Tomorrow's Internet," in *Proc. ACM SIGCOMM 2002*, August 2002, Pittsburgh, USA.

[3] T. Anderson, L. Peterson, S. Shenker, and J. T. (Editors), "Report of NSF Workshop on Overcoming Barriers to Disruptive Innovation in Networking," January 2005, Available at http://www.geni.net/documents.php.

[4] J. Crowcroft and P. Key, "Report from the Clean Slate Network Research post-SIGCOMM 2006 Workshop," *ACM CCR*, vol. 37, no. 1, pp. 75–78, January 2007.

[5] "FIND – Future Internet Design (FIND)," at http://www.nsf.gov/pubs/2006/nsf06516/nsf06516.htm.

[6] "Future Internet Research and Experimentation (FIRE) initiative - European Commission," at http://cordis.europa.eu/ist/fet/comms-fire.htm.

[7] J. Kephart and D. Chess, "The Vision of Autonomic Computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, January 2003.

[8] "Autonomic Network Architecture - http://www.ana-project.org."

[9] J. Crowcroft, S. Hand, R. Mortier, T. Roscoe, and A. Warfield, "Plutarch: an Argument for Network Pluralism," in *Proc. ACM SIGCOMM Workshop on Future Directions in Network Architecture*, August 2003, Karlsruhe, Germany.

[10] H. Zimmermann, "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection," *IEEE Trans. Commun.*, vol. 28, no. 4, pp. 425–432, April 1980.

[11] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. Kaashoek, "The click modular router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, August 2000.

[12] C. Tschudin and R. Gold, "Network Pointers," in *Proc. First ACM Workshop on Hot Topics in Networks (HotNets-I)*, October 2002, Princeton, NJ, USA.

[13] I. Stoica, D. Atkins, S. Zhuang, S. Shenker, and S. Surana, "Internet Indirection Infrastructure," in *Proc. ACM SIGCOMM 2002*, April 2002, Pittsburg, USA.

[14] R. Moskowitz and P. Nikander, "RFC-4423 Host Identity Protocol (HIP) Architecture," 2006.

[15] G. Bouabene, C. Jelger, and C. Tschudin, "Virtual network stacks," in *Proc. the ACM SIGCOMM workshop on Programmable Routers for Extensible Services of TOmorrow (PRESTO'08)*. New York, NY, USA: ACM, 2008, pp. 45–50.

[16] "Autonomic Network Architecture prototype," http://www.ana-project.org/web/software/start.

[17] T. Hossmann, A. Keller, and M. May, "Implementing the Future Internet: A Case Study of Service Discovery using Pub/Sub in the ANA Framework," in *Proc. International Conference on Future Internet Technologies (CFI'08)*, June 2008, Seoul, Korea.

[18] C. Jelger, G. Bouabene, and C. Tschudin, "Routing Namespace Patterns," September 2008, Technical Report CS-2008-01 available at http://informatik.unibas.ch/research/publications_tec_report.html.

[19] D. Clark, C. Partridge, J. Ramming, and J. Wroclawski, "A Knowledge Plane for the Internet," in *Proc. ACM SIGCOMM 2003*, August 2003, Karlsruhe, Germany.

[20] B. Jennings, S. van der Meer, S. Balasubramaniam, D. Botvich, M. Foghlu, W. Donnelly, and J. Strassner, "Towards Autonomic Management of Communications Networks," *IEEE Commun. Mag.*, pp. 112–121, October 2007.

[21] Y. Cheng, R. Farha, M. S. Kim, A. Leon-Garcia, and J. W.-K. Hong, "A Generic Architecture for Autonomic Service and Network Management," *Computer Communications*, vol. 29, no. 18, pp. 3691–3709, November 2006.

[22] N. Hutchinson and L. Peterson, "The *x*-Kernel: An Architecture for Implementing Network Protocols," *IEEE Trans. Softw. Eng.*, vol. 17, no. 1, pp. 64–76, January 1991.

Ghazi Bouabene obtained his MSc in computer science in 2006 from the University of Strasbourg, France. He is currently a PhD candidate at the University of Basel in the Computer Networks Research Group. His research interests are on the design and implementation of flexible network architectures supporting novel self-organizing protocols.

Christophe Jelger is a senior researcher at the University of Basel, Switzerland. His current research interests include future network architectures and system design, addressing and routing protocols, and ad hoc networking. He holds a PhD from the Louis Pasteur University (Strasbourg, France), and an MPhil from the University of Swansea (Wales, UK).

Christian Tschudin is a full professor for computer networks at the University of Basel. Before joining the University of Basel, he was at Uppsala University as well as ICSI in Berkeley, and did his PhD at the University of Geneva. He is interested in mobile code, artificial chemistries, wireless networks and security.

Stefan Schmid received his M.Sc. in Computer Science (Dipl.-Inf.) from the University of Ulm, Germany, in 1999. In November 2002, he graduated from Lancaster University with a Ph.D. in Computer Science. After his post doctorial research at Lancaster University, he joined NEC Laboratories Europe in Heidelberg, where he now leads the Next Generation Networking research group. He is actively involved in several EU research projects in the area of future autonomic networks. In addition, he attends and contributes to the standardization of core network and systems aspects in 3GPP.

Ariane Keller completed her Master's degree in Electrical Engineering and Information Technology at ETH Zurich in 2008; since then she is pursuing her doctoral studies in the Computer Engineering and Networks Laboratory, also at ETH Zurich. Her research interests lie in future network technologies and system architectures; she is interested in building the fundamental technologies upon which future communications infrastructures can be based.

Martin May received his PhD in 1999 for his work on Internet QoS mechanisms at INRIA, Sophia Antipolis, France. Afterwards, he spent his research as a post-doc at Sprintlabs, Burlingame, USA, followed by the foundation of a start-up in France where he worked in the field of Content Delivery Networking. After selling the company end of 2003, he joined the Swiss Institute of Technology in Zurich (ETH Zurich) as a senior research associate. In 2008, Dr. May joined the Thomson Research Lab in Paris as senior scientist where he leads the research activities in wireless and mobile networking as well as Future Internet technologies.