Future Internet - ANA

INF5090 – Thomas Plagemann

Overview

• Motivation
• Blueprint
• ANA core
• Monitoring Framework

Recap – why a Future Internet?

Motivation

• Variability in the Internet is above and below IP: it's the “hourglass” model.

[Diagram showing layers of the Internet with applications like www, email, ftp, ssh, DNS, peer-to-peer (eMule, BitTorrent), VoIP (Skype), VoD, grid, ... and network layers like Ethernet, wifi (802.11), ATM, SONET/SDH, FrameRelay, modem, ADSL, Cable, ...]

Changing/updating the Internet core (e.g., IPv6, Multicast, MIP, QoS, ... ) is difficult or impossible!
Motivation

- Solutions adopted so far:
  - Patches to cope with challenges contradict the initial design paradigms
- Incoherencies → patches to the patches

Consensus in the research community that a next step beyond the Internet is needed.

ANA: finding the least common denominator

- ANA abstractions allow variability at all levels of the architecture:
  - variants to perform a given task,
  - and networks co-exist and (can) compete, open for extensions (evolution)

Our contribution: enable & demonstrate autonomic networking.

Pitfalls in network architecture design

- Static/rigid standards instead of mechanisms for change management
- Global address space (requires uniqueness and global coordination)
- Leaking of and relying on network internal details
- Built-in address dependency (i.e. address-centric architecture)
- To avoid running into such pitfalls, we adopt an incremental approach via prototyping cycles
- Helps revealing faults or black-holes in the architecture design

You can’t “build” an architecture, you have to "grow" it

- Project is articulated around 2 prototyping cycles.
- Methodology: design, test/validate, refine.

<table>
<thead>
<tr>
<th>2006</th>
<th>2007</th>
<th>2008</th>
<th>2009</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design phase: First “Blueprint” (architectural model)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>First prototyping phase</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Testing + feedback phase</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd prototyping phase</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final evaluation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd design phase: Mature “Blueprint”</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ANA ≠ "one-size-fits-all"

- ANA does not want to propose another "one-size-fits-all network waist".
- ANA is a meta-architecture to host, interconnect and federate multiple heterogeneous networks.

Core abstractions

- ANA abstractions:
  - Compartment.
  - Information Channel (IC).
  - Information Dispatch Point (IDP).
  - Functional Block (FB).

- More tricks with compartments:
  - "Node compartment".
  - Overlays and compartment inter-stitching.

ANA Compartment

- Compartment = wrapper for networks.
- ANA does not impose how network compartments should work internally: the ANA framework specifies how networks interact.

Compartments (cont.)

- A (network) compartment defines how to join and leave a compartment:
  - member registration, trust model, authentication, etc.
- Each compartment defines a conceptual membership database.
- Registration: explicit joining and exposing is required ("default-off" model).
Compartments (cont.)

- Defines **How to reach** (communicate with) another member: peer resolution, addressing, routing, etc.

- Resolution: explicit request before sending ("no sending in the void").

Resolution process returns communication entry point. How resolution is performed is specific to each compartment.

Compartments can be overlaid, i.e., compartments can use the communication services of other compartments.

- Each compartment is free to use any addressing and naming schemes (or is free to not use addresses, for example in sensor networks).

Compartments decompose communication systems and networks into smaller and easier manageable units.

What about addresses and names?

- The main advantages are:
  - No need to manage a unique global addressing scheme.
  - No need to impose a unique way to resolve names.
  - ANA is open to future addressing and naming schemes.

- The main drawbacks are:
  - Back to the CATENET challenges: How to inter-work in such heterogeneous address/name spaces?
Local labels for handling (global) addresses

- Target resolution returns a local label = IDP
- Addresses (if any) and names (if any) limited as input for resolution
- The IDP maintains the state to reach the destination

Applications send data only to IDPs
- Bound to a flexible element

Resolution process returns communication entry point

Startpoints instead of endpoints.

Information Channels (ICs)

- Resolution process returns access to an “information channel” that can be used to reach the target member(s).
- Various types of information channels.

Core abstractions

- ANA abstractions:
  - Compartment.
  - Information Channel (IC).
  - Information Dispatch Point (IDP).
  - Functional Block (FB).

- More tricks with compartments:
  - “Node compartment”.
  - Overlays and compartment inter-stitching.
**Functional Blocks (FBs)**

Code and state that can process data packets.

- Protocols and algorithms are represented as FBs.
- FBs implement the Information Channels (abstract entities).
- Access to FBs is also via information dispatch points (IDPs).
- FBs can have multiple input and output IDPs.
- FB internally selects output IDP(s) to which data is sent.

- Data is sent to IDP which has state to call correct function inside FB.

**How ICs, FBs, and IDPs fit together**

**Modelling nodes as compartments**

- Organize a node’s functionalities as (compartment) members:
  - Member database: catalog of available functions.
  - Resolution step to access a given function.
  - Also implements access control.
  - Resolution instantiates functional blocks (FBs).
  - The node compartment hosts/executes FBs and IDPs.

**Different “views” for a compartment**

A network compartment has different views, for different usage.
**Basic design**

**Two main components:**
- The MINMEX i.e., the ANA "micro-kernel"
  - Supports core API and inter-brick communications.
  - Implements packet dispatching to IDPs.
  - Implements the Node Compartment.
- "Bricks" i.e., individual components of the ANA Playground
  - Can be a functional block which offers access to a network compartment.
  - Can also provide processing support (e.g. encryption).

**Basic design**

The ANA node can be distributed.
- We do not enforce that all the components of an "ANA Node" run on the same computer.
- The MINMEX and its bricks can communicate via various IPC types called "gates": Unix/UDP sockets, named pipes, generic netlink.
- The motivation was that the notion of a "node" was not restricted to being a physical device.

**Basic design**

Three different API levels.
- Objective: better understand the level of complexity vs. flexibility we want to reach.
  - API Level 0: maximum flexibility but developer must know all the details for encoding and decoding messages.
  - API Level 1: good flexibility and developer can use functions to encode and decode messages.
  - API Level 2: less flexibility, but function prototypes are very easy to use, code is easy to write.
Basic design

One code, two platforms.

- The base code compiles as either userspace application or Linux kernel modules
  - Userspace: easy for development and debugging, easy to use, most people can use it.
  - Linux kernel: for best performance, permits to interact with kernel network internals.
- We also started lately a support for ANA bricks written in the Erlang Programming language

Available components

- MINMEX: node compartment, IDT, IDP manipulation, status interface, API libraries.
- Bricks:
  - Ethernet and IP compartments.
  - vlink sub-system for flexible "cabling" of ANA nodes.
  - Monitoring components: core part, packet capture, measurements (CPU, net load), "ping".
  - Inter-compartment routing with regular expressions.
  - Content centered routing, Field Based routing
  - Functional composition prototype.
  - User side: chat application, various code examples.
  - And many other bricks of project partners
- Tools
  - QuickRep, timers, threads, Remote IDP Access

Basic design

One code, two platforms.

- Bricks can be developed in a "platform-agnostic" way according to a standard template.
  - The API library provides wrapper functions and mechanisms to properly handle function calls (e.g., malloc vs. kmalloc, main vs. ini)
  - We also provide "agnostic" libraries for system-specific functions such as threads and timers.
  - Passing arguments to bricks via CLI is also done via a system-agnostic mechanism (à la argc/argv)

Monitoring - Motivation

Efficient and "user friendly" monitoring with ANA properties for all ANA developers
Goals

- Extensibility:
  - arbitrary measurement metrics and bricks can be added at any time: no a priori knowledge
- Self-configuration:
  - without manual configuration
- Self-optimization:
  - optimize its service by adapting to client requirements, available bricks and current context
- Adjustable accuracy
- Efficiency
- Transparency

Conceptual Model

1. Client requests:
2. Orchestration
3. Measuring FB
4. Measuring FB
5. Measuring FB
6. Measuring FB
7. Data storage (RAM, DB...)
8. Return results:
9. Dispatcher
10. Identify metric
11. Select Measuring FB
12. Client request
13. Dispatcher
High-level Task of IMF

- Client brick forwards a monitoring request to IMF including the following fields:
  - Nonce, Type, Type parameters, Metric, Metric parameter, Non-functional requirements, Reply IDP
- IMF analyses request:
  - Which bricks can provide data for the specified metrics?
  - If more than one brick – which is the most suitable one (depends on context and client requirements)
  - Dispatcher forwards client monitoring request to the selected brick

Challenge 1

- How to achieve extensibility and avoid a priori knowledge?
- Who knows which metrics can be monitored?
  - Minimal assumption:
    - Each measuring brick developer knows which metrics the brick measures
    - Each measuring brick developer knows the context that is relevant for the brick
    - Each measuring brick developer knows the non-functional properties of the brick
    - New bricks provide the information to IMF and IMF manages it

Challenge 2

- How to select the best suited measuring brick?
  - Available bricks
  - Brick properties
  - QoS requirements of client
  - Context

Context & Brick Properties

- Example:
  - Vivaldi brick: /* Impact on non-functional properties */
    - Always: resource consumption = low
    - mobile: accuracy = bad
    - Network type =
      - static: accuracy = medium
  - Ping brick:
    - Always: resource consumption = medium
    - Always: accuracy = good
Context Usage

• The context defines how the monitoring bricks are performing in the given situation (=context)
• The monitoring request includes the particular requirements, e.g. prefer high accuracy or low resource usage
• Example of a simplified monitoring request:
  • Latency: <accuracy, medium or better, medium weight>, <resource consumption, low, high weight>

Context Usage (cont.)

• Instantiate the monitoring brick property descriptions and compare which one matches best the monitoring request
• Example:

Monitoring brick properties
- Vivaldi brick:
  Always: resource consumption = low
  Mobile:
  Network type = Network type = 
  Accuracy = bad
- Ping brick:
  Always: resource consumption = medium
  Accuracy = good

Context:
- Network type = mobile

Monitoring request:
- Latency: <accuracy, medium or better, medium weight>, <resource consumption, low, high weight>
- ping matches best

IMF Building Blocks

Vocabularies

• Assumptions:
  • Vocabularies are shared
  • Agreement on a set of vocabulary specification trees (VSTs)
  • Vocabularies are extensible
    • New subconcepts can be added in time
  • Information sharing
    • Each brick in general knows a subtree (a pruned version) of each VST
    • The IMF knows the VSTs composed of the bricks’ subtrees
Shared Vocabularies

- BrickVST
- Brick types
- New bricks announce where they fit in the BrickVST
- MetricsVST
- Metrics
- New bricks announce their metrics and where it fits in the MetricsVST
- ContextVST
- Context types and corresponding values
- QoSVST
- QoS parameters and corresponding values
- New bricks announce their QoS profile
- QoSRequirementsVST

Example: Brick VST

In BrickVST brick names like Ping and VC are probably not part of the shared vocabulary but are decided by the brick providers.

Example: MetricsVST

Both metric types and metric names are part of the shared vocabulary.

Example: ContextVST

Context types and possible values.
Example: QoSVST

QoS Parameter

- Accuracy
- Resource Consumption

Possible values:
- Bad
- Medium
- Good
- Low
- Medium
- High

QoS parameter names and corresponding possible values

Example: Metrics of Context

Bricks name

- Context Type
- Network Context
- Resource Context
- Node Type Context
- Battery Context
- CPU Context
- Battery
- CPU

Context values:
- Mobile
- Static
- Node A
- Node B
- Battery X
- Battery Y
- CPU

Context type:
- Network
- Resource
- Node Type
- Battery
- CPU

Metric type:
- Context

Information Model

Brick type | Brick name | Context type | Context value
---|---|---|---

QoS Profile

Brick type | Brick name | Context type | Context value | Context value
---|---|---|---|---

Summary

- ANA blueprint:
- IDPs
- ICs
- Compartments
- FBs and bricks
- Polymorphic Internet
- ANA core
- Monitoring Framework & semantics