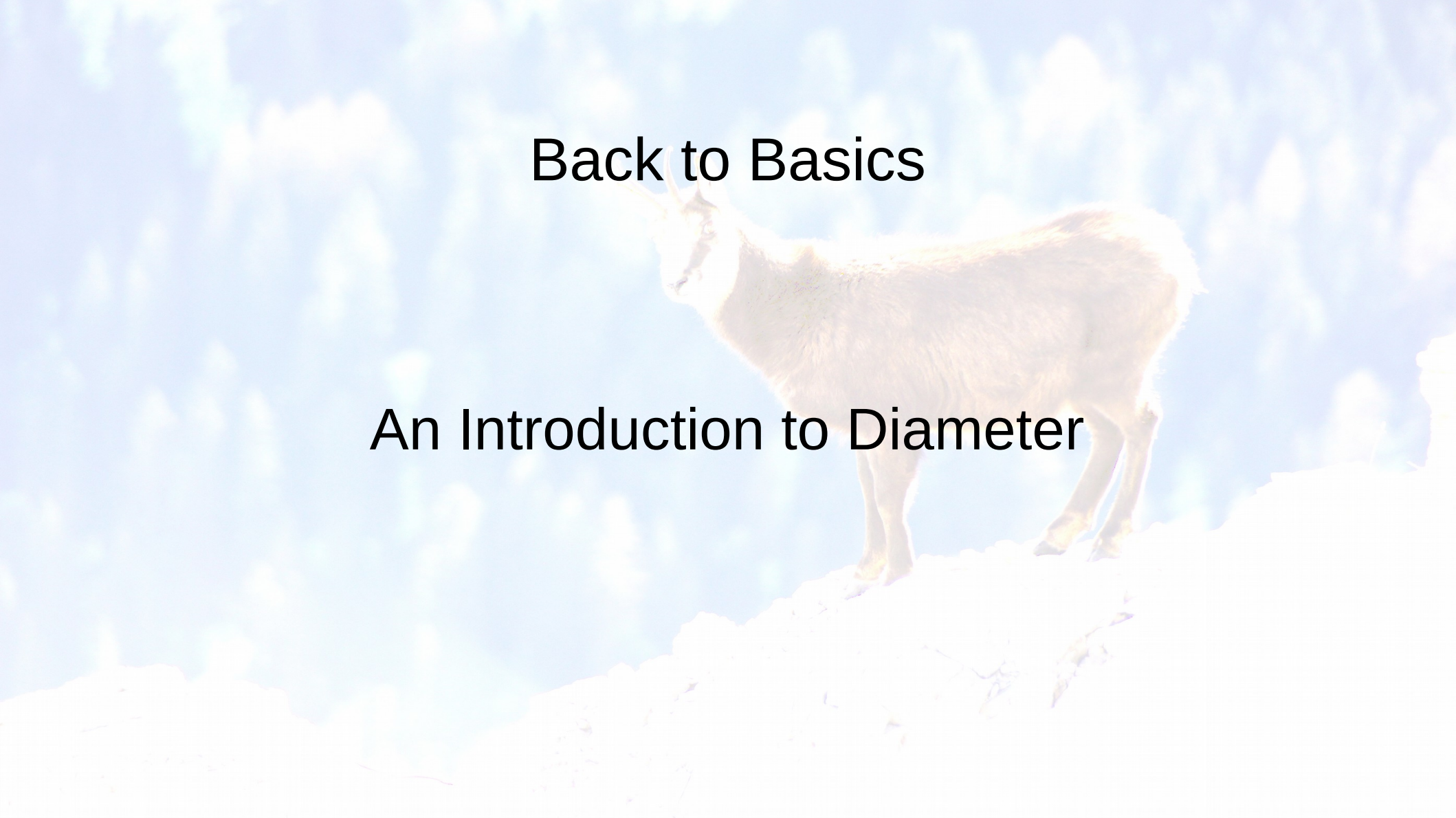
Back to Basics



An Introduction to Diameter

Who am I?

- Software engineer @ Athonet S.r.l.
- NOT a professional Erlang/Elixir developer (yet!) :(
- Mainly developed in Python, C, C++
- Full-est stack developer
- Started exploring the Beam with Elixir
-
- (not very good at slides)
  [and those were originally at 4:3, sorry!]

GLOMO Global Mobile Awards 2019 WINNER

ATHONET

Why this talk?

- Because I was working on this :)
- To re-discover the basics of the framework we're using
- To see something unusual
- (because you may say “I want to do this!”, and we are always searching for talents :))

AAA – Protocol wanted!

What is **Diameter**?

- **A**uthentication, **A**uthorization and **A**ccounting network protocol: framework for applications such as network access or IP mobility
- Application layer protocol
- Evolution of **RADIUS** (Joke intended) [seriously]
- Designed to be an AAA framework for next generation applications
- ~~RFC3588~~RFC 6733 (Diameter Base Protocol, ~~2003~~2012) and many others
- Widely used in 3GPP Control Plane protocols (authentication, billing, location services…)

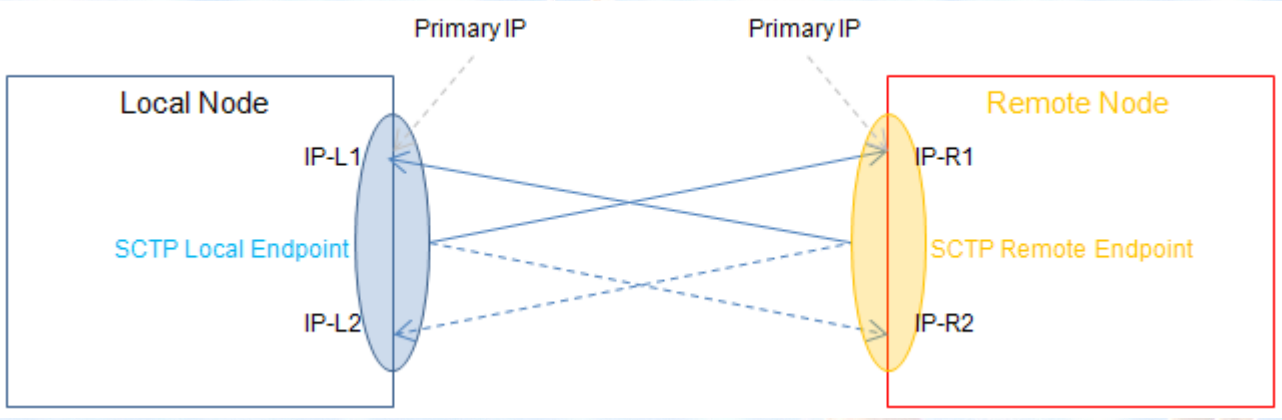
Diameter == 2 * RADIUS

RADIUS	DIAMETER
UDP Transport	TCP or **SCTP** transport
Client – Server, mono directional communication	Peer-to-peer, bidirectional communication (Server generated messages)
Manual configuration of nodes and secret (some RFC7585 for dynamic discovery through NAI)	Dynamic peer discovery using DNS and NAPTR, SRVLOC
8 bit identifiers	32 bit identifiers
Stateless	Both stateful (session, transaction) and stateless
No congestion control	Capacity, congestion control, overload (RFC 7660, 7068, ...)
Intermediate nodes (generically referred as Proxy, no specifications)	Intermediate nodes (Agents: Relay, Proxy, Redirect, Translation)
	Capabilities exchange, applications

Diameter = 2 * RADIUS (pt. 2)

RADIUS	DIAMETER
Silent failures: invalid messages are discarded	Informational Application Errors (or even success!) or Protocol Errors (“E” header bit set): agents may try to correct them
Extension at attribute level	Extension at AVP, procedure or application level. Even Base Protocol can be extended. (new AVP codes assigned by IANA. New commands by IETF consensus)
Can support IPSec (not mandatory)	Clients must support at least one between IPSec or TLS. Diameter CMS Security Application is also available.
Unstructured messages	Mandatory flags, positional requirements
	Loop detection (Route-Record), transaction path recording
	...not fully backwards compatible!

SCTP



- Message oriented (like UDP)
- In-sequence transport, congestion control (like TCP)
- Multi-homing (also asymmetric): redundancy, transparent fail-over.
- Concurrent transition of different streams of data
- Can map to TCP API
- Supported by Linux Kernel >= 2.4

Used by:

- SSH
- WebRTC
- ...Diameter!

Not used, why?

- TCP was already in (wide) use!
- TCP mostly works

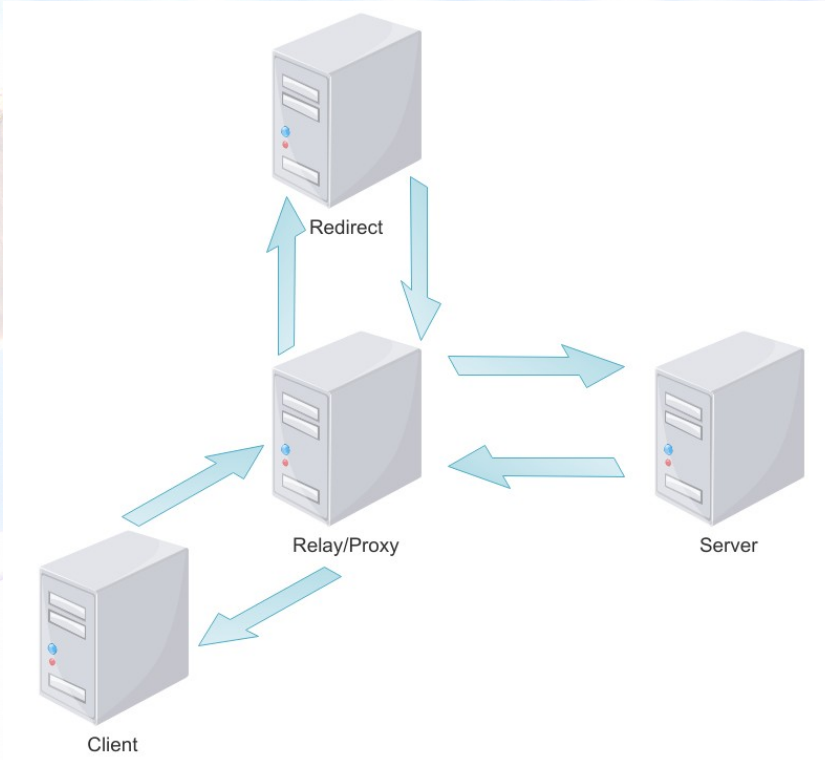
Architecture (1)

Entities in the Diameter Architecture:

- **Node**: host implementing Diameter protocol
- **Peer**: Diameter node directly connected with another Diameter node
- **Client**: device at the edge of the newtork that performs access control (*MME* – Mobility Management Entity -, *PCEF* – Policy and Charging Enforcement Function - in EPS architecture)
- **Server**: node that handles AAA functions for a particular realm (*HSS* – Home Subscriber Server -, *PCRF* – Policy and Charging Rules Function – in EPS architecture)

Architecture (2)

- Agent: node providing “networking” functions
  - ***Relay Agent***: agent that accept requests and route messages to other Diameter nodes based on message information (e.g. Destination-Realm). **No** application level processing. Relay Agents modify Diameter messages by inserting and removing routing information, and nothing more. Relays **should not** maintain session state but **must** maintain transaction state
  - ***Proxy Agent***: route Diameter messages using the Diameter Routing Table (like Relay Agents). **Can** modify messages to implement policy enforcement. Proxies **may** maintain session state and **must** maintain transaction state. Since enforcing policies requires an understanding of the service being provided, Proxies must only advertise the Diameter applications they support. Example: *Diameter Routing Agent* (DRA)
  - ***Redirect Agent***: do not relay messages, only return an answer with the information necessary for direct communication with destination. Do **not** modify messages. Do **not** receive answer messages, so do **not** maintain session state. Since they never relay requests, they are **not** required to maintain transaction state. Do **not** perform any application level processing, they provide relaying services for all Diameter applications, therefore **must** advertise the Relay Application Identifier. Example: *Subscription Locator Function* (SLF) in IMS
  - ***Translation Agent*** : translates between two protocols (e.g. RADIUS to Diameter, MAP to Diameter) allowing server conversions to Diameter, for example, while permitting client **NAS**es to be converted at a slower pace



Applications

Diameter offers the concept of *Application*: not a software, but an extension over the base protocol

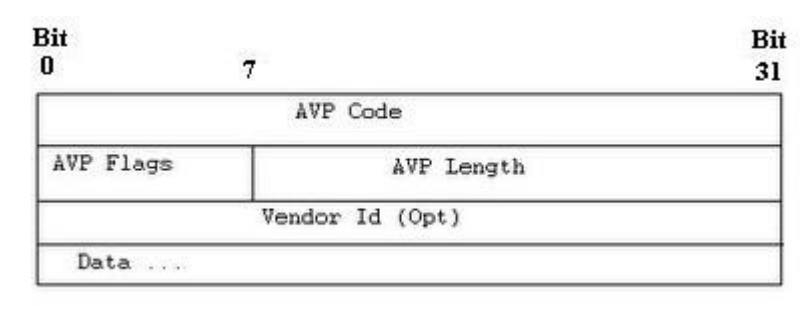
Examples:

- Diameter Credit Control Application (DCCA): real time credit control for end-user services (rfc4006)
- Cx/Dx interface: AAA for VoLTE/IMS systems
- Sh interface: data handling procedure for the Ip Multimedia Subsystem
- Diameter-WebAuth: draft for identity based for Web Application AAA

Each applications offers commands (Request/Answer) to achieve its purpose.

AVP

Like RADIUS, the protocol is based on AVP (Attribute-Value Pairs)



Field	Size	Description
AVP Code	4 bytes	Combined with Vendor ID, identifies the attribute uniquely. 1-255 reserved for **RADIUS** backwards compatibility
AVP Flags	1 byte	• **M – Mandatory bit**: if set Client, Server, Proxy and Translation agents **must** support AVP handling • **V – Vendor bit**: if set, the Vendor-Id AVP **shall** be present in the message • **P – Protected bit**: if set, the AVP content is encrypted for end-to-end security
AVP Length	3 bytes	Data + Vendor-Id + AVP Code + AVP Length + AVP Flags
Vendor Id	4 bytes	IANA Assigned identifier, for Vendor-specific extensibility

AVP (2)

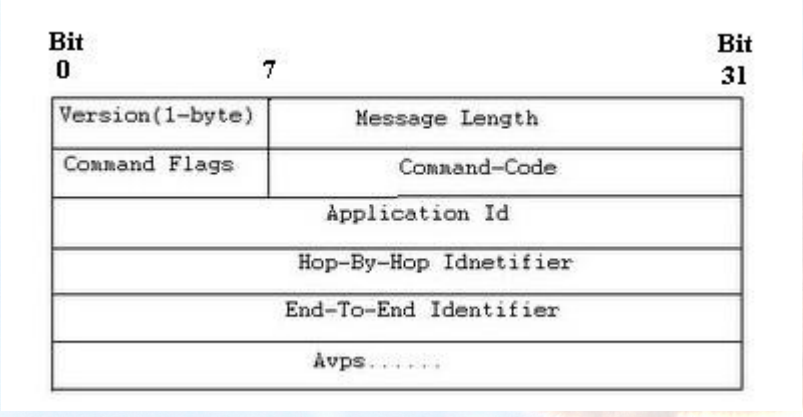
Basic AVP Types (unsurprisingly)

- OctetString
- Integer32/64
- Unsigned32/64
- Float32/64
- Grouped

Derived AVP Types

- Time
- Address
- UTF8String
- DiameterIdentity
- DiameterURI
- Enumerated
- IPFilterRule

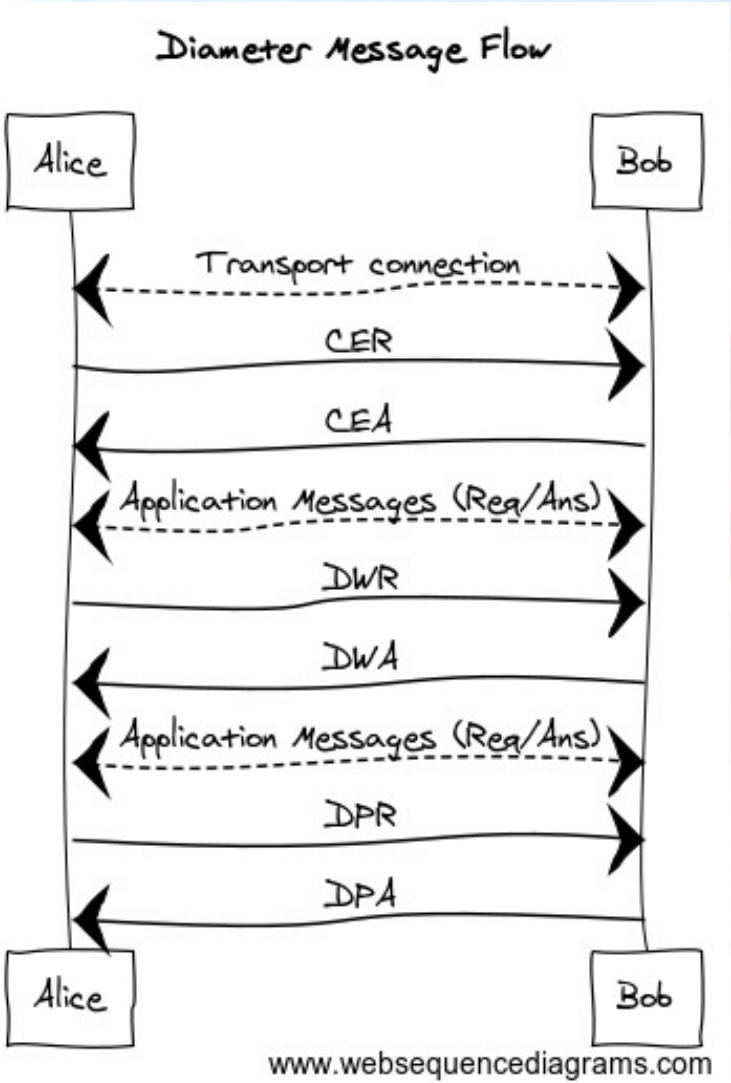
Packet



Field	Size	Description
Version	1 byte	Protocol version (well, at the moment: 1)
Length	3 byte	Let's guess :) (header + padded AVPs)
Command Flags	1 byte	• **R (Request)** bit: the message is a request (if set) or an answer • **P (Proxiable)** bit: the message *may* be proxied, relayed or redirected (if set) or *must* be locally processed • **E (Error)** bit: message contains a protocol error (if set) • **T (Potentially re-transmitted)** bit: set when resending requests not yet acknowledged as an indication of a possible duplicate
Command Code	3 bytes	• 0-255 RADIUS backwards compatibility • 256-16777213 Standard commands (allocated by IANA) • 16777214-16777215 Experimental (testing)

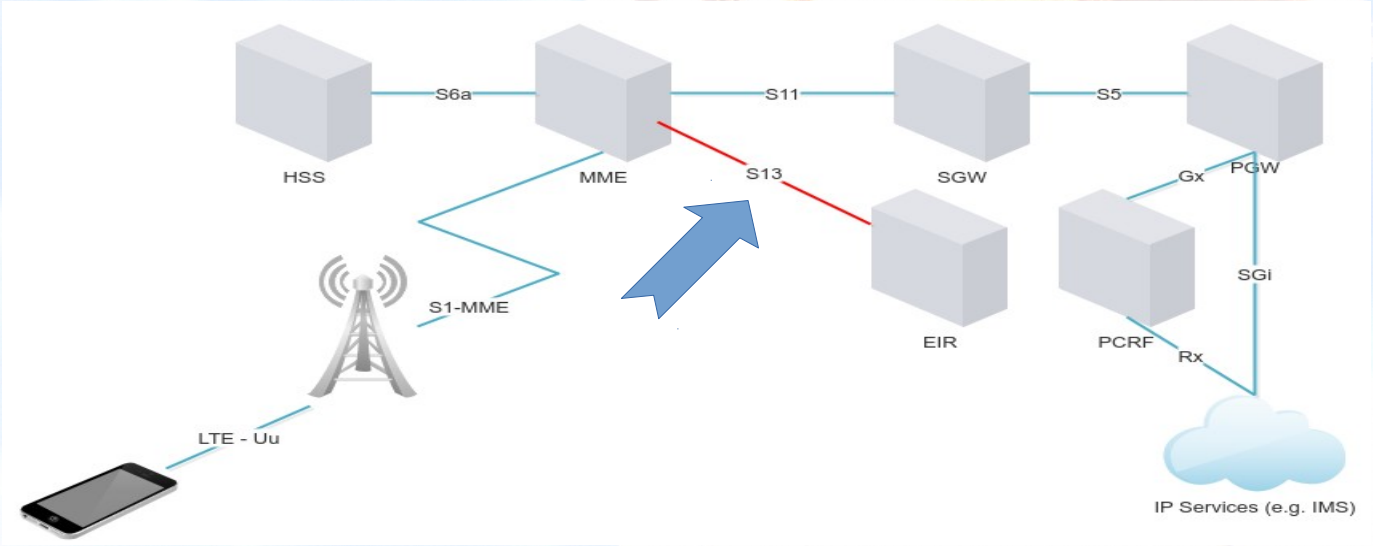
Packet (2)

Field	Size	Description
Application Id	4 bytes	Identifier of the application (standard or vendor-specific). Same values are repeated in each Application-Id AVP.
Hop-By-Hop Identifier	4 bytes	Identifier to match requests and responses and keep transactions states
End-To-End Identifier	4 bytes	Identifier to avoid duplicate requests (combined with origin-Host AVP)
AVPs	...	Command-specific content


Diameter Message Flow
Alice
Bob
Transport connection
CER
CEA
Application Messages (Req/Ans)
DWR
DWA
Application Messages (Req/Ans)
DPR
DPA
Alice
Bob
www.websequencediagrams.com

What will we build?

Equipment Identity Register (EIR)



Single message:

MME: “Do you know this user (IMSI) using this equipment (IMEI)?”

EIR: “No/Yes, and it's good/bad/whatever”

Why eDiameter?

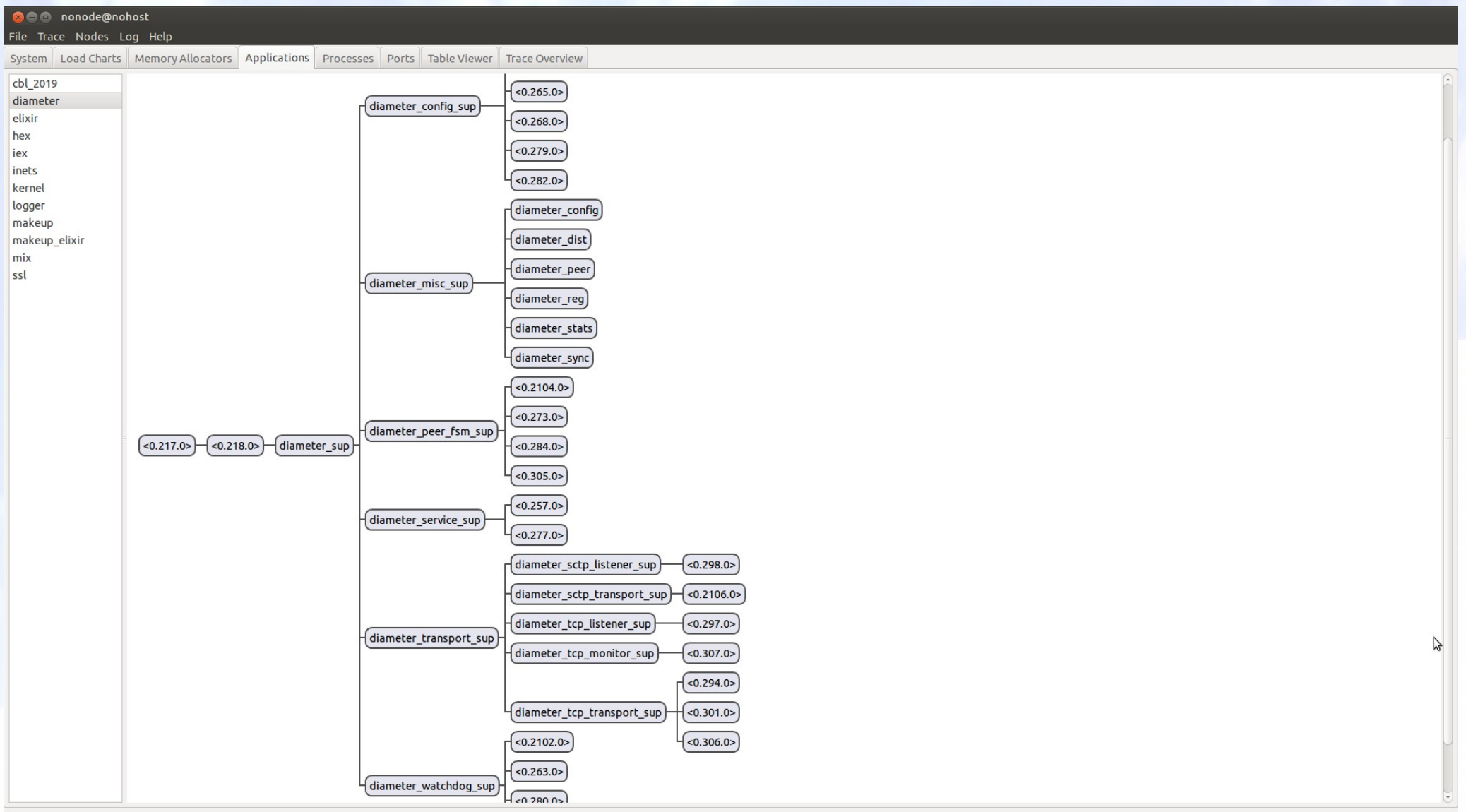
- Written initially by Ulf Wiger
- Need for a DIAMETER client for a Session Border Gateway Controller (client)
- 3rd party stacks expensive/incomplete/not customizable
- Re-written by Andres Svensson => OTP
- How to test a client? Let's add a server!

Caveats:

- RFC collapses 3 state machines into one (single-threaded event loop)

Solution:

- Transport FSM, Handshake FSM, Service FSM, Request handler (one per request)
-
- (More on the referenced slides and videos :))


nonode@nohost
File Trace Nodes Log Help
System Load Charts Memory Allocators Applications Processes Ports Table Viewer Trace Overview
cbl_2019
diameter
elixir
hex
iex
inets
kernel
logger
makeup
makeup_elixir
mix
ssl
<0.217.0>
<0.218.0>
diameter_sup
diameter_config_sup
<0.265.0>
<0.268.0>
<0.279.0>
<0.282.0>
diameter_misc_sup
diameter_config
diameter_dist
diameter_peer
diameter_reg
diameter_stats
diameter_sync
diameter_peer_fsm_sup
<0.2104.0>
<0.273.0>
<0.284.0>
<0.305.0>
diameter_service_sup
<0.257.0>
<0.277.0>
diameter_transport_sup
diameter_sctp_listener_sup
<0.298.0>
diameter_sctp_transport_sup
<0.2106.0>
diameter_tcp_listener_sup
<0.297.0>
diameter_tcp_monitor_sup
<0.307.0>
diameter_tcp_transport_sup
<0.294.0>
<0.301.0>
<0.306.0>
diameter_watchdog_sup
<0.2102.0>
<0.263.0>

eDiameter

http://erlang.org/doc/man/diameter.html

https://github.com/erlang/otp/tree/master/lib/diameter

“Basic usage consists of creating a representation of a locally implemented Diameter node and its capabilities with ***start_service/2***, adding transport capability using ***add_transport/2*** and sending Diameter requests and receiving Diameter answers with ***call/4***. Incoming Diameter requests are communicated as callbacks to a ***diameter_app(3)*** callback modules as specified in the service configuration.”

From Specification...

```
< ME-Identity-Check-Request > ::=     < Diameter Header: 324, REQ, PXY, 16777252 >
                                      < Session-Id >
                                      [ DRMP ]
                                      [ Vendor-Specific-Application-Id ]
                                      { Auth-Session-State }
                                      { Origin-Host }
                                      { Origin-Realm }
                                      [ Destination-Host ]
                                      { Destination-Realm }
                                      { Terminal-Information }
                                      [ User-Name ]
                                      *[ AVP ]
                                      *[ Proxy-Info ]
                                      *[ Route-Record ]
```

```
< ME-Identity-Check-Answer> ::=       < Diameter Header: 324, PXY, 16777252 >
                                      < Session-Id >
                                      [ DRMP ]
                                      [ Vendor-Specific-Application-Id ]
                                      [ Result-Code ]
                                      [ Experimental-Result ]
                                      { Auth-Session-State }
                                      { Origin-Host }
                                      { Origin-Realm }
                                      [ Equipment-Status ]
                                      *[ AVP ]
                                      [ Failed-AVP ]
                                      *[ Proxy-Info ]
                                      *[ Route-Record ]
```

...to implementation!

ts29272_s13.dia

```
   Origin-Realm
   Destination-Host
   Result-Code

@inherits 3gpp

@messages

    ECR ::= < Diameter Header: 324, REQ, PXY >
                < Session-Id >
                { Auth-Session-State }
                { Origin-Host }
                { Origin-Realm }
                { Destination-Realm }
                { Terminal-Information }
                [ DRMP ]
                [ Vendor-Specific-Application-Id ]
                [ Destination-Host ]
                [ User-Name ]
            *   [ AVP ]
            *   [ Proxy-Info ]
            *   [ Route-Record ]

    ECA ::= < Diameter Header: 324, PXY >
                < Session-Id >
                { Auth-Session-State }
                { Origin-Host }
                { Origin-Realm }
                [ DRMP ]
                [ Vendor-Specific-Application-Id ]
                [ Result-Code ]
                [ Experimental-Result ]
                [ Equipment-Status ]
            *   [ AVP ]
            *   [ Failed-AVP ]
            *   [ Proxy-Info ]
            *   [ Route-Record ]
```

rebar3

ts29272_s13.hrl

Luca Dei Zotti, 15 days ago | 1 author (Luca Dei Zotti)

```
%% ------------------------------------------------------
%% This is a generated file.
%% ------------------------------------------------------

-hrl_name('ts29272_s13.hrl').


%%% ------------------------------------------------------
%%% Message records:
%%% ------------------------------------------------------

-record(ts29272_s13_ECR,
    {'Session-Id', 'Auth-Session-State', 'Origin-Host',
     'Origin-Realm', 'Destination-Realm',
     'Terminal-Information', 'DRMP' = [],
     'Vendor-Specific-Application-Id' = [],
     'Destination-Host' = [], 'User-Name' = [], 'AVP' = [],
     'Proxy-Info' = [], 'Route-Record' = []}).

-record(ts29272_s13_ECA,
    {'Session-Id', 'Auth-Session-State', 'Origin-Host',
     'Origin-Realm', 'DRMP' = [],
     'Vendor-Specific-Application-Id' = [],
     'Result-Code' = [], 'Experimental-Result' = [],
     'Equipment-Status' = [], 'AVP' = [], 'Failed-AVP' = [],
     'Proxy-Info' = [], 'Route-Record' = []}).
```

diameter_dict

http://erlang.org/doc/man/diameter_dict.html

“A diameter service, as configured with **diameter:start_service/2**, specifies one or more supported Diameter applications. Each Diameter application specifies a dictionary module that knows how to encode and decode its messages and AVPs. The dictionary module is in turn generated from a file that defines these messages and AVPs. The format of such a file is defined in FILE FORMAT below. Users add support for their specific applications by creating dictionary files, compiling them to Erlang modules using either **diameterc(1)** or **diameter_make(3)** and configuring the resulting dictionaries modules on a service.

Dictionary module generation also results in a hrl file that defines records for the messages and Grouped AVPs defined by the dictionary, these records being what a user of the diameter application sends and receives”

Introducing Erlang to Elixir

```elixir
def project do
  [
    app: :cbl_2019,
    version: "0.1.0",
    elixir: "~> 1.8",
    elixirc_paths: elixirc_paths(Mix.env()),
    erlc_options: [],
    # Let's give our Erlang dependency to the compiler
    erlc_paths: ["deps/minidiadict"],      You, 9 days
    start_permanent: Mix.env() == :prod,
    deps: deps()
  ]
end
```

```elixir
defp extra_deps(env) when env in [:dev] do
  [
    {:minidiadict, git: "https://github.com/zoten/minidiadict", branch: "master"},
  ]
end
```

Crucible of Worlds

```elixir
defmodule Cbl2019.Diameter.S13Records do
  @moduledoc """
  https://hexdocs.pm/elixir/Record.html

  Binding module between Erlang and Elixir records
  "General" Diameter records
  """
  require Record  # we need to use minidiadict Erlang records
  import Record, only: [defrecord: 2, extract: 2]

  defrecord :'ts29272_s13_Experimental-Result', extract(
    :'ts29272_s13_Experimental-Result', from_lib: "minidiadict/include/ts29272_s13.hrl")
  defrecord :ts29272_s13_ECR, extract(
    :ts29272_s13_ECR, from_lib: "minidiadict/include/ts29272_s13.hrl")
  defrecord :ts29272_s13_ECA, extract(
    :ts29272_s13_ECA, from_lib: "minidiadict/include/ts29272_s13.hrl")
  defrecord :'ts29272_s13_Vendor-Specific-Application-Id', extract(
    :'ts29272_s13_Vendor-Specific-Application-Id', from_lib: "minidiadict/include/ts29272_s13.hrl")

  defrecord :'ts29272_s13_Terminal-Information', extract(
    :'ts29272_s13_Terminal-Information', from_lib: "minidiadict/include/ts29272_s13.hrl")
  # defrecord :'3gpp_Terminal-Information', extract(
  #   :'3gpp_Terminal-Information', from_lib: "minidiadict/include/3gpp.hrl")

  # Macros      You, 7 days ago • Initial commit
  # Here is the only "real" problem: I didn't find a nice way to import
  # Erlang's macros in Elixir :(
  # A lot of compiled constants from .dia files to .hrl headers are defined as macro
  defmacro ts29272_s13_equipment_status_whitelisted do
    quote do: 0
  end
  defmacro ts29272_s13_equipment_status_blacklisted do
    quote do: 1
  end
  defmacro ts29272_s13_equipment_status_greylisted do
    quote do: 2
  end

end
```

Callbacks

- **start_service/2**
- **stop_service/1**
- **subscribe**/1

```
[{:string_decode, false},
{:'Origin-Host', originhost},
{:'Origin-Realm', originrealm},
{:'Origin-State-Id', :diameter.origin_state_id()},
# should be the vendor id registered at IETF
{:'Vendor-Id', vendor_id(:'3gpp')},
{:'Product-Name', "EIR" <> Integer.to_string(idx)},
{:'Vendor-Specific-Application-Id',
    [vendor_spec_app_id(
        :auth, :ts29272_s13.vendor_id, [:ts29272_s13.id])]},
{:'Supported-Vendor-Id', [vendor_id(:'3gpp'), vendor_id(:etsi)]},
{:application, s13_application()}]
```

```
# http://erlang.org/doc/ No documentation available ribe-1
# It is not an error to subscribe to events from a service that d
# yet exist. Doing so before adding transports is required to gua
# the reception of all transport-related events.
:diameter.subscribe(name)       You, 13 days ago • Initial commit
:diameter.stop_service(name)

{:ok, trans} = Store.get(:diameter_transports) |> Enum.fetch(idx)
{:ok, servs} = Store.get(:diameter_services) |> Enum.fetch(idx)

:ok = :diameter.start_service(name, service(servs, idx))
```

Callbacks to start or stop a service, specifying the capabilities to be advertised during capabilities exchange or triggering DPR requests.

Subscribing to a service (name) means receiving all service events from it.

A service defines basic properties of the diameter node we are implementing.

Callbacks (2)

```
# Diameter Callbacks
@doc """
http://erlang.org/doc/man/diameter_app.html#Mod:peer_up-3

Note 1
There is no requirement that a callback return before incoming
requests are received

Note 2
A watchdog state machine can reach state OKAY from state
SUSPECT without a new capabilities exchange taking place.
A new transport connection (and capabilities exchange)
results in a new peer_ref().
"""
def peer_up(svcName, peer, state) do
  Logger.debug "Peer up #{inspect svcName} #{inspect peer}"
  state
end

@doc """
http://erlang.org/doc/man/diameter_app.html#Mod:peer_down-3
"""
def peer_down(svcName, peer, state) do
    Logger.debug "Peer down #{inspect svcName} #{inspect peer}"
    state
end
```

- **peer_up/3**
- **peer_down/3**
- **pick_peer/4**

```
@doc """
http://erlang.org/doc/man/diameter_app.html#Mod:pick_peer-4

Invoked as a consequence of a call to diameter:call/4 to select
a destination peer for an outgoing request. The return value
indicates the selected peer.
The candidate lists contain only those peers that have advertised
support for the Diameter application in question during
capabilities exchange, that have not be excluded by a filter
option in the call to diameter:call/4 and whose watchdog state
machine is in the OKAY state.
"""
def pick_peer(localCandidates, _remoteCandidates, _svcName, _state) do
  Logger.debug fn -> "pick_peer" end
  [peer | _] = localCandidates
  Logger.debug fn -> "Picked peer: #{inspect peer}" end
  {:ok, peer}
end
```

Control actions to perform when a peer comes up or down, or to pick the destination peer for a **diameter_call** request. The stack automatically performs selection on supported advertised applications from known peers.

Callbacks (3)

```
@doc """
http://erlang.org/doc/man/diameter_app.html#Mod:handle_request-3

Invoked when a request message is received from a peer. The application
in which the callback takes place (that is, the callback module as
configured with diameter:start_service/2) is determined by the Application
Identifier in the header of the incoming request message, the selected
module being the one whose corresponding dictionary declares itself as
defining either the application in question or the Relay application.
"""
def handle_request(packet, svcName, peer) do
  # #diameter_packet{header = #diameter_header{},
  #     avps    = [#diameter_avp{}],
  #     msg     = record() | undefined,
  #     errors = [Unsigned32() | {Unsigned32(), #diameter_avp{}}],
  #     bin     = binary(),
  #     transport_data = term()}
  Logger.info "Arrived Diameter #{msgname(packet)} message"
  action = case msgname(packet) do
      :'ECR' -> s13_handle_ecr(packet, svcName, peer)
  end
  Logger.info "Gonna send #{inspect action}"
  action
end
```

Callbacks (4)

```
# Let's build the real ECA
# < ME-Identity-Check-Answer> ::= < Diameter Header: 324, PXY, 16777252 >
#     < Session-Id >
#     [ DRMP ]
#     [ Vendor-Specific-Application-Id ]
#     [ Result-Code ]
#     [ Experimental-Result ]
#     { Auth-Session-State }
#     { Origin-Host }
#     { Origin-Realm }
#     [ Equipment-Status ]
#     *[ AVP ]      You, 13 days ago • Initial commit
#     [ Failed-AVP ]
#     *[ Proxy-Info ]
#     *[ Route-Record ]
eca = S13Records.ts29272_s13_ECA(
  'Origin-Host': ohost,
  'Origin-Realm': orealm,
  'Session-Id': S13Records.ts29272_s13_ECR(ecr, :'Session-Id'),
  'Auth-Session-State': S13Records.ts29272_s13_ECR(ecr, :'Auth-Session-State'),
  'Result-Code': result,
  'Experimental-Result': expResult,
  'Equipment-Status': equipment_status,
  # 'Supported-Features': [avp_supported_feature()],
  'Proxy-Info': S13Records.ts29272_s13_ECR(ecr, :'Proxy-Info'))
Logger.debug "Msg: #{inspect eca}"
{:reply, eca}
```

...and on, and on

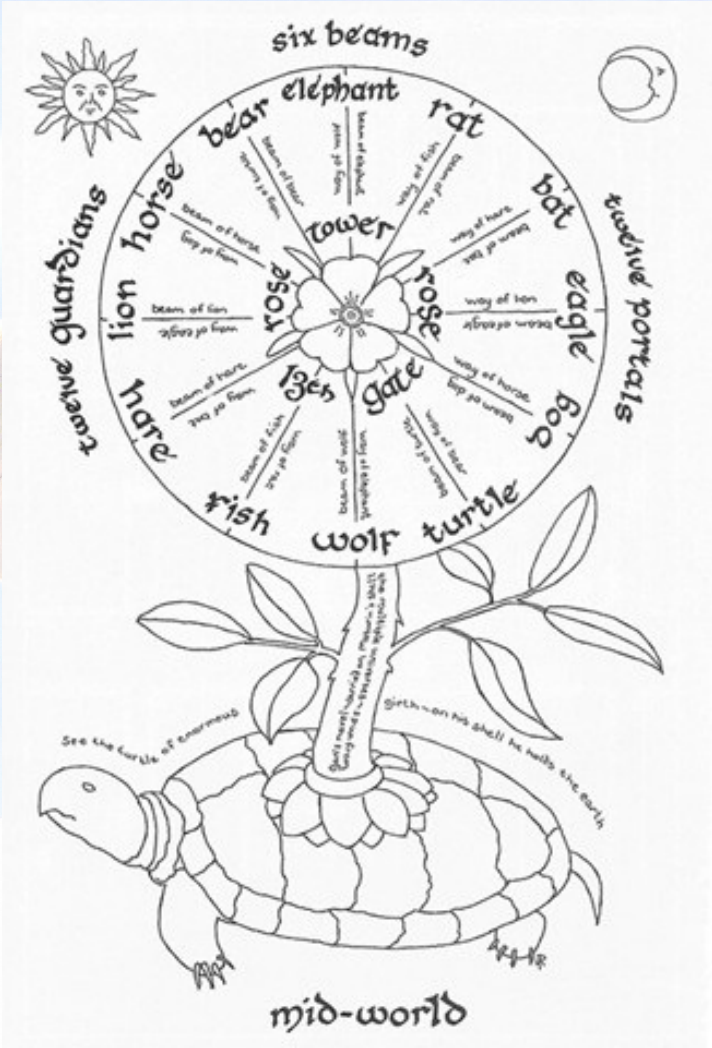
Other hooks:

- **prepare_request/3:** invoked to return a request for encoding and transport. Allows the sender to use the selected peer's capabilities to modify the outgoing request (see the client file *mme.ex*)
- **prepare_retransmit/3:** like **prepare_request/3** but for lost peer connection and alternate peer selected
- **handle_answer/4 - handle_error/4:** handlers for answer messages from peers, or to manage errors before answer

...and helpers

- **:diameter_codec.encode/2 - :diameter_codec.decode/2, session_id/1, origin_state_id/0, service_info/2 …**
- Not a lot of handy debugging information (“why my message/AVP is not sent/encoded?”): you have often to dig for solutions!

...let's follow the Beam!



Courtesy of https://stephenking.fandom.com/wiki/File:Turtle-2-.jpg

References

- https://en.wikipedia.org/wiki/Diameter_(protocol) (of course)
- http://erlang.org/doc/man/diameter.html (of super-course)
- https://github.com/erlang/otp/tree/master/lib/diameter (when I grow up I'll be ***this*** good)
- https://www.ibm.com/developerworks/library/wi-diameter/index.html
- Comparison between RADIUS and Diameter – Anna Hosia, Helsinki University of Technology – 2003
- Holger Winkelmann and Ulf Wiger -The Erlang/OTP DIAMETER Stack (https://vimeo.com/26034259)
- https://diameter-protocol.blogspot.com

Thank you!

- Questions?
- Comments?
- All code available on github.com/zoten