Implementing UDP protocols in Elixir

ANDREI C – 0X7F.DEV

whoami

- Developer for 11 years, last 3 doing Elixir (still learning)
- Licensed accountant building my own startup in Phoenix Liveview
- Vicepresident at Croatian association for open systems and internet
- Member and co-organiser of DORS/CLUC conference

0x7f HROPEN DORS/CLUC FOSDEM

The plan

1. The problem we're solving
2. Discovering a protocol of our choice
3. Creating a simple UDP server in Elixir
4. Implementing the protocol
5. Extra: custom sigils

The problem - NTP protocol

- I wanted to fake uptime
  - My naive thinking was that I can mess with the clock and fake it that way
  - You can not
  - Next year on FOSDEM: *Implementing kernel modules in Elixir*
- I kind of like solving this problem (see implementing a DNS server in JS)
- It's a cool topic to write about and talk about

Discovering the protocol

What's NTP?

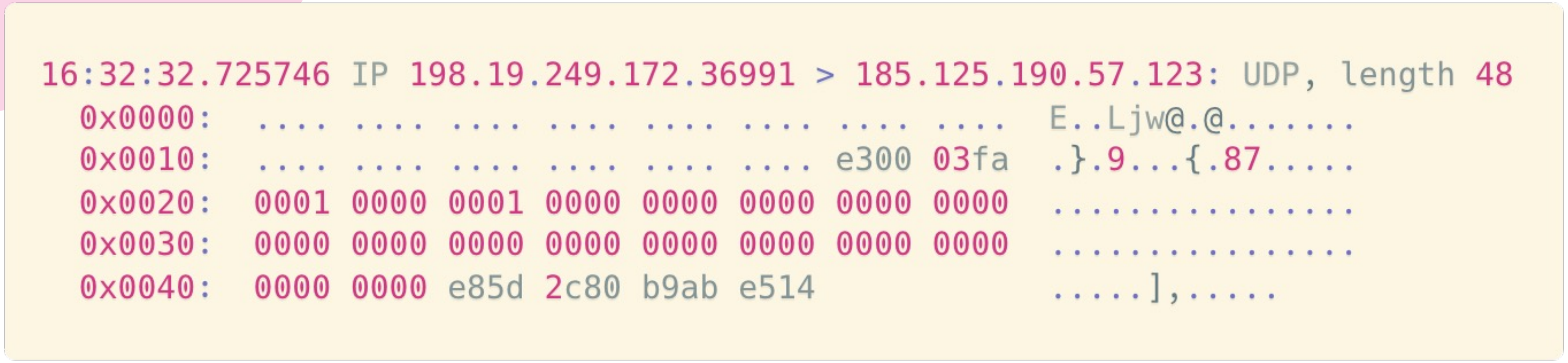
- a way to sync your hardware clock over the internet
- a terrible protocol that has been used for DDoS attacks and has expoilts left and right
- one of the easiest protocols you can implement

Gathering data

```
$ sudo apt install tcpdump ntpdate
$ # Next start our network package capture in background
$ sudo tcpdump udp -w output.pcap &
[1] 2272
$ # And now we update the time
$ sudo ntpdate -u ntp.ubuntu.com
15 Jul 16:31:21 ntpdate[2273]: adjust time server 185.125.190.58 offset +0.113190 sec
```

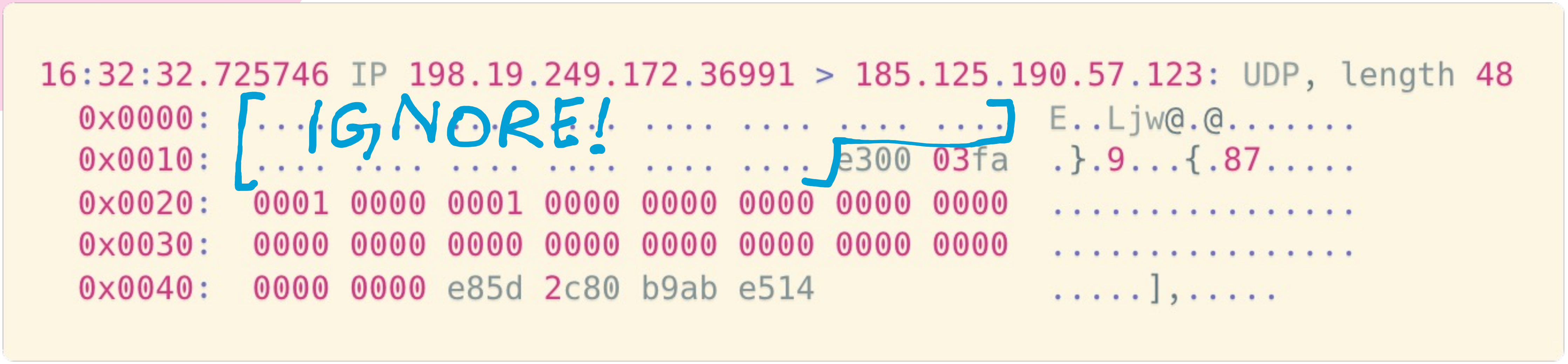
- now we captured a few UDP packets so we can see how they look like
- hopefully we can just recreate them in Elixir and call it a day

Exploring the **.pcap** file



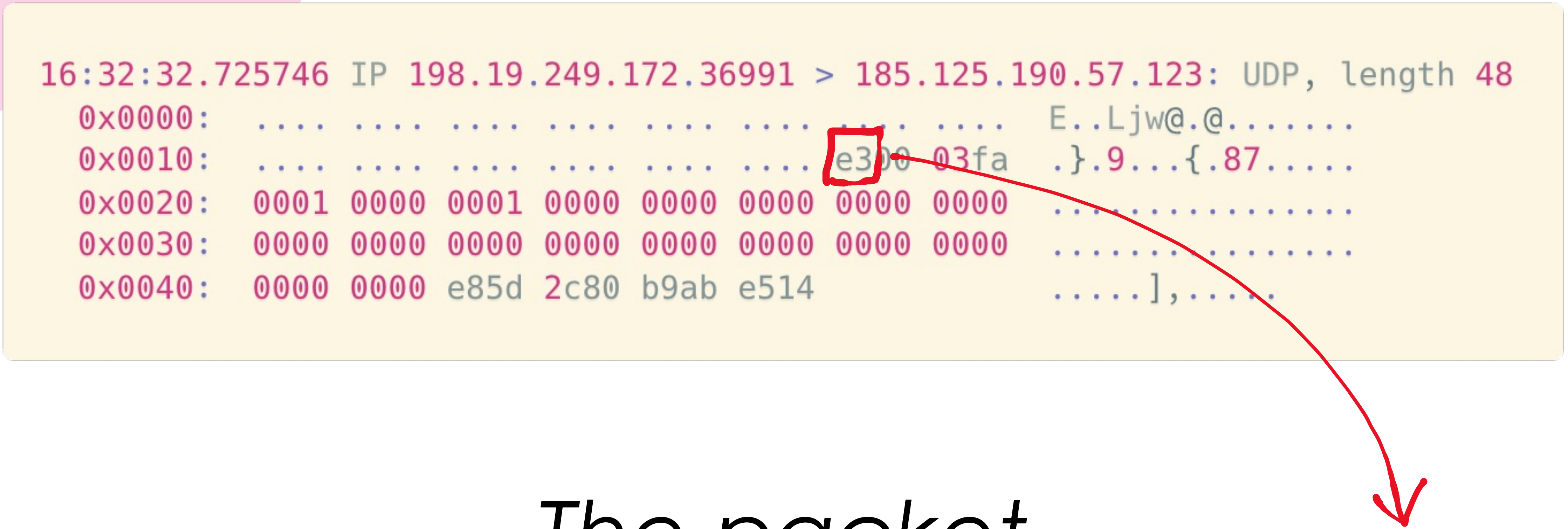
```
16:32:32.725746 IP 198.19.249.172.36991 > 185.125.190.57.123: UDP, length 48
  0x0000:  .... .... .... .... .... .... .... ....  E..Ljw@.@.......
  0x0010:  .... .... .... .... .... .... e300 03fa  .}.9...{.87.....
  0x0020:  0001 0000 0001 0000 0000 0000 0000 0000  ................
  0x0030:  0000 0000 0000 0000 0000 0000 0000 0000  ................
  0x0040:  0000 0000 e85d 2c80 b9ab e514            .....],.....
```

The packet



```
16:32:32.725746 IP 198.19.249.172.36991 > 185.125.190.57.123: UDP, length 48
   0x0000:  .... .... .... .... .... .... .... ....  E..Ljw@.@.......
   0x0010:  .... .... .... .... .... .... e300 03fa  .}.9...{.87.....
   0x0020:  0001 0000 0001 0000 0000 0000 0000 0000  ................
   0x0030:  0000 0000 0000 0000 0000 0000 0000 0000  ................
   0x0040:  0000 0000 e85d 2c80 b9ab e514            .....],.....
```

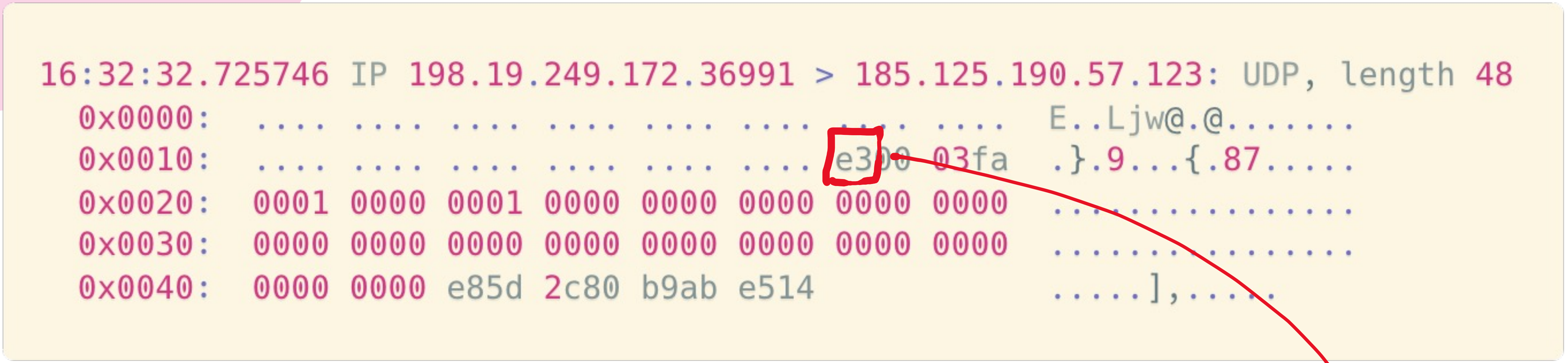
The packet



```
16:32:32.725746 IP 198.19.249.172.36991 > 185.125.190.57.123: UDP, length 48
    0x0000:  .... .... .... .... .... .... .... ....  E..Ljw@.@.......
    0x0010:  .... .... .... .... .... .... e300 03fa  .}.9...{.87.....
    0x0020:  0001 0000 0001 0000 0000 0000 0000 0000  ................
    0x0030:  0000 0000 0000 0000 0000 0000 0000 0000  ................
    0x0040:  0000 0000 e85d 2c80 b9ab e514            .....],.....
```

The packet

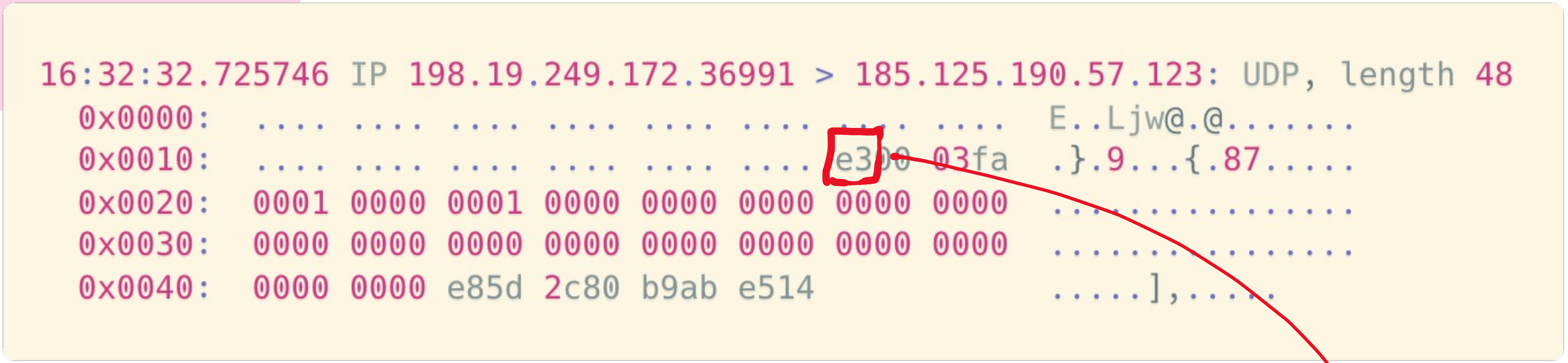
Binary: 11100011



```
16:32:32.725746 IP 198.19.249.172.36991 > 185.125.190.57.123: UDP, length 48
  0x0000:  .... .... .... .... .... .... .... ....  E..Ljw@.@.......
  0x0010:  .... .... .... .... .... .... e300 03fa  .}.9...{.87.....
  0x0020:  0001 0000 0001 0000 0000 0000 0000 0000  ................
  0x0030:  0000 0000 0000 0000 0000 0000 0000 0000  ................
  0x0040:  0000 0000 e85d 2c80 b9ab e514            .....],.....
```

The packet

Binary: 11100011

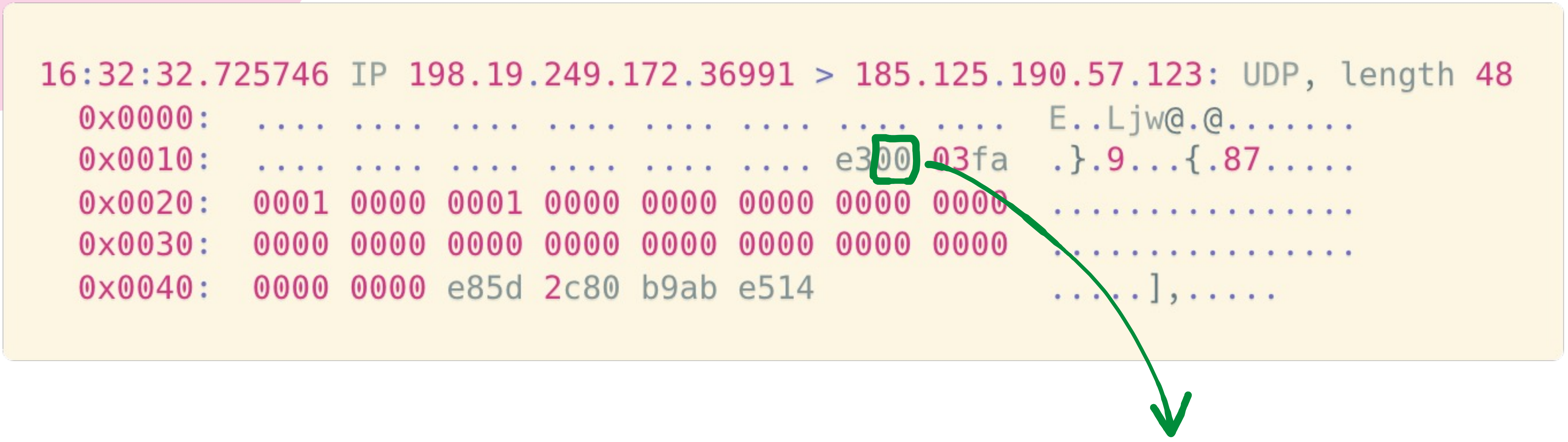


```
16:32:32.725746 IP 198.19.249.172.36991 > 185.125.190.57.123: UDP, length 48
    0x0000:  .... .... .... .... .... .... .... ....  E..Ljw@.@.......
    0x0010:  .... .... .... .... .... .... e300 03fa  .}.9...{.87.....
    0x0020:  0001 0000 0001 0000 0000 0000 0000 0000  ................
    0x0030:  0000 0000 0000 0000 0000 0000 0000 0000  ................
    0x0040:  0000 0000 e85d 2c80 b9ab e514            .....],.....
```

The packet

Binary: 11100011

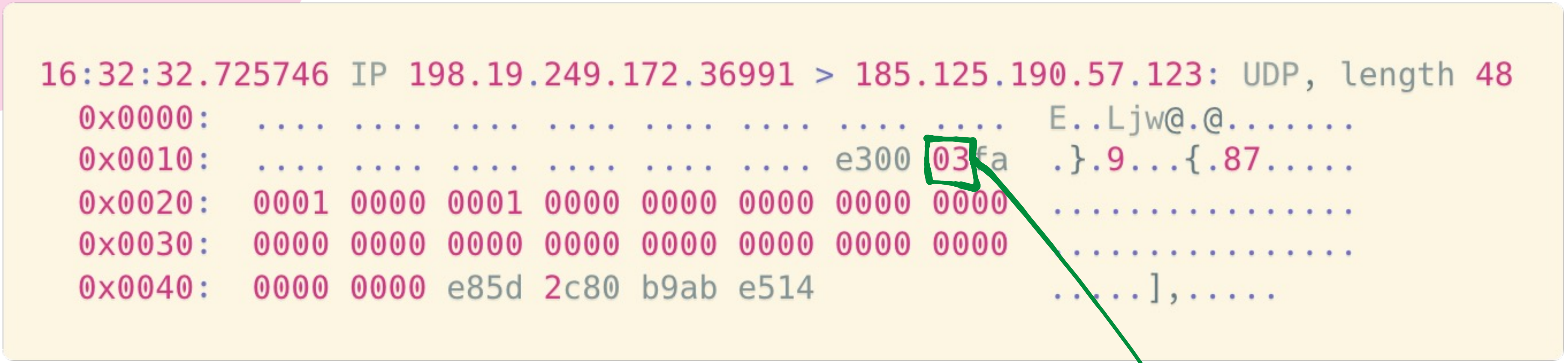
- Leap year indicator
- NPT version
- Packet mode (client)



```
16:32:32.725746 IP 198.19.249.172.36991 > 185.125.190.57.123: UDP, length 48
  0x0000:  .... .... .... .... .... .... .... ....  E..Ljw@.@.......
  0x0010:  .... .... .... .... .... .... e300 03fa  .}.9...{.87.....
  0x0020:  0001 0000 0001 0000 0000 0000 0000 0000  ................
  0x0030:  0000 0000 0000 0000 0000 0000 0000 0000  ................
  0x0040:  0000 0000 e85d 2c80 b9ab e514            .....],.....
```

Clock stratum

The packet



```
16:32:32.725746 IP 198.19.249.172.36991 > 185.125.190.57.123: UDP, length 48
  0x0000:  .... .... .... .... .... .... .... ....  E..Ljw@.@.......
  0x0010:  .... .... .... .... .... .... e300 03fa  .}.9...{.87.....
  0x0020:  0001 0000 0001 0000 0000 0000 0000 0000  ................
  0x0030:  0000 0000 0000 0000 0000 0000 0000 0000  ................
  0x0040:  0000 0000 e85d 2c80 b9ab e514            .....],.....
```

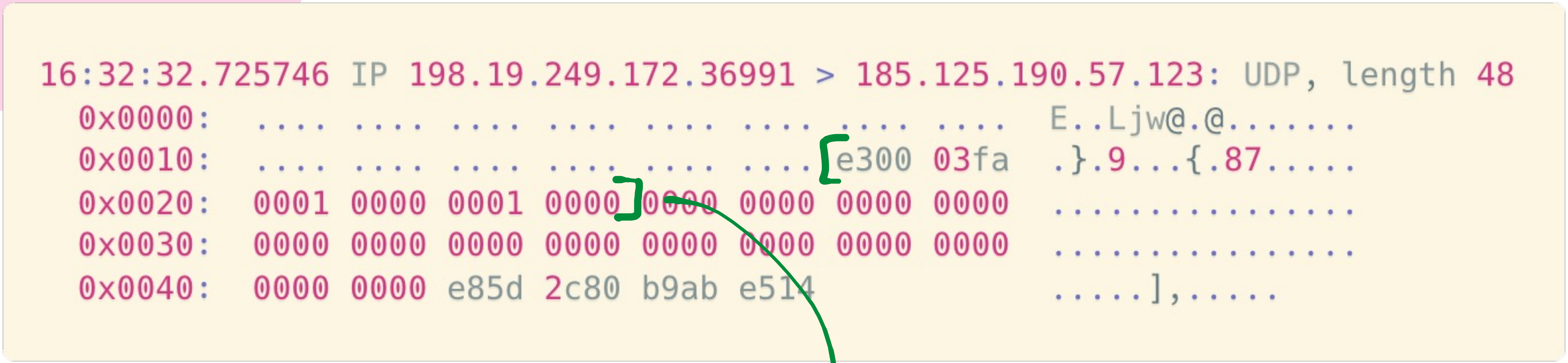
The packet

Pooling interval

```
16:32:32.725746 IP 198.19.249.172.36991 > 185.125.190.57.123: UDP, length 48
    0x0000:  .... .... .... .... .... .... .... ....  E..Ljw@.@.......
    0x0010:  .... .... .... .... .... .... e300 03[fa  .}.9...{.87.....
    0x0020:  0001 0000 0001 0000] 0000 0000 0000 0000  ................
    0x0030:  0000 0000 0000 0000 0000 0000 0000 0000  ................
    0x0040:  0000 0000 e85d 2c80 b9ab e514            .....],.....
```

The packet

Clock precision, delay, dispersion

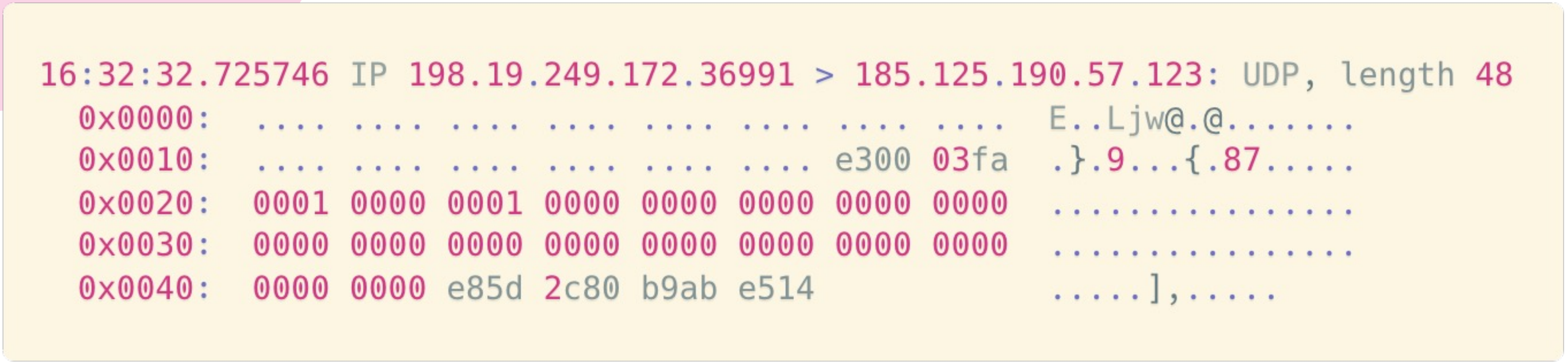


```
16:32:32.725746 IP 198.19.249.172.36991 > 185.125.190.57.123: UDP, length 48
  0x0000:  .... .... .... .... .... .... .... ....  E..Ljw@.@.......
  0x0010:  .... .... .... .... .... .... e300 03fa  .}.9...{.87.....
  0x0020:  0001 0000 0001 0000 0000 0000 0000 0000  ................
  0x0030:  0000 0000 0000 0000 0000 0000 0000 0000  ................
  0x0040:  0000 0000 e85d 2c80 b9ab e514            .....],.....
```

The packet

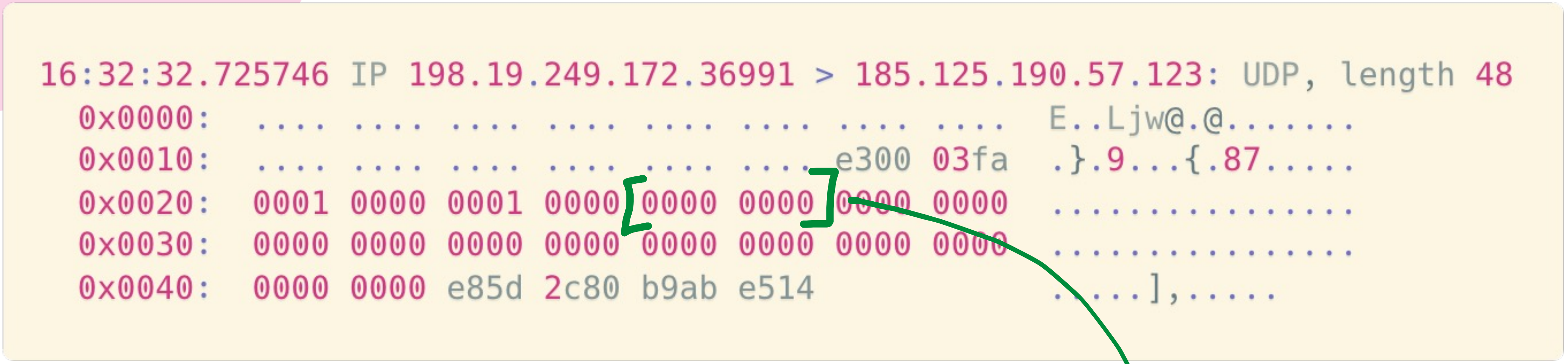
https://0x7f.dev/post/ntp-implementation-in-elixir/#fn:4

0x7f HROPEN DORS/CLUC FOSDEM



```
16:32:32.725746 IP 198.19.249.172.36991 > 185.125.190.57.123: UDP, length 48
  0x0000:  .... .... .... .... .... .... .... ....  E..Ljw@.@.......
  0x0010:  .... .... .... .... .... .... e300 03fa  .}.9...{.87.....
  0x0020:  0001 0000 0001 0000 0000 0000 0000 0000  ................
  0x0030:  0000 0000 0000 0000 0000 0000 0000 0000  ................
  0x0040:  0000 0000 e85d 2c80 b9ab e514            .....],.....
```

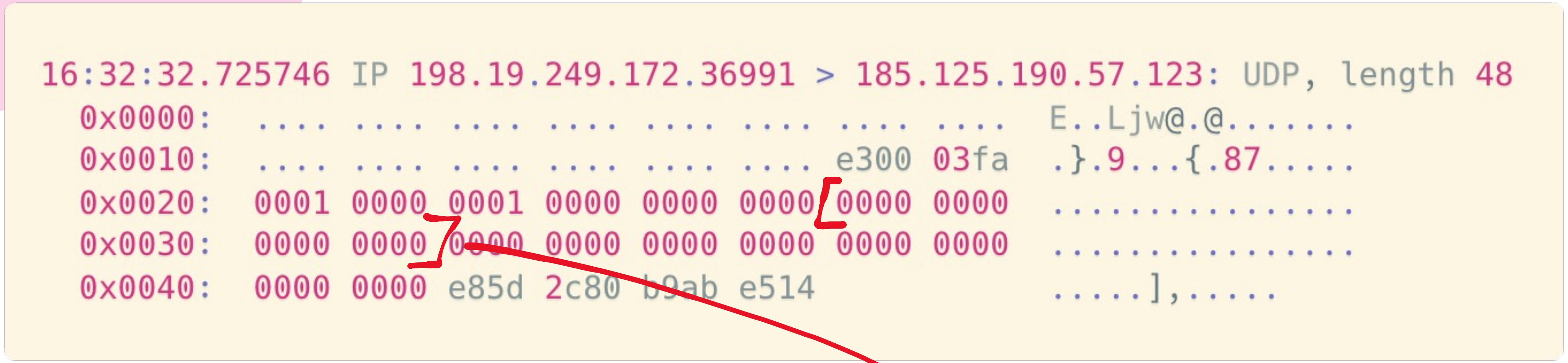
The packet



```
16:32:32.725746 IP 198.19.249.172.36991 > 185.125.190.57.123: UDP, length 48
   0x0000:  .... .... .... .... .... .... .... ....  E..Ljw@.@.......
   0x0010:  .... .... .... .... .... .... e300 03fa  .}.9...{.87.....
   0x0020:  0001 0000 0001 0000 [0000 0000] 0000 0000  ................
   0x0030:  0000 0000 0000 0000 0000 0000 0000 0000  ................
   0x0040:  0000 0000 e85d 2c80 b9ab e514            .....],.....
```

The packet

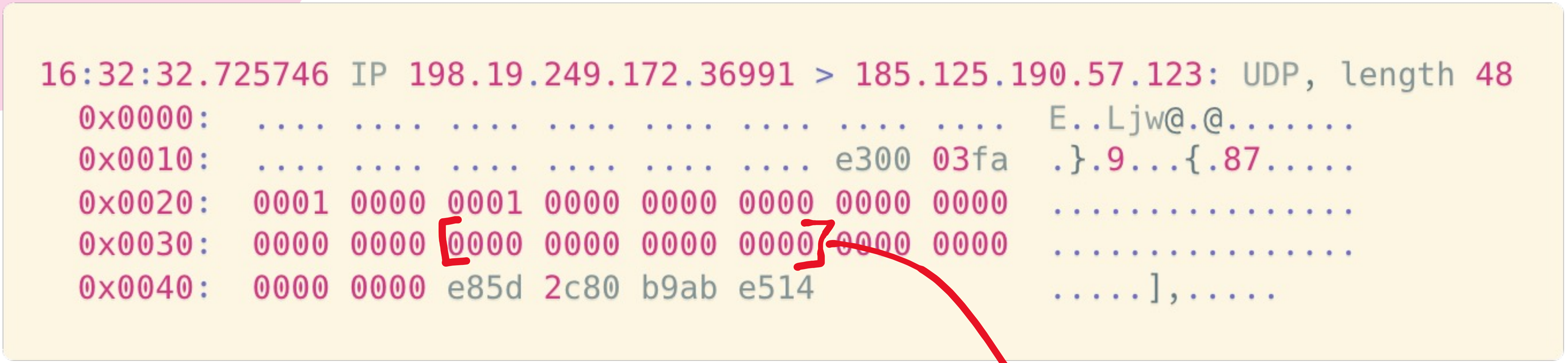
Reference ID (store this)



```
16:32:32.725746 IP 198.19.249.172.36991 > 185.125.190.57.123: UDP, length 48
    0x0000:  .... .... .... .... .... .... .... ....  E..Ljw@.@.......
    0x0010:  .... .... .... .... .... .... e300 03fa  .}.9...{.87.....
    0x0020:  0001 0000 0001 0000 0000 0000 0000 0000  ................
    0x0030:  0000 0000 0000 0000 0000 0000 0000 0000  ................
    0x0040:  0000 0000 e85d 2c80 b9ab e514            .....],.....
```

Ref. timestamp

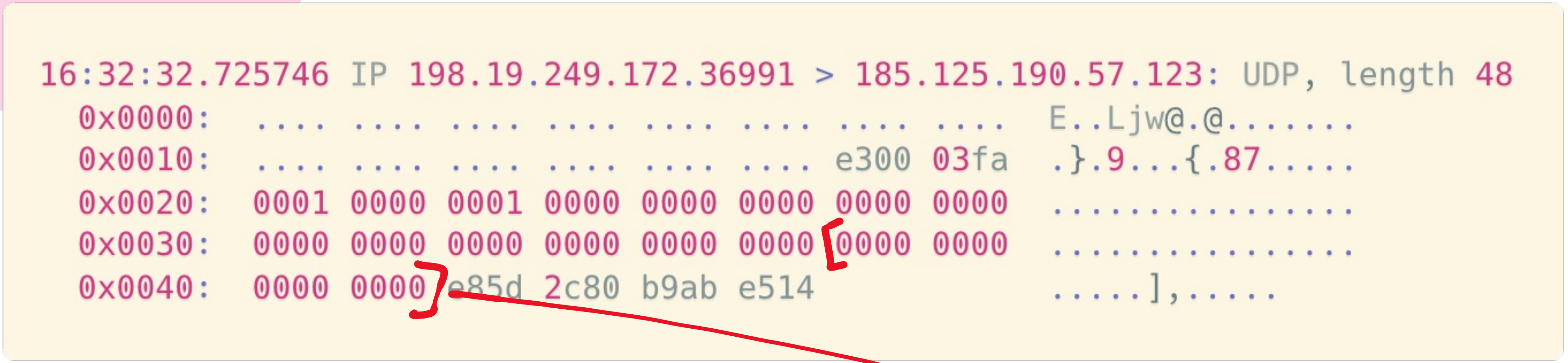
The packet



```
16:32:32.725746 IP 198.19.249.172.36991 > 185.125.190.57.123: UDP, length 48
    0x0000:  .... .... .... .... .... .... .... ....  E..Ljw@.@.......
    0x0010:  .... .... .... .... .... .... e300 03fa  .}.9...{.87.....
    0x0020:  0001 0000 0001 0000 0000 0000 0000 0000  ................
    0x0030:  0000 0000 [0000 0000 0000 0000] 0000 0000  ................
    0x0040:  0000 0000 e85d 2c80 b9ab e514            .....],.....
```

Origin timestamp

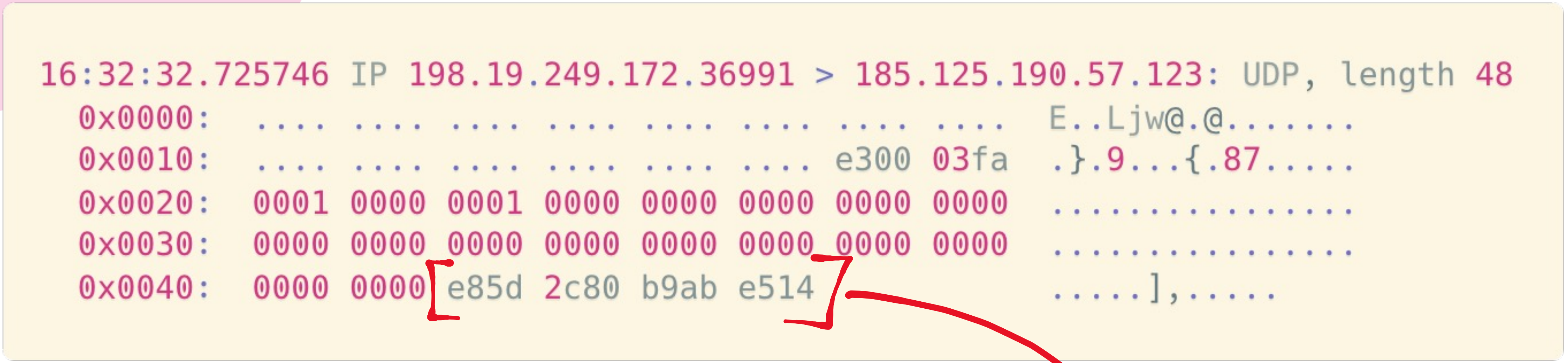
The packet



```
16:32:32.725746 IP 198.19.249.172.36991 > 185.125.190.57.123: UDP, length 48
   0x0000:   .... .... .... .... .... .... .... ....  E..Ljw@.@.......
   0x0010:   .... .... .... .... .... .... e300 03fa  .}.9...{.87.....
   0x0020:   0001 0000 0001 0000 0000 0000 0000 0000  ................
   0x0030:   0000 0000 0000 0000 0000 0000 [0000 0000  ................
   0x0040:   0000 0000] e85d 2c80 b9ab e514           .....],.....
```

The packet

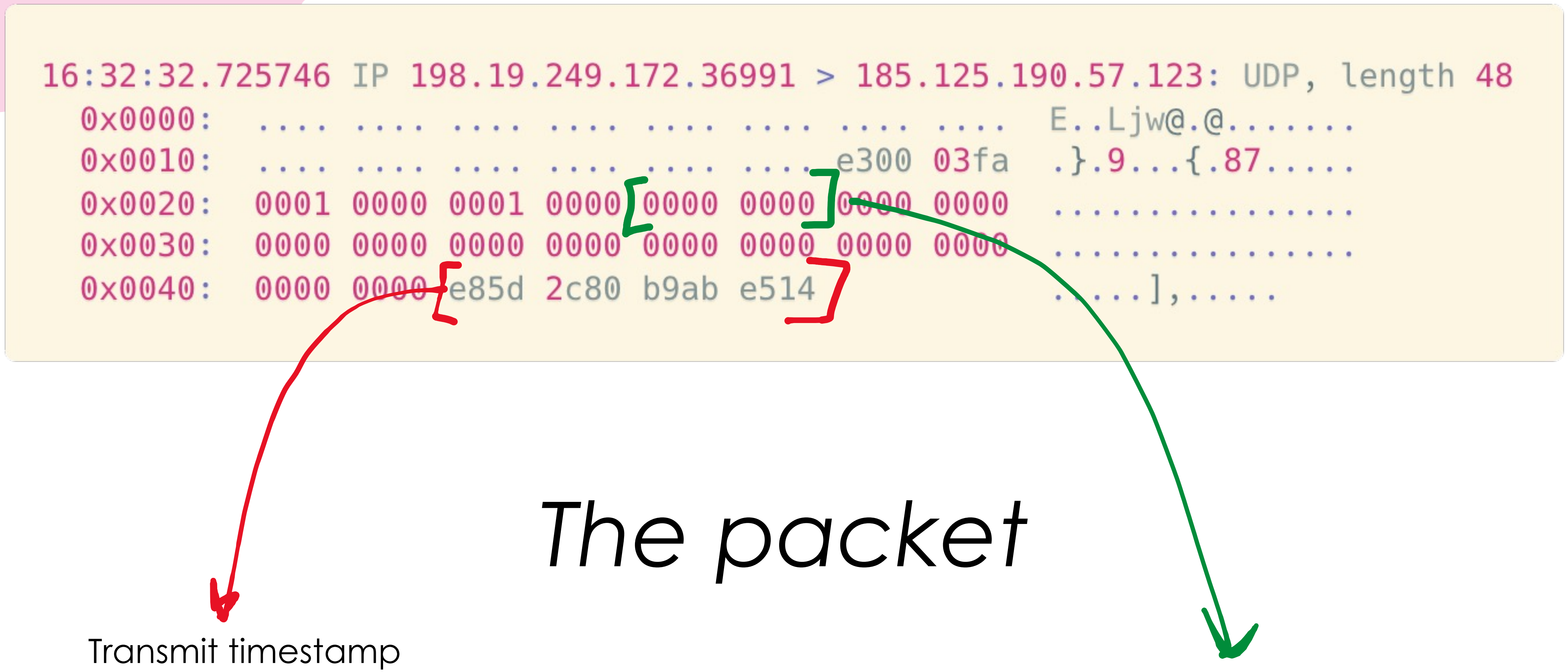
Receive timestamp



```
16:32:32.725746 IP 198.19.249.172.36991 > 185.125.190.57.123: UDP, length 48
  0x0000:  .... .... .... .... .... .... .... ....  E..Ljw@.@.......
  0x0010:  .... .... .... .... .... .... e300 03fa  .}.9...{.87.....
  0x0020:  0001 0000 0001 0000 0000 0000 0000 0000  ................
  0x0030:  0000 0000 0000 0000 0000 0000 0000 0000  ................
  0x0040:  0000 0000 e85d 2c80 b9ab e514            .....],.....
```

Transmit timestamp
Return as it!

The packet



```
16:32:32.725746 IP 198.19.249.172.36991 > 185.125.190.57.123: UDP, length 48
	0x0000:  .... .... .... .... .... .... .... ....  E..Ljw@.@.......
	0x0010:  .... .... .... .... .... .... e300 03fa  .}.9...{.87.....
	0x0020:  0001 0000 0001 0000 0000 0000 0000 0000  ................
	0x0030:  0000 0000 0000 0000 0000 0000 0000 0000  ................
	0x0040:  0000 0000 e85d 2c80 b9ab e514            .....],.....
```

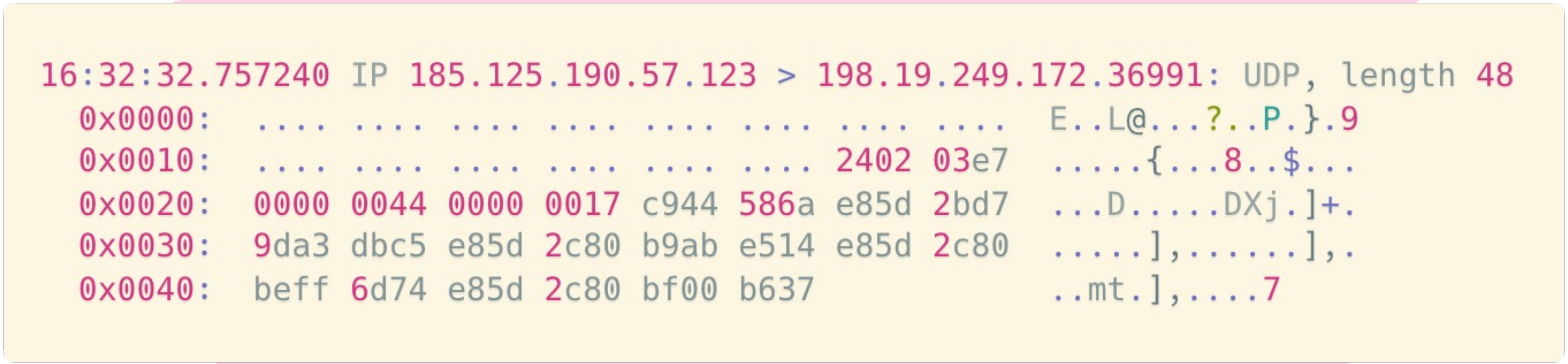
The packet

Transmit timestamp
Return as it!

Reference ID (store this)

Data we “actually” need from the **request**

```
<<_::binary-size(12),
  id::binary-size(4),
  _::binary-size(24),
  origin_timestamp::binary-size(8)>> = request
```



```
16:32:32.757240 IP 185.125.190.57.123 > 198.19.249.172.36991: UDP, length 48
  0x0000:  .... .... .... .... .... .... .... ....  E..L@...?..P.}.9
  0x0010:  .... .... .... .... .... .... 2402 03e7  .....{...8..$...
  0x0020:  0000 0044 0000 0017 c944 586a e85d 2bd7  ...D.....DXj.]+.
  0x0030:  9da3 dbc5 e85d 2c80 b9ab e514 e85d 2c80  .....],......],.
  0x0040:  beff 6d74 e85d 2c80 bf00 b637            ..mt.],....7
```

We set:

Reference, Origin, and Receive timestamps to "now"

Creating a simple **UDP server in Elixir**

Running on **port 123**

```elixir
{:ok, socket} = :gen_udp.open(port, [:binary, {:active, false}])
```

```elixir
{:ok, socket} = :gen_udp.open(port, [:binary, {:active, false}])

case :gen_udp.recv(socket, 0) do
  {:ok, {ip, port, data}} ->
    :gen_udp.send(socket, ip, port, "Hello, world!")

  {:ok, :udp_closed} ->
    # do something

  {:error, reason} ->
    # do something
end
```

Receives ONE packet

```
defmodule SimpleServer do







end
```

```elixir
defmodule SimpleServer do
  def init(port) do
    {:ok, socket} = :gen_udp.open(port, [:binary, {:active, false}])


  end











end
```

```elixir
defmodule SimpleServer do
  def init(port) do
    {:ok, socket} = :gen_udp.open(port, [:binary, {:active, false}])

    loop(socket)
  end

  def loop(socket) do
    case :gen_udp.recv(socket, 0) do
      {:ok, :udp_closed} ->
        # do something

      {:error, reason} ->
        # do something

      {:ok, {ip, port, data}} ->
        :gen_udp.send(socket, ip, port, "Hello, world!")
    end
  end
end
```

```elixir
defmodule SimpleServer do
  def init(port) do
    {:ok, socket} = :gen_udp.open(port, [:binary, {:active, false}])

    loop(socket)
  end

  def loop(socket) do
    case :gen_udp.recv(socket, 0) do
      {:ok, :udp_closed} ->
        # do something

      {:error, reason} ->
        # do something

      {:ok, {ip, port, data}} ->
        :gen_udp.send(socket, ip, port, "Hello, world!")
    end
  end  ← LOOP :)
end

SimpleServer.init(123)
```

GEN ERVER

```elixir
defmodule UdpServer do
  use GenServer

  def init(_params) do



  end

  def handle_continue(:loop, socket) do










    end
  end
end
```

```elixir
defmodule UdpServer do
  use GenServer

  def init(_params) do
    {:ok, socket} = :gen_udp.open(123, [:binary, {:active, false}])

    {:ok, socket, {:continue, :loop}}
  end

  def handle_continue(:loop, socket) do









  end
end
```

```elixir
defmodule UdpServer do
  use GenServer

  def init(_params) do
    {:ok, socket} = :gen_udp.open(123, [:binary, {:active, false}])

    {:ok, socket, {:continue, :loop}}
  end

  def handle_continue(:loop, socket) do
    case :gen_udp.recv(socket, 0) do
      {:ok, :udp_closed} ->
        # TODO: implement

      {:error, reason} ->
        # TODO: implement

      {:ok, {ip, port, data}} ->
        :gen_udp.send(socket, ip, port, "Hello, world!")

    end
  end
end
```

```elixir
defmodule UdpServer do
  use GenServer

  def init(_params) do
    {:ok, socket} = :gen_udp.open(123, [:binary, {:active, false}])

    {:ok, socket, {:continue, :loop}}
  end

  def handle_continue(:loop, socket) do
    case :gen_udp.recv(socket, 0) do
      {:ok, :udp_closed} ->
        # TODO: implement

      {:error, reason} ->
        # TODO: implement

      {:ok, {ip, port, data}} ->
        :gen_udp.send(socket, ip, port, "Hello, world!")
        {:noreply, socket, {:continue, :loop}}
    end
  end
end
```

```elixir
defmodule UdpServer do
  use GenServer

  def init(_params) do
    {:ok, socket} = :gen_udp.open(123, [:binary, {:active, false}])

    {:ok, socket, {:continue, :loop}}
  end
```

```elixir
def start_link(params) do
  GenServer.start_link(__MODULE__, params, name: __MODULE__)
end
```

```elixir
      {:error, reason} ->
        # TODO: implement

      {:ok, {ip, port, data}} ->
        :gen_udp.send(socket, ip, port, "Hello, world!")
        {:noreply, socket, {:continue, :loop}}
    end
  end
end
```

```elixir
defmodule Application do
  use Application

  def start(_type, _args) do
    children = [
      UdpServer
    ]

    opts = [strategy: :one_for_one, name: NtpServer.Supervisor]
    Supervisor.start_link(children, opts)
  end
end
```

Implementing the protocol

```elixir
def generate_ntp_response(<<_::binary-size(40), origin_timestamp::binary>> = _request) do







end
```

```
def generate_ntp_response(<<_::binary-size(40), origin_timestamp::binary>> = _request) do
  now = System.system_time(:second)

  receive_timestamp = now
  transmit_timestamp = receive_timestamp










end
```

```elixir
def generate_ntp_response(<<_::binary-size(40), origin_timestamp::binary>> = _request) do
  now = System.system_time(:second)

  receive_timestamp = now
  transmit_timestamp = receive_timestamp

  header = ~b(24 02 03 E7) <> <<0::size(64)>>
  id = ~b(56 17 C3 1E)



end
```

```elixir
@ntp_constant 2_208_988_800

def generate_ntp_response(<<_::binary-size(40), origin_timestamp::binary>> = _request) do
  now = System.system_time(:second)

  receive_timestamp = now
  transmit_timestamp = receive_timestamp

  header = ~b(24 02 03 E7) <> <<0::size(64)>>
  id = ~b(56 17 C3 1E)
  reference_timestamp = <<receive_timestamp + @ntp_constant::size(32), 0::size(32)>>
  origin_timestamp = origin_timestamp
  receive_timestamp = <<receive_timestamp + @ntp_constant::size(32), 0::size(32)>>
  transmit_timestamp = <<transmit_timestamp + @ntp_constant::size(32), 0::size(32)>>



end
```

```
@ntp_constant 2_208_988_800

def generate_ntp_response(<<_::binary-size(40), origin_timestamp::binary>> = _request) do
  now = System.system_time(:second)

  receive_timestamp = now
  transmit_timestamp = receive_timestamp

  header = ~b(24 02 03 E7) <> <<0::size(64)>>
  id = ~b(56 17 C3 1E)
  reference_timestamp = <<receive_timestamp + @ntp_constant::size(32), 0::size(32)>>
  origin_timestamp = origin_timestamp
  receive_timestamp = <<receive_timestamp + @ntp_constant::size(32), 0::size(32)>>
  transmit_timestamp = <<transmit_timestamp + @ntp_constant::size(32), 0::size(32)>>



end
```

To convert NTP timestamps to Unix timestamps (or vice versa), the offset of 2_208_988_800 is added or subtracted, depending on the direction of the conversion.

```elixir
@ntp_constant 2_208_988_800

def generate_ntp_response(<<_::binary-size(40), origin_timestamp::binary>> = _request) do
  now = System.system_time(:second)

  receive_timestamp = now
  transmit_timestamp = receive_timestamp

  header = ~b(24 02 03 E7) <> <<0::size(64)>>
  id = ~b(56 17 C3 1E)
  reference_timestamp = <<receive_timestamp + @ntp_constant::size(32), 0::size(32)>>
  origin_timestamp = origin_timestamp
  receive_timestamp = <<receive_timestamp + @ntp_constant::size(32), 0::size(32)>>
  transmit_timestamp = <<transmit_timestamp + @ntp_constant::size(32), 0::size(32)>>

  <<header::binary, id::binary, reference_timestamp::binary, origin_timestamp::binary,
    receive_timestamp::binary, transmit_timestamp::binary>>
end
```

```elixir
def handle_continue(:loop, socket) do
  case :gen_udp.recv(socket, 0) do
    {:ok, :udp_closed} ->
      Logger.warning("UDP socket closed")

    {:error, reason} ->
      Logger.error("Error: #{reason}")

    {:ok, {ip, port, request}} ->
      packet = generate_ntp_response(request)
      :gen_udp.send(socket, ip, port, packet)
      {:noreply, socket, {:continue, :loop}}
  end
end
```

```elixir
def handle_continue(:loop, socket) do
  case :gen_udp.recv(socket, 0) do
    {:ok, :udp_closed} ->
      Logger.warning("UDP socket closed")

    {:error, reason} ->
      Logger.error("Error: #{reason}")

    {:ok, {ip, port, request}} ->
      packet = generate_ntp_response(request)
      :gen_udp.send(socket, ip, port, packet)
      {:noreply, socket, {:continue, :loop}}
  end
end
```

Code is on Github

https://github.com/andreicek/ntp_server

Extra: Custom sigils

```
<<0x7F>>
# becomes
~b(7F)
```

```
<<0x7F>>
# becomes
~b(7F)
```

```elixir
# lib/bitstring_sigil.ex
defmodule NtpServer.BitstringSigil do
  def sigil_b(string, _opts) do
    # TODO: implementation
  end
end

# lib/ntp_server.ex
defmodule NtpServer.UdpServer do
  use GenServer
  import NtpServer.BitstringSigil
end
```

```
<<0x7F>>
# becomes
~b(7F)
```

```
string #=> "e3 00 03 fa"
```

```
<<0x7F>>
# becomes
~b(7F)
```

```
string #=> "e3 00 03 fa"
|> String.upcase() #=> "E3 00 03 FA"
|> String.split("\n") #=> ["E3 00 03 FA"]
```

```
<<0x7F>>
# becomes
~b(7F)
```

```
string #=> "e3 00 03 fa"
|> String.upcase() #=> "E3 00 03 FA"
|> String.split("\n") #=> ["E3 00 03 FA"]
|> Enum.map(&String.split(&1, " ")) #=> [["E3", "00", "03", "FA"]]
|> List.flatten() #=> ["E3", "00", "03", "FA"]
```

```
<<0x7F>>
# becomes
~b(7F)
```

```
string #=> "e3 00 03 fa"
|> String.upcase() #=> "E3 00 03 FA"
|> String.split("\n") #=> ["E3 00 03 FA"]
|> Enum.map(&String.split(&1, " ")) #=> [["E3", "00", "03", "FA"]]
|> List.flatten() #=> ["E3", "00", "03", "FA"]
|> Enum.reject(&(&1 == "")) #=> ["E3", "00", "03", "FA"]
```

```
<<0x7F>>
# becomes
~b(7F)
```

```
string #=> "e3 00 03 fa"
|> String.upcase() #=> "E3 00 03 FA"
|> String.split("\n") #=> ["E3 00 03 FA"]
|> Enum.map(&String.split(&1, " ")) #=> [["E3", "00", "03", "FA"]]
|> List.flatten() #=> ["E3", "00", "03", "FA"]
|> Enum.reject(&(&1 == "")) #=> ["E3", "00", "03", "FA"]
|> Enum.join() #=> "E30003FA"
```

```
<<0x7F>>
# becomes
~b(7F)
```

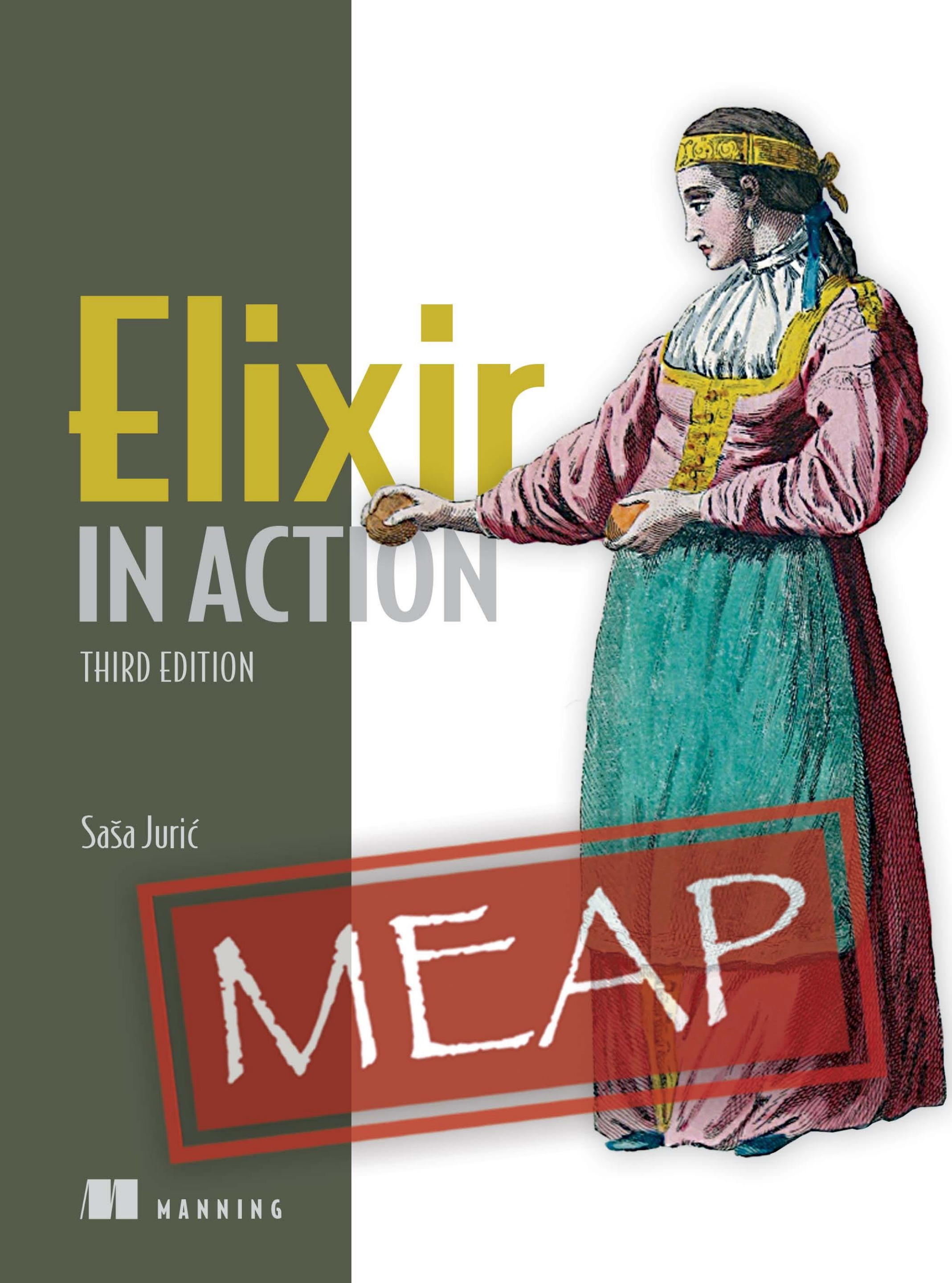
```
string #=> "e3 00 03 fa"
|> String.upcase() #=> "E3 00 03 FA"
|> String.split("\n") #=> ["E3 00 03 FA"]
|> Enum.map(&String.split(&1, " ")) #=> [["E3", "00", "03", "FA"]]
|> List.flatten() #=> ["E3", "00", "03", "FA"]
|> Enum.reject(&(&1 == "")) #=> ["E3", "00", "03", "FA"]
|> Enum.join() #=> "E30003FA"
|> Base.decode16!() #=> <<227, 0, 3, 250>>
```

```
<<0x7F>>
# becomes
~b(7F)
```

Yes, I know... Blame "plane" brain...

```
string
|> String.replace(~r/s\+/u, "")
|> Base.decode16!()
```

```
string #=> "e3 00 03 fa"
|> String.upcase() #=> "E3 00 03 FA"
|> String.split("\n") #=> ["E3 00 03 FA"]
|> Enum.map(&String.split(&1, " ")) #=> [["E3", "00", "03", "FA"]]
|> List.flatten() #=> ["E3", "00", "03", "FA"]
|> Enum.reject(&(&1 == "")) #=> ["E3", "00", "03", "FA"]
|> Enum.join() #=> "E30003FA"
|> Base.decode16!() #=> <<227, 0, 3, 250>>
```



For deep dive into Elixir I recommend:

Elixir in action, 3rd edition by Saša Jurić

35% off code: **au35jur**

https://www.manning.com/books/elixir-in-action-third-edition

JOIN US
29TH OPEN SYSTEMS DAYS
CROATIAN LINUX USERS' CONFERENCE
15-19 MAY 2024
ZAGREB - CROATIA
Algebra University, Gradišćanska 24, 10000 Zagreb

Come join us on the largest FOSS conference in Eastern Europe

20% off code: **DC24OPEN**

https://dorscluc.org/tickets