OpenSIPS As An Entreprise UC Solution

11 May 2016

About Us

About Us

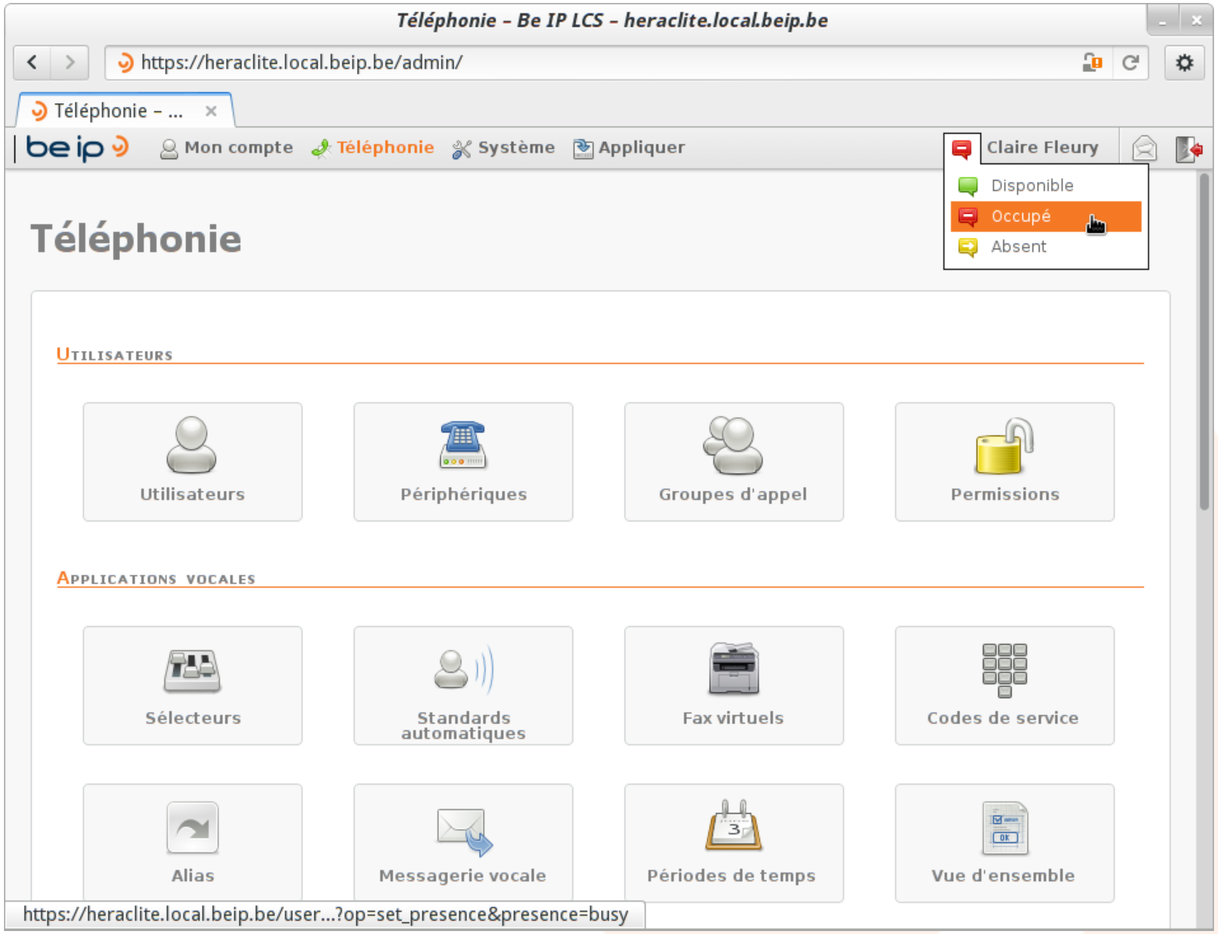
About Be IP

- We develop a SIP UC platform since 2003
- Around 22k users mainly in Belgium
- Most customers are from government agencies

About Us

Our product

- "On Premises" or "In The Cloud"
- SIP components
  - Asterisk since 2003
  - OpenSIPS since 2012
- Feature set
  - Similar to traditional proprietary vendors
  - Strong focus on reliability / redundancy
  - Strong focus on simplicity



About Us

About OpenSIPS

- Standard roles
  - Registrar
  - Proxy
  - Call Forwarding Server
  - Presence Server
  - …
- Release
  - 1.8.x with custom patches
    - https://github.com/dsandras/opensips
  - 2.1.x in our labs

Architecture

Architecture

Simple architecture

- Just a summary here
- Will evolve in the coming months / years

Redundant architecture

- Maximum two main servers
  - Active-active redundancy
  - OpenSIPS with a shared MariaDB SQL database
  - DNS SRV and/or NAPTR as underlying mechanism

Architecture

Redundant architecture

- Several satellite servers
  - Running OpenSIPS
  - The location and subscriber tables are replicated using a script
  - Limited SIP Trunking support
    - Requires a local trunk to handle inbound and outbound calls
    - Handles number rewriting - external number vs internal number

Our OpenSIPS Script

Our OpenSIPS Script

Handles SIP requests and responses

- Initial requests
- In-Dialog requests
- Calls & Call Counting
- Redundancy
- Presence
- Instant Messaging
- …

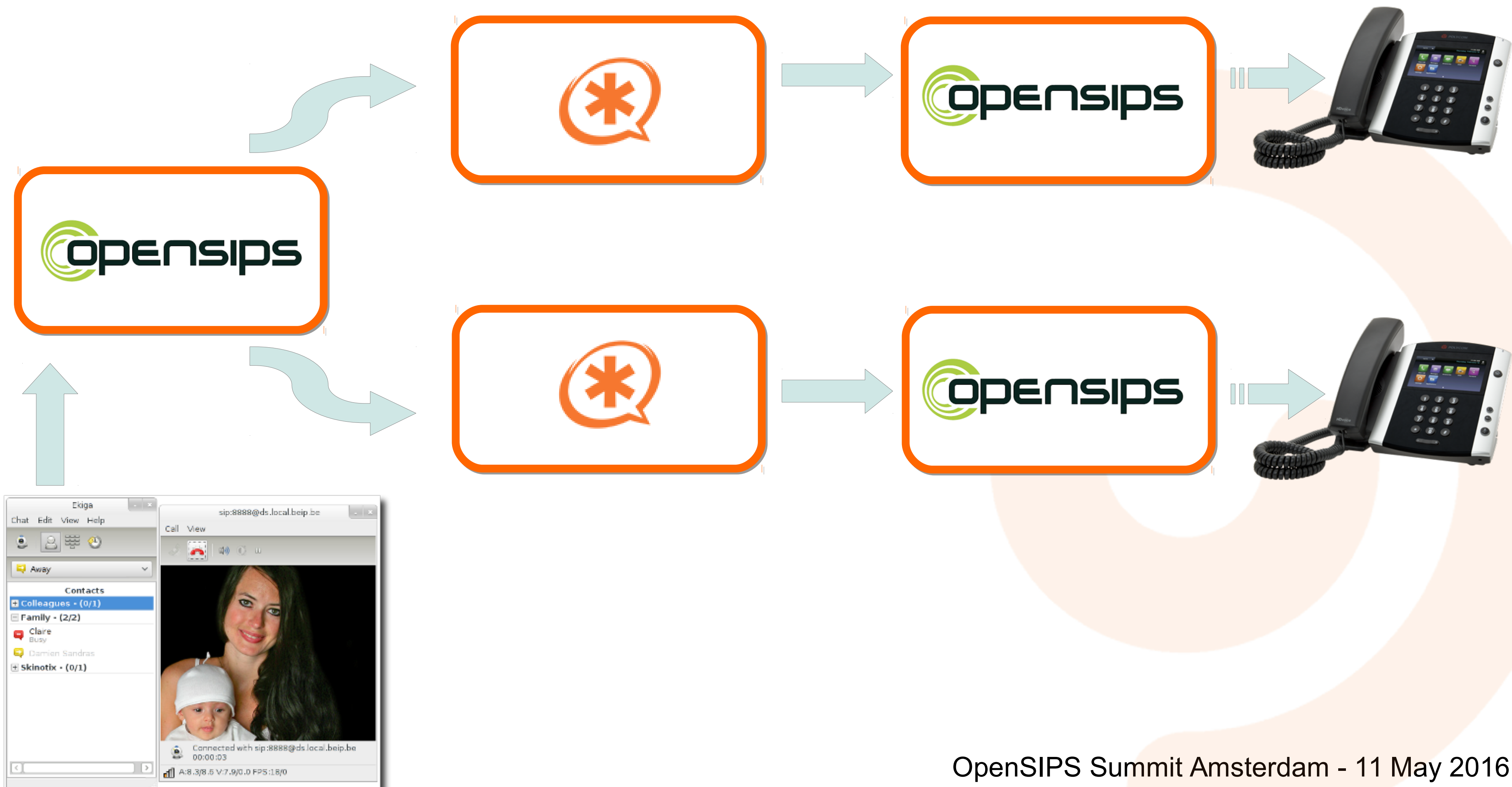
Relays to local Asterisk after processing

- Fallback to “redundant” Asterisk if required

Generated from templates

Our OpenSIPS Script

In terms of calls :



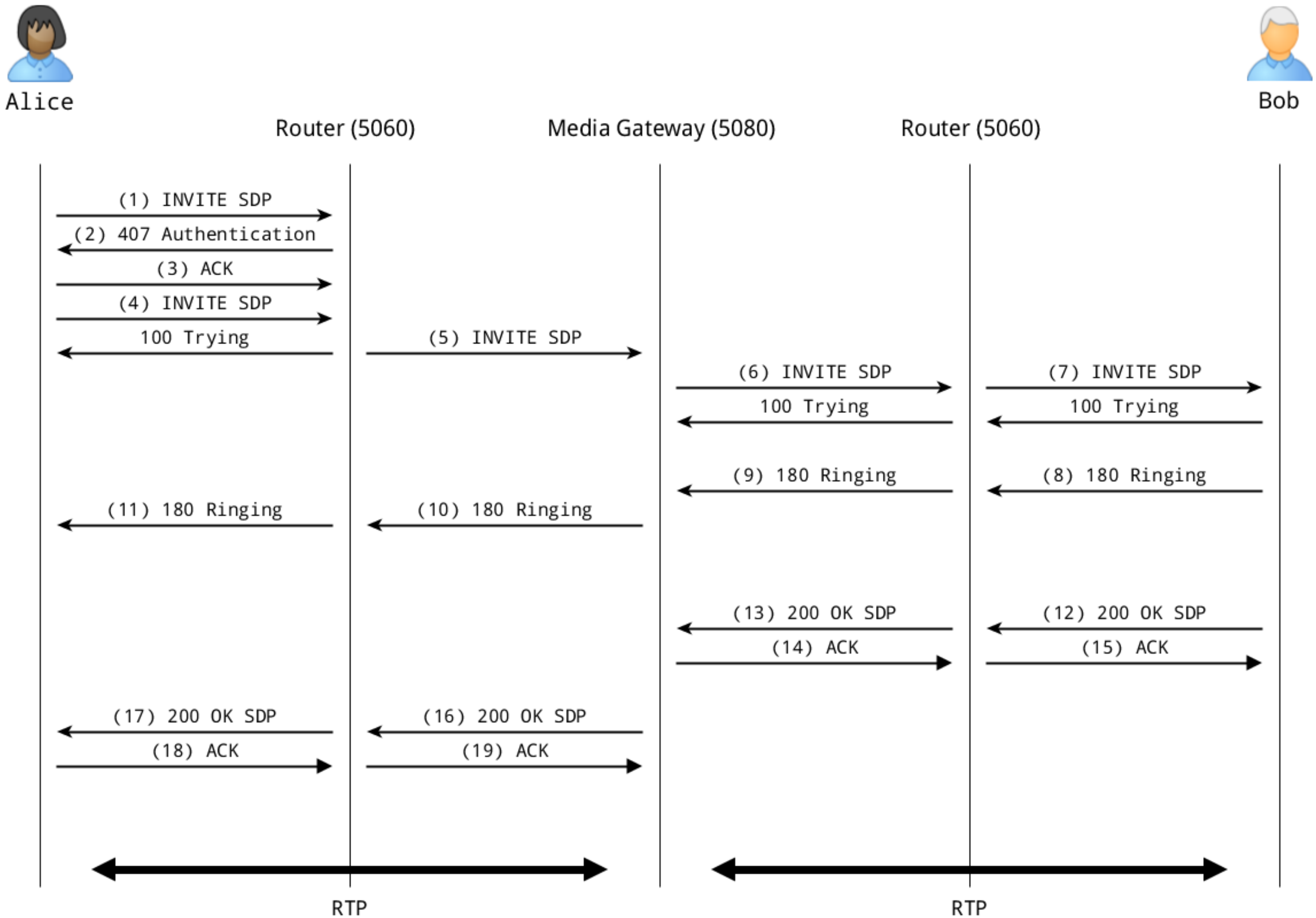
Initial INVITE Routing

Initial INVITE Routing

Two possible legs

- An inbound leg
  - From a Trunk or from a Peer
  - To Asterisk
- An outbound leg
  - To a Trunk or to a Peer
  - From Asterisk
- Both legs are handled similarly
  - Normal script routing – `route`
  - Reply route handling – `t_on_reply`
  - Failure route handling – `t_on_failure` / `t_on_failure_reply`

Initial INVITE Routing



Initial INVITE Routing

Information passing

- From one leg to the other
- In OpenSIPS
  - Mainly `cache_store` / `cache_fetch`
    - `Redis` if no redundancy impact
    - `SQL` in other cases
  - Also `get_dialog_info` using a key like the inbound Call-ID
- Between OpenSIPS and Asterisk
  - Using custom `X-BeIP` – headers that we add & remove
  - Using `cache_store / cache_fetch` / `AGI` scripts with `Redis`

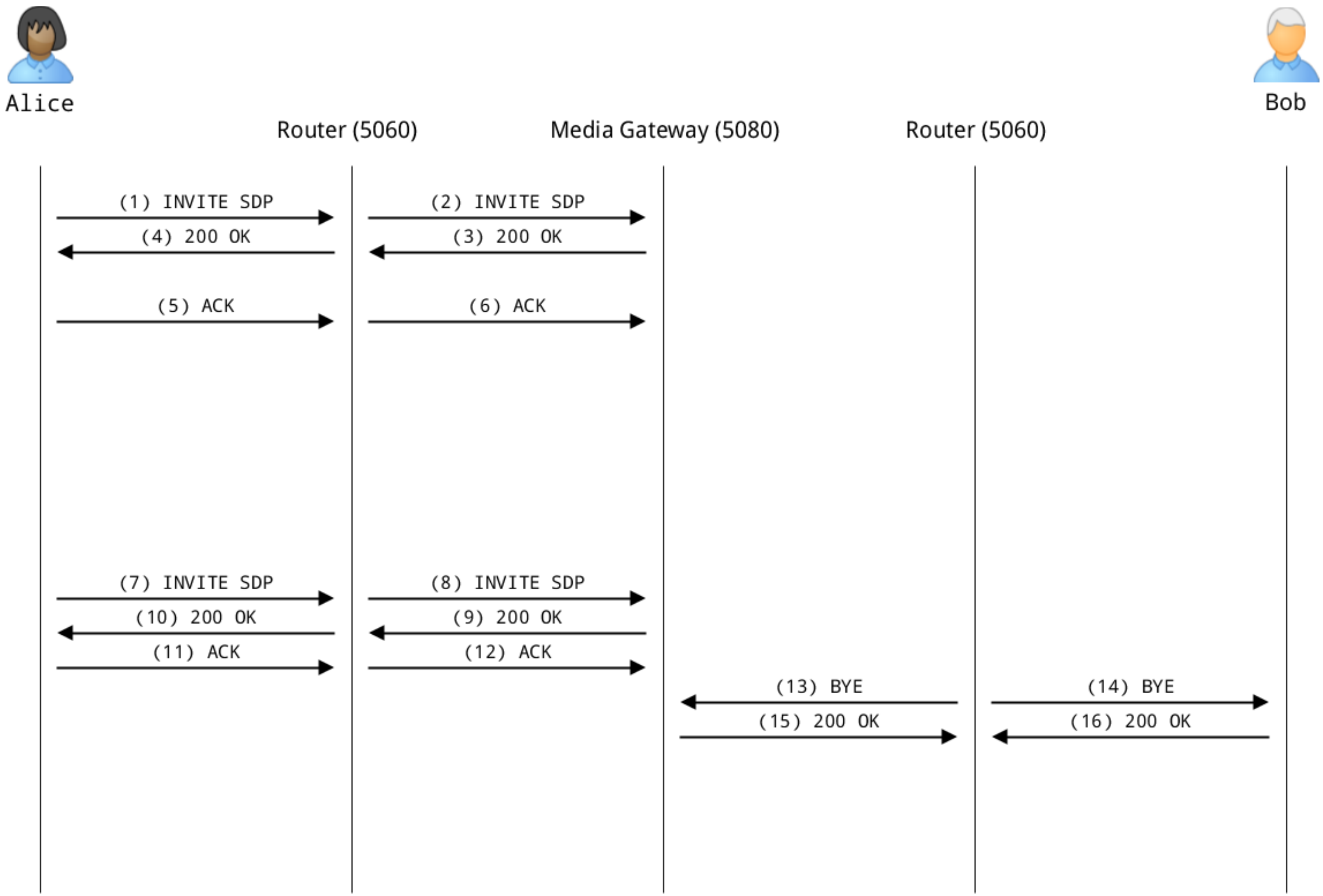
Inbound Call Leg

Inbound Leg

Inbound Call Leg – Main Route

- Caller/Called ID Name substitution
  - Currently with an SQL lookup, soon with a REST API
    - Issue is performance / latency
  - Used for address book integration / cellphone integration
  - From / To substitution
    - `uac_replace_from` / `uac_replace_to` for INVITE substitution
    - `$avp(new_from)` / `$avp(new_to)` for dialog-info NOTIFY requests
- Session Timers handling
    - `sst_flag`

Inbound Leg


Alice
Bob
Router (5060)
Media Gateway (5080)
Router (5060)
(1) INVITE SDP
(2) INVITE SDP
(4) 200 OK
(3) 200 OK
(5) ACK
(6) ACK
(7) INVITE SDP
(8) INVITE SDP
(10) 200 OK
(9) 200 OK
(11) ACK
(12) ACK
(13) BYE
(14) BYE
(15) 200 OK
(16) 200 OK

Handling Calls

Inbound Call Leg – Main Route

- Call recording
    - Using RTProxy, If required
- Call Pick-up
    - First check if the user has the correct rights
    - We need to determine which Asterisk handles the call
    - Asterisk INVITE-based Call Pick-up
        - `603` response code on pick-up failure
        - Route to Asterisk and handle failure for redundancy – `t_on_failure_reply`
    - SIP-based Call Pick-up
        - `get_dialog_info` on `callid` specified in `Replaces` header

          `Replaces: 12345678,to-tag=7744;from-tag=5693`
        - Route to correct Asterisk directly and relay Asterisk response code

Inbound Leg

Inbound Call Leg – Main Route

- Finally, route to
  - Local Asterisk
  - Remote Asterisk if the callee or caller has an ongoing call there
    - Uses `get_profile_size` and shared profiles
- Don't forget reply and failure routes
    - `t_on_reply`
    - `t_on_failure`

Inbound Leg

Inbound Call Leg – Reply Route

- Called ID name substitution
  - Updates called ID name in the reply (e.g. 180 Ringing)
  - Uses `P-Asserted-Identity` header
- Add a Warning header if required
  - Could be a warning from the other leg
    - Some SIP providers / SIP entities use Warning headers to indicate … warnings
    - `cache_fetch` + `redis`
  - Could be an internal warning
    - Remote peer presence status indication *"Jack is busy"*
- Store callback-on-busy information
  - If reply code is `486` busy
  - `cache_store` + `SQL` (because of redundancy)

Inbound Leg

Inbound Call Leg – Failure Route

- Handle call pick-up failure
  - For Asterisk-based pick-up
  - In that case, re-route to other Asterisk
- Handle lack of answer from local or remote Asterisk
  - Fallback or `503`

Outbound Call Leg

Handling Calls

Outbound Call Leg – Main Route

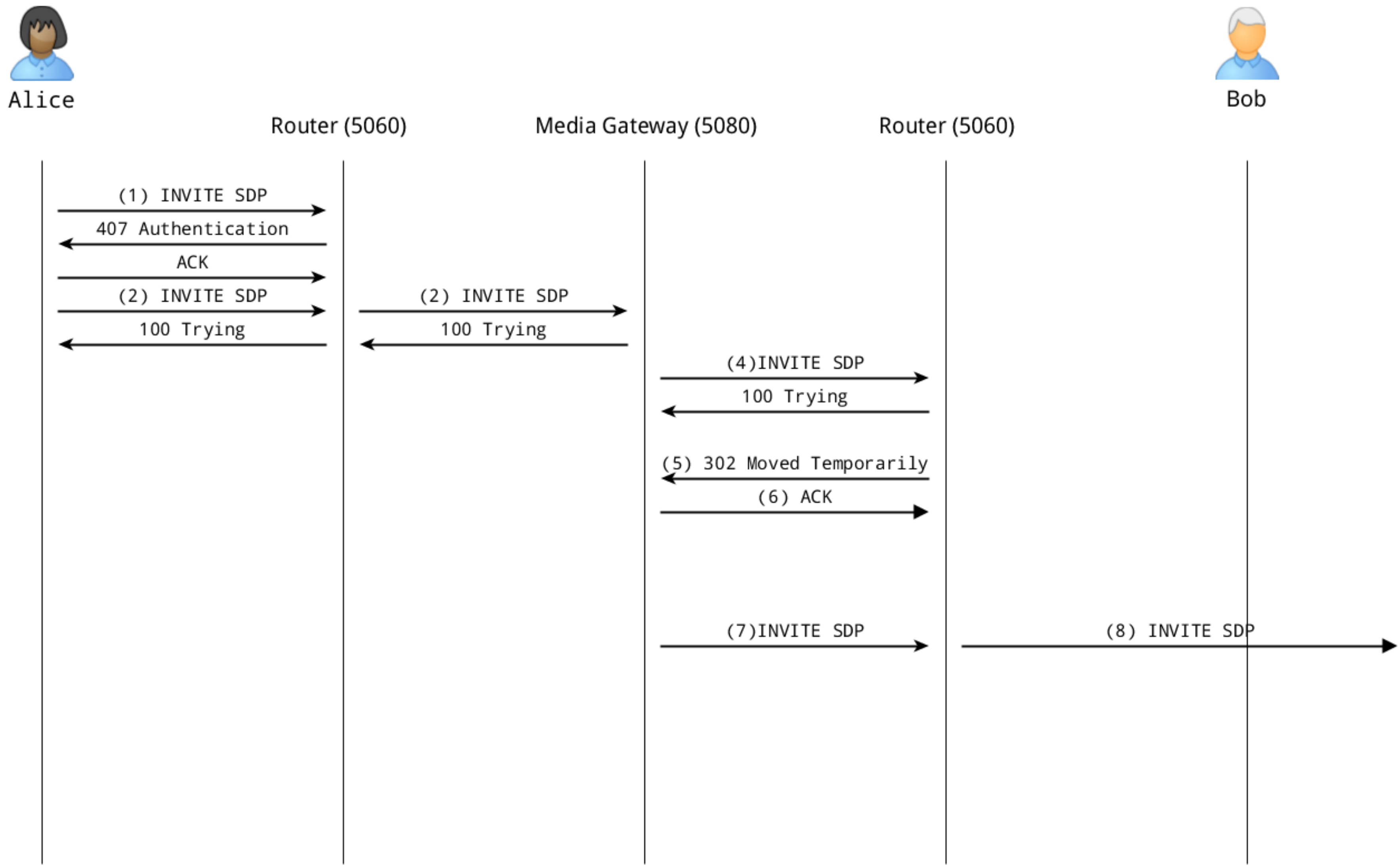
- Similar to the inbound call leg
  - Caller ID Name substitution
  - Enable Session Timers
  - Enable Call Recording

Handling Calls

Outbound Call Leg – Main Route

- Call forwarding
  - Depending
    - On the aggregated presence status
    - On the call origin: internal (peer) / external (trunk)

Handling Calls



Handling Calls

Outbound Call Leg – Main Route

- Call forwarding
  - Aggregated presence status
    - Comes from CacheDB – `cache_fetch` (custom patch)
    - Simplified into 4 states from RFC 4480 : default, away, busy, vacation
  - Immediate call forwarding
    - Per configuration: presence state / follow me
    - Or other failure cases
      - No call waiting and ongoing call – `get_profile_size`
      - Offline / peer exists but no contact records – `! registered("location") && db_does_uri_exist()`

Handling Calls

Outbound Call Leg – Main Route

- Call forwarding
  - Delayed call forwarding
    - No answer
    - Blind transfer failure – `X-BeIP-BlindXFER`
  - Can be ignored (e.g. Queue Calls)
  - `X-BeIP-` header
- Distinctive rings
  - Internal / external / group calls
  - Can be ignored
  - `Alert-Info` header

Handling Calls

Outbound Call Leg – Main Route

- Call Pick-up
  - Store `get_dialog_info` parameters
    - `call-id` as key
    - Callee user part
- Finally, locate and relay
  - Handle forking
    - Parallel forking
    - Serial forking – based on Q (might be emulated)
- Don’t forget reply and failure routes
    - `t_on_reply`
    - `t_on_failure`

Initial INVITE Routing

Outbound Call Leg – Reply Route

- Handle remote peer / trunk replies
  - Store a Warning header if required
    - Could be a warning from the remote trunk – `cache_store` + `redis`
  - Store the reply code
    - Reply code is stored – `cache_store` + `redis`
    - We want Asterisk to use the real reply code to react appropriately – AGI is used for redis interaction

Initial INVITE Routing

Outbound Call Leg – Failure Route

- More call forwarding
  - No answer
  - Offline
  - Busy
- Or relay error code back to Asterisk

Other OpenSIPS Features

Other Features

Other features we implement with OpenSIPS

- Call Recording
- Call Counting
- Security
- Presence
- ...

The Future

The Future

Drop the “Two Servers” Limitation

- Each server should handle its own devices
- Each device should be reached from its main server

Multi-tenant mode / Multi-domain

WebRTC

Thank You!

Questions?

Damien Sandras
+32 67 28 76 75
ds@beip.be
http://www.beip.be


be ip
www.beip.be