building rtc services with lua and kamailio

Daniel-Constantin Mierla
Co-Founder Kamailio Project
www.asipto.com
@miconda

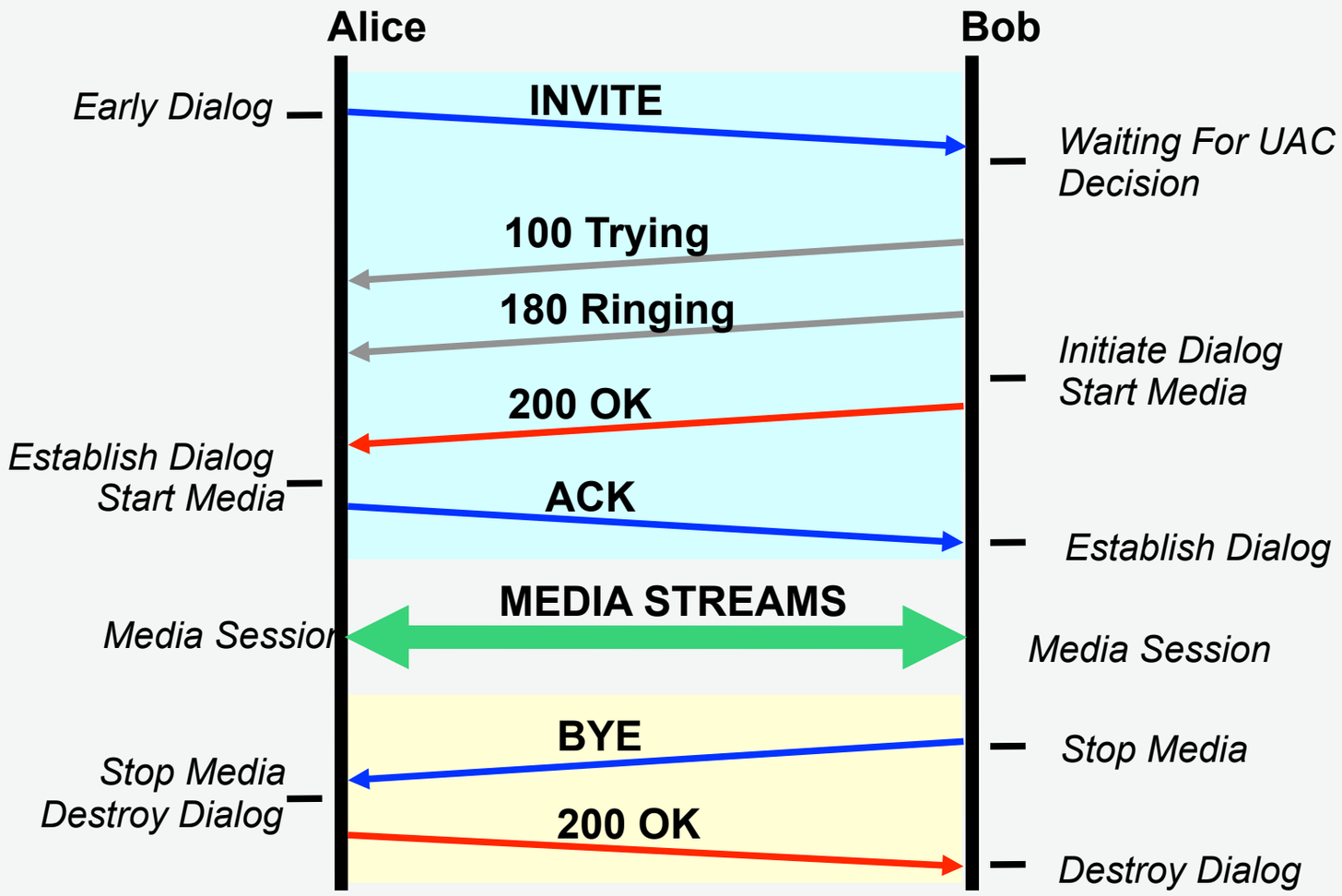
- www.kamailio.org
- open source sip server
- aka sip router or sip proxy

- focus in scalability and flexibility
- sip (session initiation protocol)
- ietf open standard - rfc3261
- https://tools.ietf.org/html/rfc3261

- can be used for:
  - voice (ip telephony - voip)
  - video
  - instant messaging
  - presence
  - other rtc (gaming, desktop sharing, …)
  - secure communications (TLS), IPv4/IPv6
- quite similar packet structure to http
- adopted by ITU/3GPP (signaling for voice in 4G+)

SIP CALL

voip call maps over sip dialog
sip dialog is a sequence of sip transactions
sip transaction is a sip request and one or many responses



SIP REQUEST

INVITE

Start line	**INVITE sip:user@sipserver.com SIP/2.0**
Message headers	**Via: SIP/2.0/UDP 10.10.10.10:5060;branch=z9hG4bKxy** **From: "Me" <sip:me@sipserver.org>;tag=a012** **To: "User" <sip:user@sipserver.org>** **Call-ID: d@10.10.10.10** **CSeq: 1 INVITE** **Contact: <sip:10.10.10.10:5060>** **User-Agent: SIPTelephone** **Content-Type: application/sdp** **Content-Length: 251**
Message body	**v=0** **o=audio1 0 0 IN IP4 10.10.10.10** **s=session** **c=IN IP4 10.10.10.10** **m=audio 54742 RTP/AVP 4 3** **a=rtpmap:4 G723/8000** **a=rtpmap:3 GSM/8000**

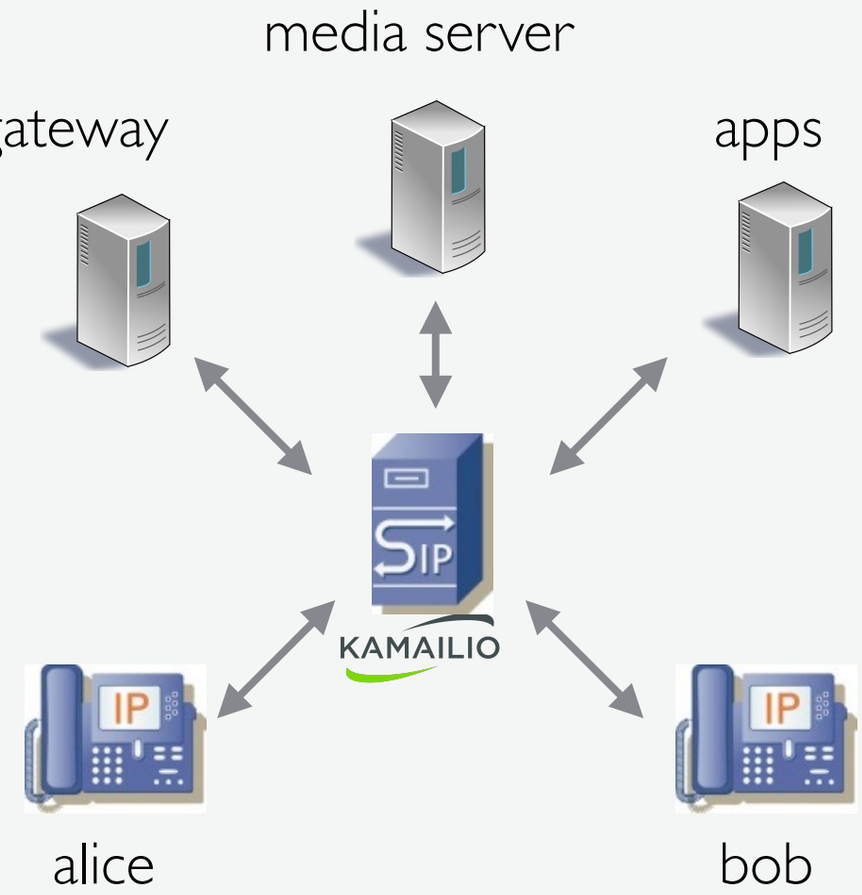
SIP RESPONSE

INVITE
sip response == sip reply

Start line	**SIP/2.0 200 OK**
Message headers	**Via: SIP/2.0/UDP 10.10.10.10:5060;branch=z9hG4bKxy** **From: "Me" <sip:me@sipserver.org>;tag=a012** **To: "User" <sip:user@sipserver.org>;tag=b034** **Call-ID: d@10.10.10.10** **CSeq: 1 INVITE** **Contact: <sip:10.10.10.20:5060>** **User-Agent: SIPSoftPhone** **Content-Type: application/sdp** **Content-Length: 123**
Message body	**v=0** **o=audio2 0 0 IN IP4 10.10.10.20** **s=session** **c=IN IP4 10.10.10.20** **m=audio 62043 RTP/AVP 0 4**

- the base role of kamailio

  - routing SIP packets: requests and responses



FEATURES

- Modular SIP Proxy, Registrar and Redirect server
- Designed for scalability and flexibility
- IPv4, IPv6, UDP, TCP, TLS, SCTP, WebSocket
- NAT Traversal, internal and external caching engines
- JSON, XMLRPC, HTTP APIs
- IMS Extensions, SIP-I/SIP-T, IM & Presence
- SQL and NoSQL backends
- Asynchronous processing (TCP/TLS, SIP routing), external event API
- Embedded interpreters (**Lua**, Perl, Python, .Net, Java)
- Load balancing, LCR, DID routing, Number portability, …

OWN CFG SCRIPTING LANGUAGE

```
#!KAMAILIO

debug=2
log_stderror=yes
listen=udp:127.0.0.1

loadmodule “rr.so”
loadmodule “sl.so”
loadmodule “pv.so”
loadmodule “textops.so”
loadmodule “siputils.so”

modparam(“rr”, “append_fromtag”, 1)

# load modules
request_route {
   route(WITHINDLG);
   if(is_method(“INVITE”)) {
        $du = “sip:127.0.0.1:9”;
        record_route();
        forward();
        exit;
   }
   sl_send_reply(“403”, “Not allowed”);
   exit;
}

route[WITHINDLG] {
    if(!has_totag()) return;
    if(loose_route()) {
        forward();
        exit;
    }
   sl_send_reply(“404”, “Not here”);
   exit;
}
```

kamailio.cfg

- comments
- preprocessor directive
- core parameters
- custom core parameters
- load modules
- module parameters
- routing blocks
  - *expressions*
  - *conditions*
  - *actions*
  - *control statements*
  - *subroutes*

KAMAILIO.CFG LANGUAGE

- initial design in 2001-2002
  - targeting optimization for sip routing
- characteristics
  - sort of pre-compilation at startup
  - linear execution
  - routing blocks similar to functions
  - specific routing blocks executed on various events
- limitations
  - only int and string values
  - requires C coding (sometime lex and yacc) to extend it
  - startup pre-compilation not designed for reload

LUA IN KAMAILIO

- https://www.lua.org
- app_lua module added in January 2010
  - embedding the lua interpreter
  - targeting execution of lua snippets
  - using functions from inside routing blocks of kamailio.cfg
- exported a new module to lua - sr - with many submodule
  - sr.hdr, sr.pv, sr.sl, …
- limitations
  - it required C coding inside app_lua for extending the lua sr module

https://www.kamailio.org/docs/modules/stable/modules/app_lua.html

EXECUTING LUA INSIDE KAMAILIO.CFG

myscript.lua

```
...
function sr_append_fu_to_reply()
        sr.hdr.append_to_reply("P-From: " .. sr.pv.get("$fu") .. "\r\n");
end
...
```

```
...
modparam("app_lua", "load", "/usr/local/etc/kamailio/lua/myscript.lua")
...
request_route {
    ...
    if(!lua_run("sr_append_fu_to_reply")) {
        xdbg("SCRIPT: failed to execute lua function!\n");
    }
    ...
}
...
```

kamailio.cfg

apps

document

json
xml
serialization

---

json
jansson
rtjson
xmlops
transformations

transport

http
jsonrpc
sip
custom

---

utils
http_client
jsonrpc-c
evapi
uac

control

interpreter

---

native
lua
python
javascript
perl - .net - java

OVERCOMING LIMITATIONS

- main kamailio.cfg scripting limitations
  - reload and language extensions
- kamailio.cfg sections at runtime
  - passive: global parameters and module sections (attributes)
  - active: routing blocks (actions)
- solution - kamailio v5.0
  - allow use of lua for the entire active part (actions)

KAMAILIO 5.0

ROUTING LOGIC
USING DIFFERENT
INTERPRETERS

kemi

kamailio embedded interface

a framework that allows selecting
your favourite scripting language
for writing sip routing rules

FOSDEM 2017 - Brussels

KEMI & KAMAILIO.CFG

- not affected
  - core parameters
  - loading modules
  - module parameters
- new
  - specify the language for routing rules (routing blocks)
  - cfgengine="language"
  - at this moment, language can be: native, lua, javascript, python
  - default is 'native'
  - 'lua' requires 'app_lua' module
  - 'python' requires 'app_python' module
  - 'javascript' requires 'app_jsdt' module

KEMI & LUA

- new lua module
  - KSR
- characteristics
  - automatic exports of new functions to KSR
  - no C coding anymore in app_lua
  - routing blocks entirely written in lua
- backward compatible
  - sr module still available
  - embedded execution inside kamailio.cfg still possible

NATIVE VS. LUA

```
request_route {

        # per request initial checks
        route(REQINIT);

        # NAT detection
        route(NATDETECT);

        # CANCEL processing
        if (is_method("CANCEL")) {
                if (t_check_trans()) {
                        route(RELAY);
                }
                exit;
        }

        # handle requests within SIP dialogs
        route(WITHINDLG);
```

```
function ksr_request_route()

        -- per request initial checks
        ksr_route_reqinit();

        -- NAT detection
        ksr_route_natdetect();

        -- CANCEL processing
        if KSR.pv.get("$rm") == "CANCEL" then
                if KSR.tm.t_check_trans()>0 then
                        ksr_route_relay();
                end
                return 1;
        end

        -- handle requests within SIP dialogs
        ksr_route_withindlg();
```

NATIVE VS. LUA

```
# Caller NAT detection
route[NATDETECT] {
        force_rport();
        if (nat_uac_test("19")) {
                if (is_method("REGISTER")) {
                        fix_nated_register();
                } else {
                        if(is_first_hop())
                                set_contact_alias();
                }
                setflag(FLT_NATS);
        }
        return;
}
```

```
-- Caller NAT detection
function ksr_route_natdetect()
        KSR.force_rport();
        if KSR.nathelper.nat_uac_test(19)>0 then
                if KSR.pv.get("$rm")=="REGISTER" then
                        KSR.nathelper.fix_nated_register();
                elseif KSR.siputils.is_first_hop()>0 then
                        KSR.nathelper.set_contact_alias();
                end
                KSR.setflag(FLT_NATS);
        end
        return 1;
end
```

SIP ROUTING IN LUA

- features
  - detect scanning attacks and block DoS attacks
  - authentication
  - authorization
  - accounting
  - location service
  - nat traversal

https://github.com/kamailio/kamailio/blob/master/misc/examples/kemi/kamailio-basic-kemi-lua.lua

https://github.com/kamailio/kamailio/blob/master/misc/examples/kemi/kamailio-basic-kemi.cfg

DEVELOPER MODE

LUA FOR SIP ROUTING

FOSDEM 2017 - Brussels

IMPORTANT

- **fast, very fast**
  - handle thousands of call setups per second
  - critical feature
- **dynamic**
  - KSR exports a matter of loaded native kamailio modules
- **preserve experience**
  - stop execution or routing script

STOP ROUTING SCRIPT EXECUTION

- native kamailio.cfg
  - exit
- lua
  - libc exit results in killing kamailio - not wanted
- solution
  - KSR.x.exit()
  - workaround using lua error()

KSR.X.EXIT()

```
static str _sr_kemi_lua_exit_string = str_init("~~ksr~exit~~");


str* sr_kemi_lua_exit_string_get(void)
{
	return &_sr_kemi_lua_exit_string;
}

static int sr_kemi_lua_exit (lua_State *L)
{
	str *s;

	LM_DBG("script exit call\n");
	s = sr_kemi_lua_exit_string_get();
	lua_getglobal(L, "error");
	lua_pushstring(L, s->s);
	lua_call(L, 1, 0);
	return 0;
}
```

KSR.X.EXIT()

```
...
ret = lua_pcall(_sr_L_env.LL, n, 0, 0);
if(ret!=0) {
        txt.s = (char*)lua_tostring(_sr_L_env.LL, -1);
        if(txt.s!=NULL) {
                if(strcmp(txt.s[n], _sr_kemi_lua_exit_string.s[n]) !=0 ) {
                        LM_ERR("error from Lua: %s\n", txt.s);
                } else {
                        LM_DBG("ksr error call from Lua: %s\n", txt.s);
                }
        } else {
                LM_ERR("error from Lua: unknown\n");
        }
        lua_pop(_sr_L_env.LL, 1);
        if(n==1) {
                return 1;
        } else {
                LM_ERR("error executing: %s (err: %d)\n", func, ret);
                return -1;
        }
}
...
```

DYNAMIC & FAST

initial approach

```
void lua_sr_kemi_register_module(lua_State *L, str *mname, int midx)
{
	int ret;
#define LUA_SR_SBUF_SIZE   1024
	char sbuf[LUA_SR_SBUF_SIZE];

	ret = snprintf(sbuf, LUA_SR_SBUF_SIZE-1,
			"KSR.%.*s = {}\n"
			"KSR.%.*s.__index = function (table, key)\n"
			"  return function (...)\n"
			"    return KSR_MOD_C('%.*s', %d, key, ...)\n"
			"  end\n"
			"end\n"
			"setmetatable(KSR.%.*s, KSR.%.*s)\n",
			mname->len, mname->s,
			mname->len, mname->s,
			mname->len, mname->s,
			midx,
			mname->len, mname->s,
			mname->len, mname->s
		);
	ret = luaL_dostring(L, sbuf);

	LM_DBG("pushing lua KSR.%.*s table definition returned %d\n",
			mname->len, mname->s, ret);
}
```

DYNAMIC & FAST

initial approach

- everything is a hash table in Lua
  - including modules
- dive into internals
  - change the indexing function of module hash table

DYNAMIC & FAST

initial approach

```
int sr_kemi_KSR_MOD_C(lua_State* L)
{
	str mname;
	int midx;
	str fname;
	mname.s = (char*)lua_tostring(L, 1);
	midx = lua_tointeger(L, 2);
	fname.s = (char*)lua_tostring(L, 3);
	if(mname.s==NULL || fname.s==NULL) {
		LM_ERR("null params: %p %p\n", mname.s, fname.s);
		return app_lua_return_false(L);
	}
	mname.len = strlen(mname.s);
	fname.len = strlen(fname.s);
	LM_DBG("module function execution of: %s.%s (%d)\n",
			mname.s, fname.s, midx);
	return sr_kemi_exec_func(L, &mname, midx, &fname);
}
```

CONCERNS

- for each execution of a KSR function, search in the list of exports by module and function name

- Lua should be better at indexing its functions
  - avoid not benefiting of improvements done by Lua
- optimization
  - provide an index for KSR submodule

DYNAMIC & FAST

final approach

- estimated 1024 max KSR functions
- auto generated 1024 Lua exports functions
- index KSR functions in one-to-one relation

DYNAMIC & FAST

final approach

Exported To Lua	Internal Kamailio Structure
sr_kemi_lua_exec_func_0	sr_kemi_lua_export_t(t_relay)
. . .	. . .
sr_kemi_lua_exec_func_100	sr_kemi_lua_export_t(sl_send_reply)
. . .	. . .
null	null
. . .	. . .

DYNAMIC & FAST

final approach

```
static int sr_kemi_lua_exec_func_0(lua_State *L)
{
	return sr_kemi_lua_exec_func(L, 0);
}

...

static int sr_kemi_lua_exec_func_1023(lua_State *L)
{
	return sr_kemi_lua_exec_func(L, 1023);
}

...
typedef struct sr_kemi_lua_export {
	lua_CFunction pfunc;
	sr_kemi_t *ket;
} sr_kemi_lua_export_t;

static sr_kemi_lua_export_t _sr_kemi_lua_export_list[] = {
	{ sr_kemi_lua_exec_func_0, NULL},
...
	{ sr_kemi_lua_exec_func_1023, NULL},
	{NULL, NULL}
};
```

DYNAMIC & FAST

final approach

```
sr_kemi_t *sr_kemi_lua_export_get(int idx)
{
        if(idx<0 || idx>=SR_KEMI_LUA_EXPORT_SIZE)
                return NULL;
        return _sr_kemi_lua_export_list[idx].ket;
}

...

int sr_kemi_lua_exec_func(lua_State* L, int eidx)
{
        sr_kemi_t *ket;

        ket = sr_kemi_lua_export_get(eidx);
        return sr_kemi_lua_exec_func_ex(L, ket, 0);
}
```

PERFORMANCES

final approach

- testing system
  - VirtualBox with 2GB or RAM and 2 processors, having Linux Mint as guest OS. The host was a MAC OS X running on a Mid 2012 model of Macbook Pro
- results
  - INTERPRETER - AVERAGE - MIN - MAX
  - NATIVE - 302.275 - 6 - 3824
  - LUA - 308.32 - 6 - 3596

LUA BENEFITS

- documentation of the programming language
- independent evolution and new features
- larger set of extensions
- sensitive, but: coroutines, threads, …
- routing script reload

SIPWEET

Send tweets on SIP events

- missed call notification
- instant messaging
- reminders

Monitor tweets for SIP services

- tweet-to-dial
- reminder setup
- scheduled calls
- profile update

Design

- Lua twitter library
- Twitter operation is an HTTP request
  - can take some time to be processed
  - we cannot afford that when processing SIP signaling
  - solution: use asynchronous processing
    - config file message queue
    - dedicated process for twitter operations
- Kamailio modules
  - app_lua
  - mqueue
  - rtimer
  - sqlops
- Sample implementation
  - notification of a missed call
  - use of Twitter direct message

SIPWEET

Lua script

```
-- SIPweet

-- loading module
require("twitter")

local initialized = 0
local mytw

function sipweetinit()
    if initialized == 0 then
            mytw = twitter.client("Consumer Key",
                        "Consumer Secret",
                        "OAuth Token",
                        "OAuth Token Secret");
        initialized = 1
    end
end

function sipweetdm(userid, message)
    sipweetinit()
        mytw:sendMessage{ user = userid, text = message }
end
```

Database table

```
CREATE TABLE `sipweetusers` (
 `twuser` varchar(64) NOT NULL,
  `sipuser` varchar(64) NOT NULL,
 UNIQUE KEY (`twuser`),
  UNIQUE KEY (`sipuser`)
);
```

```
[[select twuser from sipweetusers
     where sipuser='']]
     .. KSR.pv.get("$rU") .. [['']]



sipweetdm(twuser, [[Missed call from]]
   .. KSR.pv.get("$fU") )
```

RESOURCES

- https://www.lua.org
- https://www.kamailio.org
- https://www.kamailio.org/docs/modules/devel/modules/app_lua.html
- https://www.kamailio.org/docs/modules/devel/modules/app_lua.html#app_lua.r.list
- https://www.kamailio.org/docs/modules/devel/modules/app_lua.html#app_lua.r.reload
- https://www.kamailio.org/wiki/embeddedapi/devel/lua

KAMAILIO WORLD
CONFERENCE & EXHIBITION
BERLIN, GERMANY, MAY 8-10, 2017

THANK YOU!!!

QUESTIONS?

Daniel-Constantin Mierla
Co-Founder Kamailio Project
www.kamailio.org
www.asipto.com
@miconda