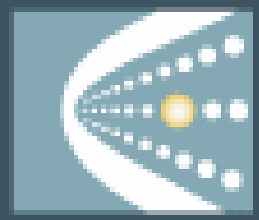
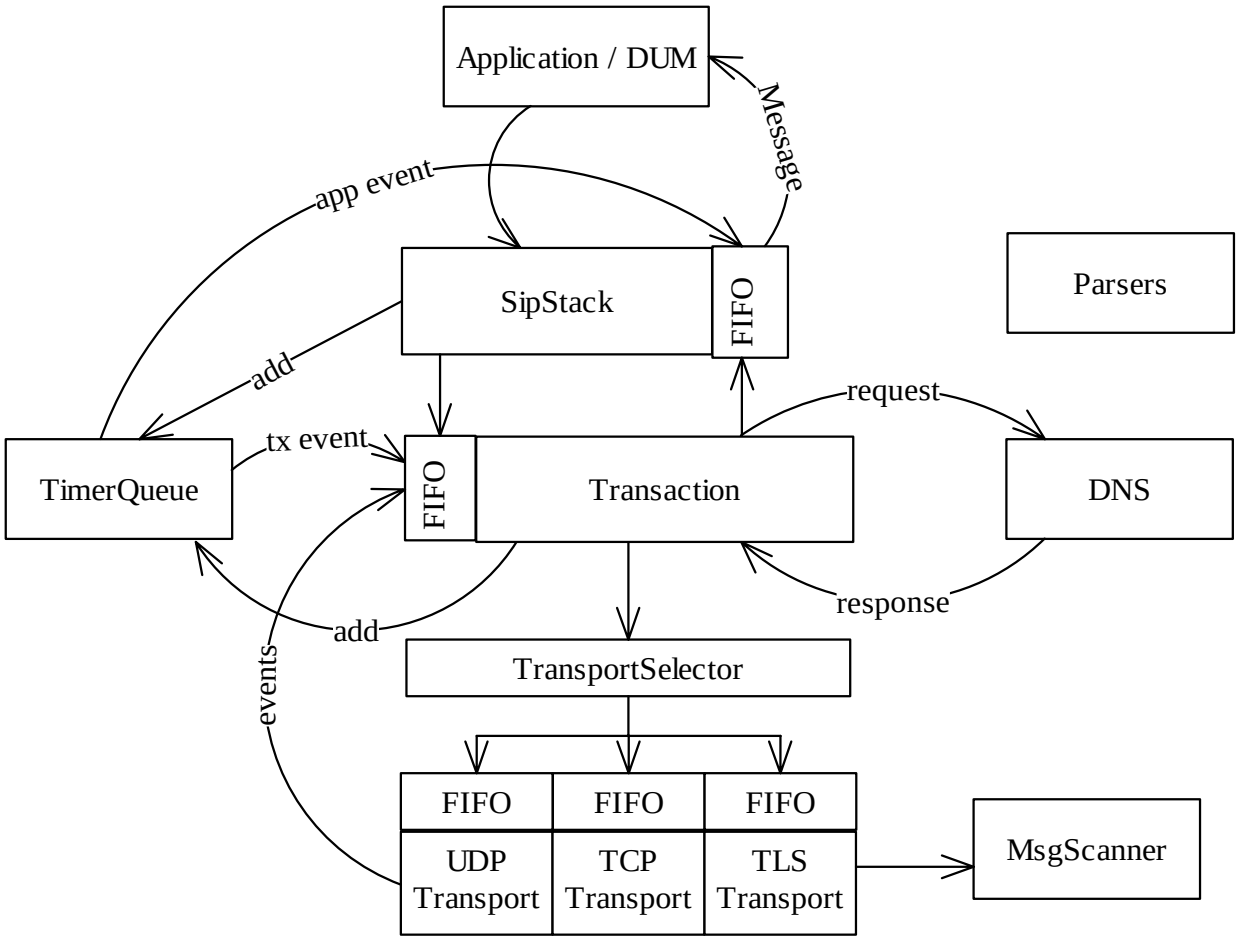


SIPfoundry

Using repro, an new Open Source SIP Server

Derek MacDonald
derek@sipfoundry.org
derek@xten.com

Architecture



“hello world”

- Use Helper class to help construct messages

```
#include “resiprocate/SipStack.hxx”
#include “resiprocate/Helper.hxx”
#include “resiprocate/SipMessage.hxx”
#include “resiprocate/NameAddr.hxx”
using namespace resip;
…
SipStack stack;
stack.addTransport(TCP, 5060);
NameAddr from(“<sip:jason@teltel.com>”);
auto_ptr<SipMessage> reg(Helper::makeRegister(from, from);
stack.send(*reg);
…
```

SIP Messages

- Let the stack manage its own thread and select call

```
SipStack stack;
StackThread stackThread(stack);
stackThread.run();

While(usleep(10)) {
  SipMessage* msg = stack.receive();
  if (msg) {
    // process the incoming sip message
    delete msg;
  }
}
```

SIP Headers

- http://www.sipfoundry.org/reSIProcate/using.txt

```
SipMessage msg;
NameAddr to;
to.uri().user() = “jason”;
to.uri().host() = “teltel.com”;
msg.header(h_To) = to;
msg.header(h_To).displayName() = “Jason Fischl”;
assert(msg.exists(h_To));
msg.header(h_CSeq).sequence()++;

NameAddrs routes = msg.header(h_Routes);
NameAddrs reversed = routes.reverse();
for (NameAddrs::iterator i=routes.begin();
     i != routes.end(); ++i)
{
    std::cerr << *i << endl;
}
NameAddr firstRoute = routes.back();
routes.pop_back();
```

SIP Parameters

```
SipMessage request;
assert(request.header(h_To).exists(p_tag)); // exists
request.header(h_To).param(p_tag) = “jason”; // assign
request.header(h_To).remove(p_tag);  // remove
request.header(h_To).uri().param(p_q) = 1.0;  // FloatParam
request.header(h_Vias).front().param(p_branch).getTransactionId();

Auth auth;
auth.scheme() = “Digest”;
auth.param(p_nonce) = “blah”;  // QuotedDataParam
auth.param(p_algorithm) = “MD5”;

UnknownParameterType p_myparam(“myparam”);
request.header(h_RequestLine).param(p_myparam) = “myvalue”;
```

Contents

- Flexible, extensible Content management
- Multi-part MIME

```
SipMessage invite;
const SdpContents* offer =
dynamic_cast<const SdpContents*>(invite.getContents);
if (offer) {
  SipMessage ok; // make from invite request
  SdpContents answer; // make from the offer
  ok.setContents(&answer);
  delete offer;
}
```

Practicalities

- Compilers
  - g++, VS.NET, VC++ 6
  - Intel compiler
- Build environment
  - Windows
  - Linux/Unix
  - QNX
  - MAC OS X
  - Solaris
- Namespaces
  - All code in resip namespace
- Exceptions
  - BaseException
  - Must catch
- Assertions
  - Can be compiled out

Logging

```
#include “resiprocate/Logger.hxx”
#define RESIPROCATE_SUBSYSTEM resip::Subsystem::TEST
using namespace resip;
…
// can use Log::SYSLOG, Log::COUT or Log::FILE
Log::initialize(Log::COUT, Log::toLevel(“INFO”), argv[0]);
SipMessage msg;
InfoLog (<< “The message is “ << msg.brief());
DebugLog (<< “detail: “ << msg);
Log::setLevel(Log::Debug);
```

Contributors

Participating Companies:

Contributors:

- Adam Roach, adam@dynamicsoft.com
- Alan Hawrylyshen, alan@jasomi.com
- Bob Bramwell bob@jasomi.com
- Bryan Ogawa bko@cisco.com
- Cullen Jennings, fluffy@cisco.com
- David Bryan, dave@bryan.cx
- David Butcher, david@purplecomm.com
- Derek MacDonald, derek@xten.com
- Jacob Butcher, resip@nowhen.com
- Jason Fischl, jason@purplecomm.com
- Ken Kittlitz <ken@jasomi.com>
- Kevin Chmilar, kevin@jasomi.com
- Robert Sparks, rjsparks@nostrum.com
- Rohan Mahy, rmahy@cisco.com
- Ryan Kereliuk, ryker@ryker.org
- Scott Godin, slgodin@icescape.com

Motivation

- A very extensible open source proxy based on reSIProcate
  - Leverage resip's extensive transport/ipv6 support
- Easy out of box experience; the web based provisioning allows it to be used for basic authenticated calls in minutes
- closely tracks the evolving standardization efforts
  - Identity(draft-ietf-sip-identity-05) is already implemented
  - Outbound connections (draft-ietf-sip-outbound) is in progress

History

- Idea started with an e-mail from Cullen Jennings in Nov. 04
- Initial development was right before SIPit in Banff
  - Tested at SIPit in Banff
  - Based on reSIProcate and the Dialog Usage Manager

Key Features

- Correct and comprehensive implementation of the relevant standards from the SIP working groups
- Support for multiple transport protocols over both IPv4 and IPv6(UDP, TCP, TLS, DTLS)
- Rigorous security mechanisms, including the newest SIP Security IETF efforts
- Simple user management through an embedded configuration web server
- In-memory location server

Key Features (cont)

- Use of readily available databases (currently Berkeley DB and MySQL) to store user data.
  - DB abstraction to allow easy extensibility
- Extendable to support provider enhanced features while processing requests

- The core of repro is two Transaction Users (TUs)
  - A DUM instance for the Registrar
  - A Proxy TU
    - In-memory registration database shared by both
- Web server for user and fixed route configuration which persists to the DB abstraction
  - The default is Berkeley DB

Extensibility

- Repro is built on the chain of responsibility pattern
- Currently have the RequestProcessor chain
- Each RequestProcessor has a handleRequest method, which can return:
  - Continue
  - WaitingForEvent
  - SkipThisChain
  - SkipAllChains
- RequestProcessorChain is itself a RequestProcessor

Different Chains of Responsibility

- The repro design has three different feature chains
  - Features that handle requests; may send responses or add a target to the Request Context. These are “monkeys”
  - Features that can modify requests that are generated post target selection, “baboons”
  - Features that modify responses and the final response which is sent back to the server transaction, “lemurs”
- So far only monkeys have been implemented

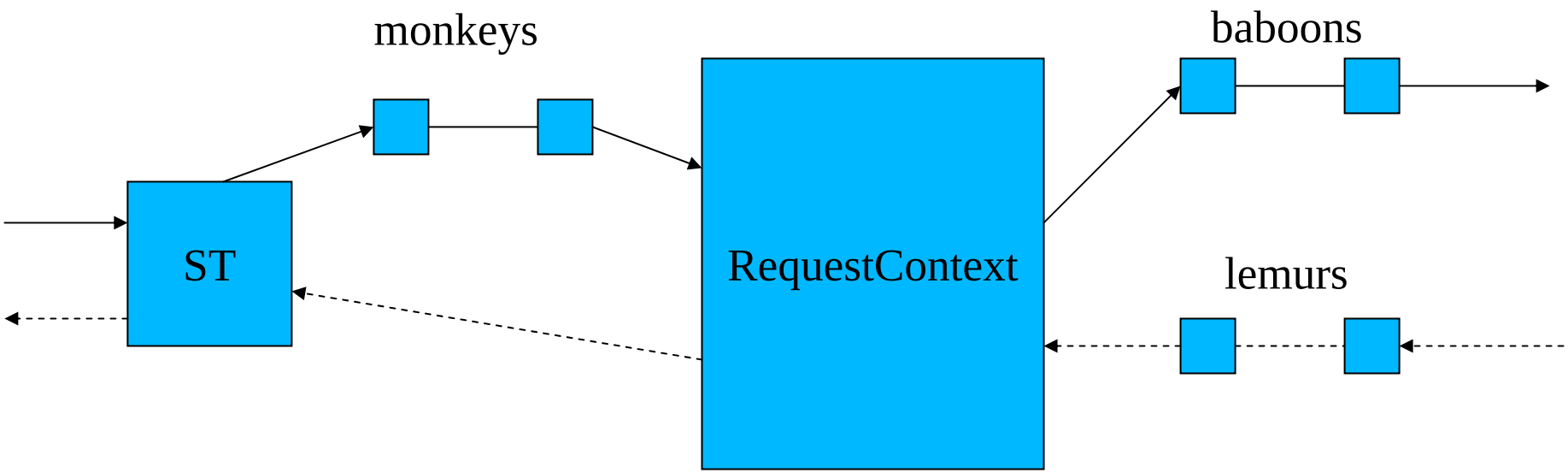
Existing Monkeys

- AmIResponsible
  - Skips the rest of the current chain if Request is not in one of this repro instances domain
  - Will have a relay list, generates a 403 if Request isn't in the relay list
- ConstantLocationMonkey
  - For test only, good sample of a very simple monkey
- StaticRoute
  - Applies configured routes from the RouteStore, which uses the abstract database
- GruuMonkey
  - work in progress to implement GRUU and sip-outbound

Existing Monkeys (cont)

- LocationServer
  - Fetches locations from the RegistrationPersistenceManager instance usually shared with the registrar
- StrictRouteFixup
  - Follows route headers(skipping rest of chain) after route header preprocessing has occurred
- DigestAuthenticator
  - Challenges requests as necessary, uses the same database as the registrar to locate secrets

Monkey Flow

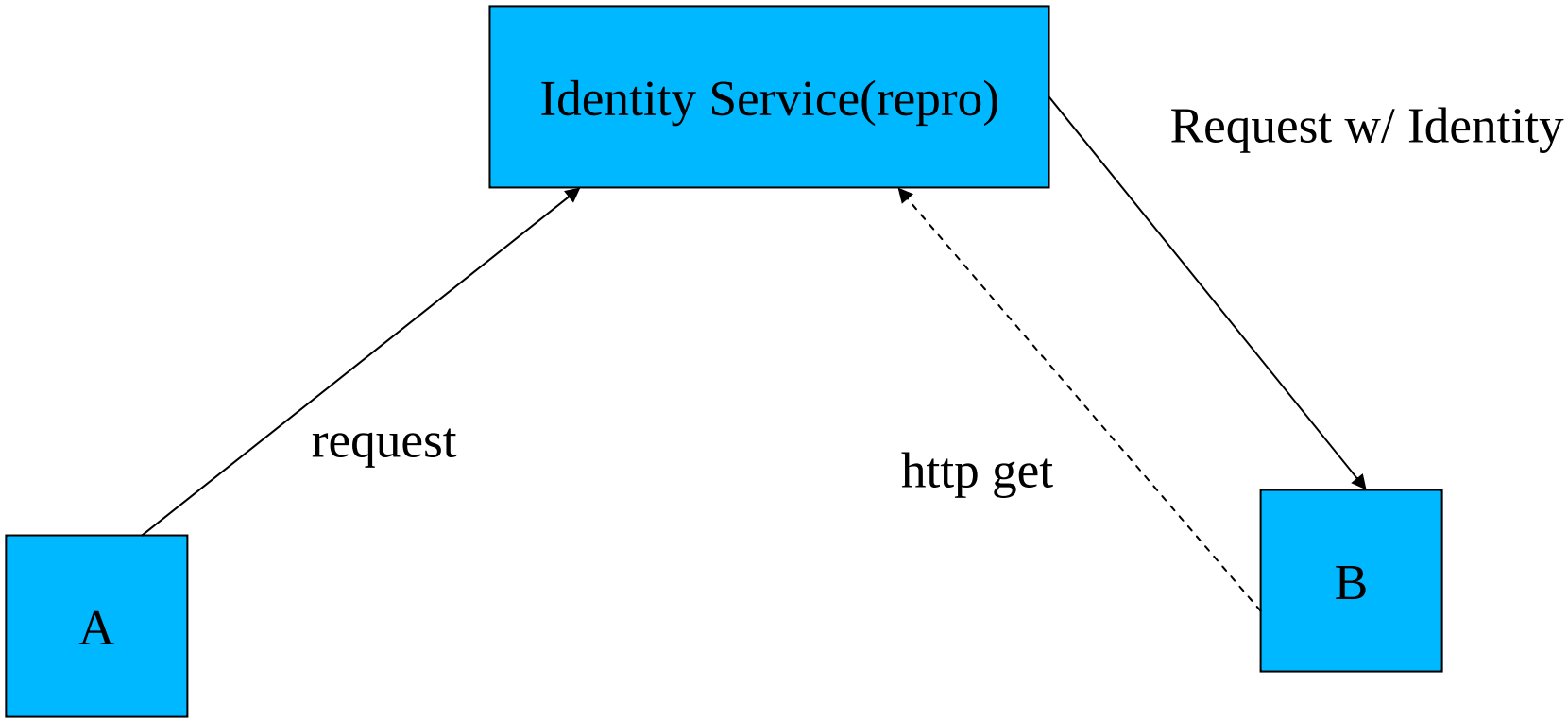


Request flow

Response Flow

Repro command line Driver

- Default monkey chain:
  - StrictRouteFixup --> IsTrustedNode --> DigestAuthenticator --> AmIResponsible --> StaticRoute --> LocationServer
- Defaults to BerkeleyDb if USE_MYSQL was not defined in the build
- Authentication can be turned off at the command line
- Record route uri can be specified on the command line


Identity Service(repro)
Request w/ Identity
request
http get
A
B

SIP Identity

- An Identity Service adds an Identity and Identity-Info header to requests, if the request passes a challenge
- Identity is a hash over particular headers(including Data& from) plus the body, signed by the private key of the service
- Identity-Info is target where the user can fetch the cert of the identity service
- An endpoint that receives the message uses this cert to validate the hash, and the hash to validate the message

Contributors

- Adam Roach, adam@estacado.net
- Cullen Jennings, fluffy@cisco.com
- Derek MacDonald, derek@xten.com
- Jason Fischl, jason@sipedge.com
- Robert Sparks, rjs@estacado.net
- Rohan Mahy, rohan@ekabal.com