Load Testing of SIP and WebRTC Infrastructures

Lorenzo Miniero

@elminiero

Kamailio World

8th May 2017,

Outline

You may remember me from last year!

What's Meetecho?

- A company born in 2009 as an academic spin-off
  - University research efforts brought to the market
  - Proudly brewed in sunny Napoli, Italy ☺
- Focus on real-time multimedia applications
  - Web conferencing only, at first
  - Then widened the scope to multimedia in general
  - Strong perspective on standardization and open source
    - *WebRTC rulez!*
- Several activities
  - Consulting services
  - Commercial support & licenses
  - Streaming of live events (e.g., IETF, ACM SIGCOMM, ...)
  - Products (conferencing, webinar, ...)

What's Meetecho?

- A company born in 2009 as an academic spin-off
  - University research efforts brought to the market
  - Proudly brewed in sunny Napoli, Italy ☺
- Focus on real-time multimedia applications
  - Web conferencing only, at first
  - Then widened the scope to multimedia in general
  - Strong perspective on standardization and open source
    - *WebRTC rulez!*
- Several activities
  - Consulting services
  - Commercial support & licenses
  - Streaming of live events (e.g., IETF, ACM SIGCOMM, ...)
  - Products (conferencing, webinar, ...)

What's Meetecho?

- A company born in 2009 as an academic spin-off
  - University research efforts brought to the market
  - Proudly brewed in sunny Napoli, Italy ☺
- Focus on real-time multimedia applications
  - Web conferencing only, at first
  - Then widened the scope to multimedia in general
  - Strong perspective on standardization and open source
    - *WebRTC rulez!*
- Several activities
  - Consulting services
  - Commercial support & licenses
  - Streaming of live events (e.g., IETF, ACM SIGCOMM, ...)
  - Products (conferencing, webinar, ...)

(*Napoli looks a bit like this...)

Load Testing of SIP Infrastructures

- Typically, a few different objectives
  - How many SIP sessions can my service handle?
  - How many calls per second, and how many concurrent calls?
  - Does the number change when media is involved?
- Programmable and customizable are important requirements
  - Not all the calls are the same
  - Different applications, different scenarios
  - Custom headers/fields may be involved
- Several popular tools available for the purpose
  - SIPp: a SIP protocol test tool
    - https://github.com/SIPp/sipp
  - Seagull: an Open Source Multi-protocol traffic generator
    - http://gull.sourceforge.net/
  - Many other solutions (often proprietary though)

Load Testing of SIP Infrastructures

- Typically, a few different objectives
  - How many SIP sessions can my service handle?
  - How many calls per second, and how many concurrent calls?
  - Does the number change when media is involved?
- Programmable and customizable are important requirements
  - Not all the calls are the same
  - Different applications, different scenarios
  - Custom headers/fields may be involved
- Several popular tools available for the purpose
  - SIPp: a SIP protocol test tool
    - https://github.com/SIPp/sipp
  - Seagull: an Open Source Multi-protocol traffic generator
    - http://gull.sourceforge.net/
  - Many other solutions (often proprietary though)

Load Testing of SIP Infrastructures

- Typically, a few different objectives
  - How many SIP sessions can my service handle?
  - How many calls per second, and how many concurrent calls?
  - Does the number change when media is involved?
- Programmable and customizable are important requirements
  - Not all the calls are the same
  - Different applications, different scenarios
  - Custom headers/fields may be involved
- Several popular tools available for the purpose
  - SIPp: a SIP protocol test tool
    - https://github.com/SIPp/sipp
  - Seagull: an Open Source Multi-protocol traffic generator
    - http://gull.sourceforge.net/
  - Many other solutions (often proprietary though)

SIPp: a SIP protocol test tool

- Open source SIP traffic generator
  - https://github.com/SIPp/sipp
- De-facto standard tool for generating SIP traffic, with or without media
- XML files to design testing session
  - Expected sequence of SIP messages, and how to react
  - Variables for custom fields (e.g., addresses, usernames, etc.)
  - Tool comes with some pre-compiled scenarios (e.g., UAC vs. UAS)
- Several options to customize session, even in real-time
  - Call rate, period, overall number of calls, etc.
- Media can be sent as well
  - Replay of pre-captured pcap files (e.g., RTP content)

SIPp: a SIP protocol test tool

- Open source SIP traffic generator
  - https://github.com/SIPp/sipp
- De-facto standard tool for generating SIP traffic, with or without media
- XML files to design testing session
  - Expected sequence of SIP messages, and how to react
  - Variables for custom fields (e.g., addresses, usernames, etc.)
  - Tool comes with some pre-compiled scenarios (e.g., UAC vs. UAS)
- Several options to customize session, even in real-time
  - Call rate, period, overall number of calls, etc.
- Media can be sent as well
  - Replay of pre-captured pcap files (e.g., RTP content)

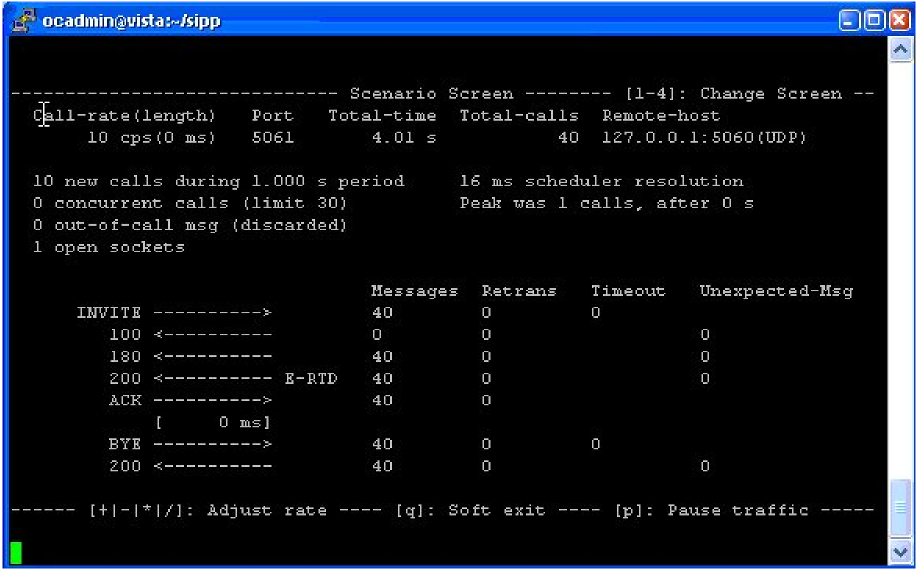
SIPp: a SIP protocol test tool

- Open source SIP traffic generator
  - https://github.com/SIPp/sipp
- De-facto standard tool for generating SIP traffic, with or without media
- XML files to design testing session
  - Expected sequence of SIP messages, and how to react
  - Variables for custom fields (e.g., addresses, usernames, etc.)
  - Tool comes with some pre-compiled scenarios (e.g., UAC vs. UAS)
- Several options to customize session, even in real-time
  - Call rate, period, overall number of calls, etc.
- Media can be sent as well
  - Replay of pre-captured pcap files (e.g., RTP content)

SIPp: a SIP protocol test tool

- Open source SIP traffic generator
  - https://github.com/SIPp/sipp
- De-facto standard tool for generating SIP traffic, with or without media
- XML files to design testing session
  - Expected sequence of SIP messages, and how to react
  - Variables for custom fields (e.g., addresses, usernames, etc.)
  - Tool comes with some pre-compiled scenarios (e.g., UAC vs. UAS)
- Several options to customize session, even in real-time
  - Call rate, period, overall number of calls, etc.
- Media can be sent as well
  - Replay of pre-captured pcap files (e.g., RTP content)

SIPp: a SIP protocol test tool




```
------------------------------ Scenario Screen -------- [1-4]: Change Screen --
  Call-rate(length)   Port   Total-time  Total-calls  Remote-host
       10 cps(0 ms)   5061       4.01 s           40  127.0.0.1:5060(UDP)

  10 new calls during 1.000 s period     16 ms scheduler resolution
  0 concurrent calls (limit 30)          Peak was 1 calls, after 0 s
  0 out-of-call msg (discarded)
  1 open sockets

                                Messages  Retrans   Timeout   Unexpected-Msg
      INVITE ---------->        40        0         0
         100 <----------        0         0                   0
         180 <----------        40        0                   0
         200 <---------- E-RTD  40        0                   0
         ACK ---------->        40        0
             [      0 ms]
         BYE ---------->        40        0         0
         200 <----------        40        0                   0

------ [+|-|*|/]: Adjust rate ---- [q]: Soft exit ---- [p]: Pause traffic -----
```

SIPp XML example: a UAC

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">

<scenario name="UAC␣with␣media">
  <send retrans="500">
        [..INVITE..]
  </send>

  <recv response="100" optional="true">
  </recv>

  <recv response="180" optional="true">
  </recv>

  <recv response="200" rtd="true" crlf="true">
  </recv>

  <send>
        [..ACK..]
  </send>

  <pause milliseconds="30000"/>

  <send retrans="500">
        [..BYE..]
  </send>

  <recv response="200" crlf="true">
  </recv>

</scenario>
```

SIPp XML example: a UAC

```xml
<send retrans="500">
  <![CDATA[

    INVITE sip:[service]@[remote_ip]:[remote_port] SIP/2.0
    Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
    From: pippozzo <sip:pippozzo@[local_ip]:[local_port]>;tag=[call_number]
    To: [service] <sip:[service]@[remote_ip]:[remote_port]>
    X-Custom-Header-ID: xyz
    Call-ID: [call_id]
    CSeq: 1 INVITE
    Contact: sip:pippozzo@[local_ip]:[local_port]
    Max-Forwards: 70
    Subject: AudioConf Test
    Content-Type: application/sdp
    Content-Length: [len]

    v=0
    o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]
    s=-
    c=IN IP[local_ip_type] [local_ip]
    t=0 0
    m=audio [auto_media_port] RTP/AVP 8
    a=rtpmap:8 PCMA/8000
    a=rtpmap:101 telephone-event/8000
    a=fmtp:101 0-11,16

  ]]>
</send>
```

SIPp XML example: a UAC

```
<!-- Pause 8 seconds, which is approximately the -->
<!-- duration of the PCAP file                   -->
<pause milliseconds="8000"/>

<!-- Play a pre-recorded PCAP file (RTP stream)  -->
<nop>
  <action>
    <exec play_pcap_audio="/usr/share/sipp/pcap/g711a.pcap"/>
  </action>
</nop>
```

Making things easier with SIPp

- SIPp is very powerful
  - ... but designing XML files can be a drag!
  - A couple of tools can help with that
- Sippy Cup
  - SIPp helper tool, written in Ruby
  - Can generate SIPp load test profiles and the corresponding pcap media
  - http://mojolingo.github.io/sippy_cup/
- What about a web frontend?
  - http://sipp.sourceforge.net/web_frontend/
  - https://github.com/mojolingo/SIPTreadmill (Sippy Cup authors)
  - https://github.com/SIPp/pysipp ("SIPp for Humans" ☺)

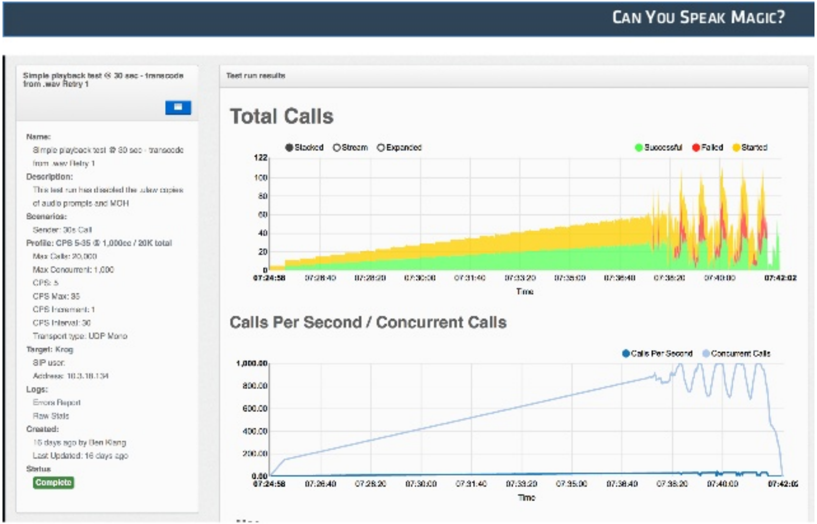
Making things easier with SIPp

- SIPp is very powerful
  - ... but designing XML files can be a drag!
  - A couple of tools can help with that
- Sippy Cup
  - SIPp helper tool, written in Ruby
  - Can generate SIPp load test profiles and the corresponding pcap media
  - http://mojolingo.github.io/sippy_cup/
- What about a web frontend?
  - http://sipp.sourceforge.net/web_frontend/
  - https://github.com/mojolingo/SIPTreadmill (Sippy Cup authors)
  - https://github.com/SIPp/pysipp ("SIPp for Humans" ☺)

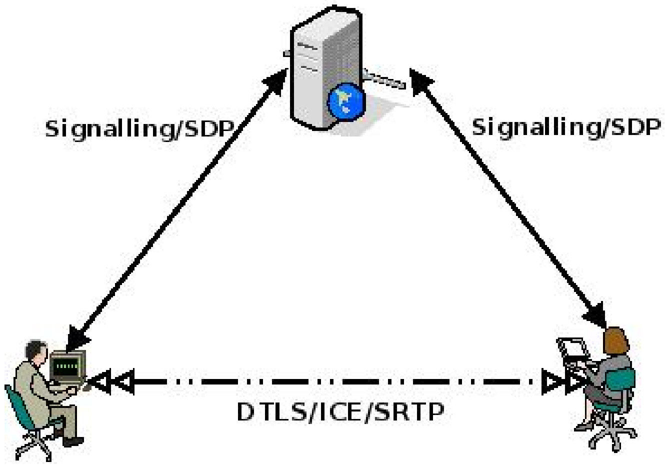
Making things easier with SIPp

- SIPp is very powerful
  - ... but designing XML files can be a drag!
  - A couple of tools can help with that
- Sippy Cup
  - SIPp helper tool, written in Ruby
  - Can generate SIPp load test profiles and the corresponding pcap media
  - http://mojolingo.github.io/sippy_cup/
- What about a web frontend?
  - http://sipp.sourceforge.net/web_frontend/
  - https://github.com/mojolingo/SIPTreadmill (Sippy Cup authors)
  - https://github.com/SIPp/pysipp ("SIPp for Humans" ☺)

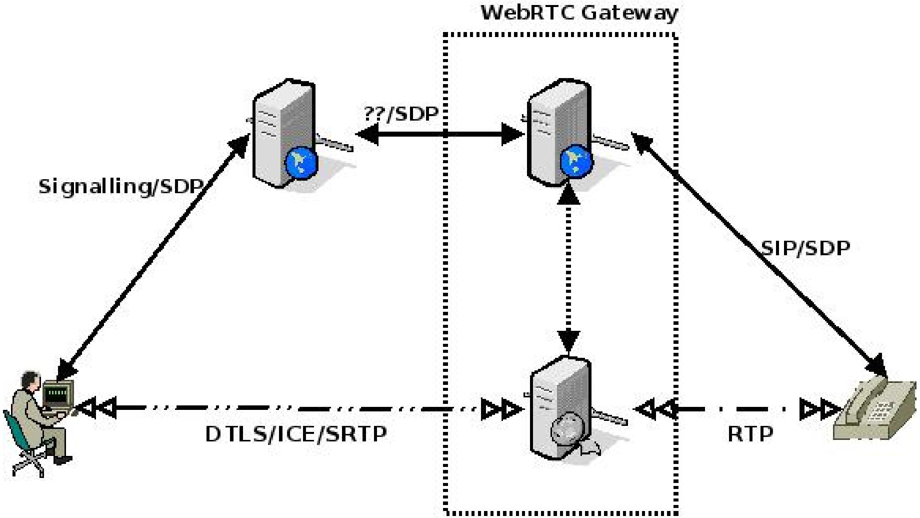
Web frontend example: SIP Treadmill



The problem: getting SIP and WebRTC to like each other



The problem: getting SIP and WebRTC to like each other



Bridging the gap: the WebRTC protocol suite

- Signalling (well, sort of) and Negotiation
  - Javascript Session Establishment Protocol (JSEP)
  - Session Description Protocol (SDP) adaptation
- Connection Establishment and NAT Traversal
  - Session Traversal Utilities for NAT (STUN)
  - Traversal Using Relay NAT (TURN)
  - Interactive Connectivity Establishment (ICE)
- Media Transport and Control
  - Real-time Transport (and Control) Protocol (RTP/RTCP)
  - Secure Extensions to RTP (SRTP)
  - Datagram Transport Layer Security (DTLS)
- Multimedia codecs
  - Opus audio codec (MTI, Mandatory-to-implement)
  - VP8 and H.264 video codecs (MTI, Mandatory-to-implement)
- Generic Data
  - WebRTC Data Channels (SCTP)

Bridging the gap: the WebRTC protocol suite

- Signalling (well, sort of) and Negotiation
  - Javascript Session Establishment Protocol (JSEP)
  - Session Description Protocol (SDP) adaptation
- Connection Establishment and NAT Traversal
  - Session Traversal Utilities for NAT (STUN)
  - Traversal Using Relay NAT (TURN)
  - Interactive Connectivity Establishment (ICE)
- Media Transport and Control
  - Real-time Transport (and Control) Protocol (RTP/RTCP)
  - Secure Extensions to RTP (SRTP)
  - Datagram Transport Layer Security (DTLS)
- Multimedia codecs
  - Opus audio codec (MTI, Mandatory-to-implement)
  - VP8 and H.264 video codecs (MTI, Mandatory-to-implement)
- Generic Data
  - WebRTC Data Channels (SCTP)

What makes SIP+WebRTC testing harder?

- First of all, simulating users is not trivial...
  - ... they're not just web users
  - ... they're not "regular" SIP clients either
- Approaches like SIPp wouldn't work ☹
  - They might work for the signalling part...
  - ... but pcap files would NOT mix well with ICE/DTLS/SRTP
  - WebRTC too dynamic in nature to just replay traffic
- Stressing tools for HTTP/WS exist, but they don't account for WebRTC
  - JMeter, Siege, HTTPerf etc. very good at generating HTTP traffic
  - ... but then how do you create PeerConnections and/or exchange media?
  - PhantomJS could handle dynamic application, but not the WebRTC part

What makes SIP+WebRTC testing harder?

- First of all, simulating users is not trivial...
  - ... they're not just web users
  - ... they're not "regular" SIP clients either
- Approaches like SIPp wouldn't work ☹
  - They might work for the signalling part...
  - ... but pcap files would NOT mix well with ICE/DTLS/SRTP
  - WebRTC too dynamic in nature to just replay traffic
- Stressing tools for HTTP/WS exist, but they don't account for WebRTC
  - JMeter, Siege, HTTPerf etc. very good at generating HTTP traffic
  - ... but then how do you create PeerConnections and/or exchange media?
  - PhantomJS could handle dynamic application, but not the WebRTC part

What makes SIP+WebRTC testing harder?

- First of all, simulating users is not trivial...
  - ... they're not just web users
  - ... they're not "regular" SIP clients either
- Approaches like SIPp wouldn't work ☹
  - They might work for the signalling part...
  - ... but pcap files would NOT mix well with ICE/DTLS/SRTP
  - WebRTC too dynamic in nature to just replay traffic
- Stressing tools for HTTP/WS exist, but they don't account for WebRTC
  - JMeter, Siege, HTTPerf etc. very good at generating HTTP traffic
  - ... but then how do you create PeerConnections and/or exchange media?
  - PhantomJS could handle dynamic application, but not the WebRTC part

Testing the “pane e puparuoli” way ☺

- The first approach that comes to mind: just open multiple tabs!
  - WebRTC applications are web applications
  - Unlike softphones, quite easy to open more tabs for more “calls”
  - Besides, JavaScript allows for easy programmability
    - e.g., custom page that generates multiple calls
- As it is, not very scalable of course...
  - You can only open so many tabs yourself
    - Local resources and/or browser connection limits may be exhausted
    - Fake devices can help, here, but only up to a point
  - Friends can help, but how many can you bother? ☺
- Sounds very silly and naive (and it probably is)
  - ... but as we’ll see in a minute, not that much!
  - (what if this could be automated and “remotized”?)

Testing the “pane e puparuoli” way ☺

- The first approach that comes to mind: just open multiple tabs!
  - WebRTC applications are web applications
  - Unlike softphones, quite easy to open more tabs for more “calls”
  - Besides, JavaScript allows for easy programmability
    - e.g., custom page that generates multiple calls
- As it is, not very scalable of course...
  - You can only open so many tabs yourself
    - Local resources and/or browser connection limits may be exhausted
    - Fake devices can help, here, but only up to a point
  - Friends can help, but how many can you bother? ☺
- Sounds very silly and naive (and it probably is)
  - ... but as we’ll see in a minute, not that much!
  - (what if this could be automated and “remotized”?)

Testing the “pane e puparuoli” way ☺

- The first approach that comes to mind: just open multiple tabs!
  - WebRTC applications are web applications
  - Unlike softphones, quite easy to open more tabs for more “calls”
  - Besides, JavaScript allows for easy programmability
    - e.g., custom page that generates multiple calls
- As it is, not very scalable of course...
  - You can only open so many tabs yourself
    - Local resources and/or browser connection limits may be exhausted
    - Fake devices can help, here, but only up to a point
  - Friends can help, but how many can you bother? ☺
- Sounds very silly and naive (and it probably is)
  - ... but as we’ll see in a minute, not that much!
  - (what if this could be automated and “remotized”?)

SeleniumHQ – Web Browser Automation

- Opening tons of browsers/tabs locally and manually doesn't scale well...

SeleniumHQ

- http://www.seleniumhq.org/

- Tool for browser *automation*
  - Web applications are scriptable by definition (and can collect stats)
  - Remotely controlled browser instances to open those
  - Scenario can be programmed in different languages (e.g., Java, Python, etc.)
- Uses WebDrivers for different browsers
  - *Fake* devices help automate media capture (and permissions)
  - Tricks exist to start browsers "headless" too (e.g., Xvfb)
    - Side note: will Headless Chromium help even more here?
- Foundation for some commercial services too (e.g., testRTC)

SeleniumHQ – Web Browser Automation

- Opening tons of browsers/tabs locally and manually doesn't scale well...

SeleniumHQ

- http://www.seleniumhq.org/

- Tool for browser *automation*
  - Web applications are scriptable by definition (and can collect stats)
  - Remotely controlled browser instances to open those
  - Scenario can be programmed in different languages (e.g., Java, Python, etc.)
- Uses WebDrivers for different browsers
  - *Fake* devices help automate media capture (and permissions)
  - Tricks exist to start browsers "headless" too (e.g., Xvfb)
    - Side note: will Headless Chromium help even more here?
- Foundation for some commercial services too (e.g., testRTC)

SeleniumHQ – Web Browser Automation

- Opening tons of browsers/tabs locally and manually doesn't scale well...

SeleniumHQ

- http://www.seleniumhq.org/

- Tool for browser *automation*
  - Web applications are scriptable by definition (and can collect stats)
  - Remotely controlled browser instances to open those
  - Scenario can be programmed in different languages (e.g., Java, Python, etc.)
- Uses WebDrivers for different browsers
  - *Fake* devices help automate media capture (and permissions)
  - Tricks exist to start browsers "headless" too (e.g., Xvfb)
    - Side note: will Headless Chromium help even more here?
- Foundation for some commercial services too (e.g., testRTC)

SeleniumHQ – Web Browser Automation

- Opening tons of browsers/tabs locally and manually doesn't scale well...

SeleniumHQ

- http://www.seleniumhq.org/

- Tool for browser *automation*
  - Web applications are scriptable by definition (and can collect stats)
  - Remotely controlled browser instances to open those
  - Scenario can be programmed in different languages (e.g., Java, Python, etc.)
- Uses WebDrivers for different browsers
  - *Fake* devices help automate media capture (and permissions)
  - Tricks exist to start browsers "headless" too (e.g., Xvfb)
    - Side note: will Headless Chromium help even more here?
- Foundation for some commercial services too (e.g., testRTC)

SeleniumHQ – Web Browser Automation

- Opening tons of browsers/tabs locally and manually doesn't scale well...

SeleniumHQ

- http://www.seleniumhq.org/

- Tool for browser *automation*
  - Web applications are scriptable by definition (and can collect stats)
  - Remotely controlled browser instances to open those
  - Scenario can be programmed in different languages (e.g., Java, Python, etc.)
- Uses WebDrivers for different browsers
  - *Fake* devices help automate media capture (and permissions)
  - Tricks exist to start browsers "headless" too (e.g., Xvfb)
    - Side note: will Headless Chromium help even more here?
- Foundation for some commercial services too (e.g., testRTC)

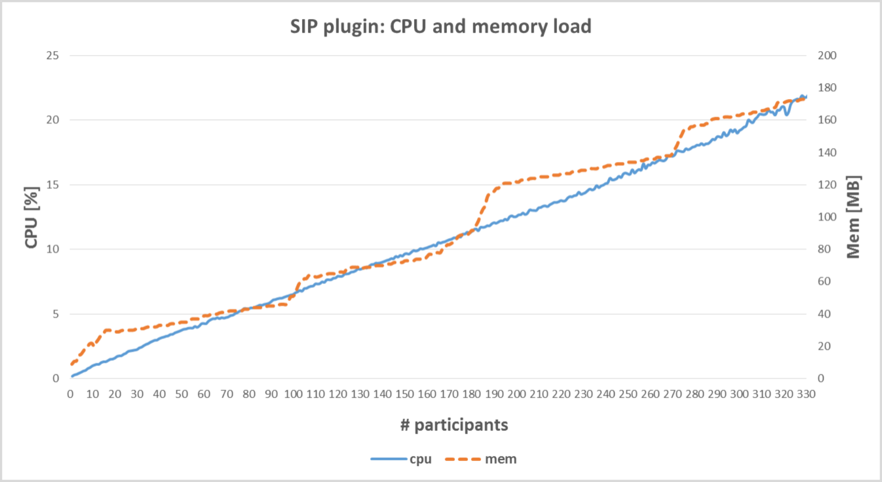
SeleniumHQ: a sample orchetrator in JavaScript

```
var webdriver = require('selenium-webdriver'),
    By = webdriver.By,
    until = webdriver.until;

var chrome = require('selenium-webdriver/chrome');
var options = new chrome.Options();
options.addArguments('--use-fake-device-for-media-stream', '--use-fake-ui-for-media-stream');

var driver = new webdriver.Builder()
    .forBrowser('chrome')
        .withCapabilities(options.toCapabilities())
    .build();

driver.get('http://localhost:8000/siptest.html');
driver.findElement(By.id('start')).click();
driver.findElement(By.id('username')).sendKeys('sip:janususer@localhost');
driver.findElement(By.id('password')).sendKeys('januspwd');
driver.findElement(By.id('registerset')).click();
driver.findElement(By.id('secret')).click();
driver.wait(until.elementLocated(By.className('bootbox')), 10000);
driver.findElement(By.css('button[data-bb-handler='ok']')).click();
driver.findElement(By.id('register')).click();
driver.wait(until.elementLocated(By.className('bootbox')), 10000);
driver.findElement(By.css('button[data-bb-handler='confirm']')).click();
driver.findElement(By.id('peer')).sendKeys('sip:600@localhost');
driver.findElement(By.id('call')).click();

driver.sleep(10000);    // Wait 10s and then hangup
driver.findElement(By.id('call')).click();

driver.quit();
```

SeleniumHQ: grid of controlled browsers

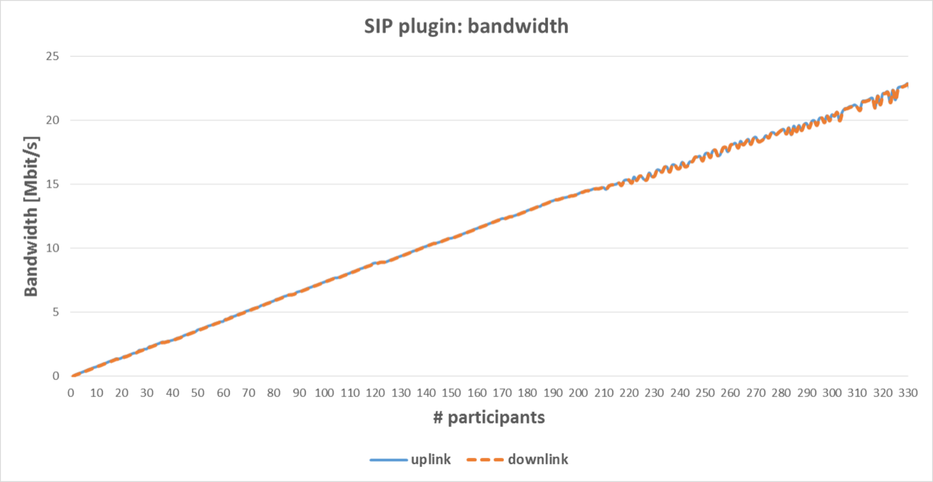
https://github.com/SeleniumHQ/selenium/wiki/Grid2

A SeleniumHQ story: testing the Janus SIP plugin



"Performance analysis of the Janus WebRTC gateway" (AWeS '15)
http://dl.acm.org/citation.cfm?doid=2749215.2749223

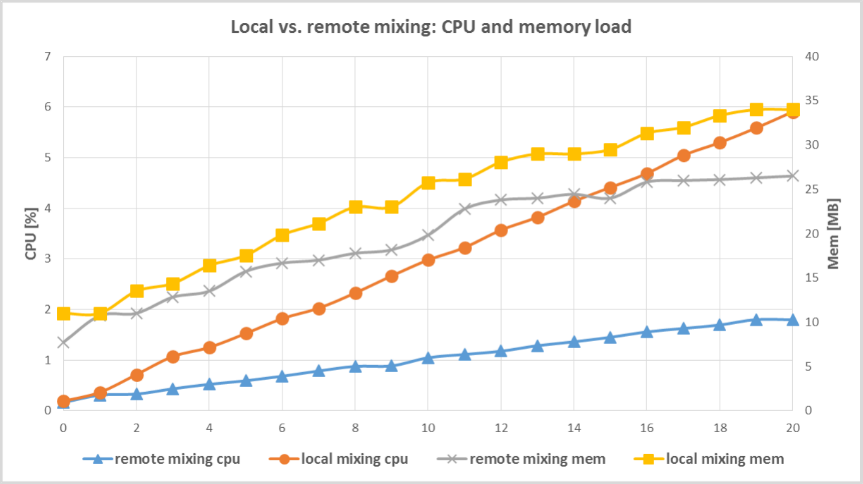
A SeleniumHQ story: testing the Janus SIP plugin



"Performance analysis of the Janus WebRTC gateway" (AWeS '15)

http://dl.acm.org/citation.cfm?doid=2749215.2749223

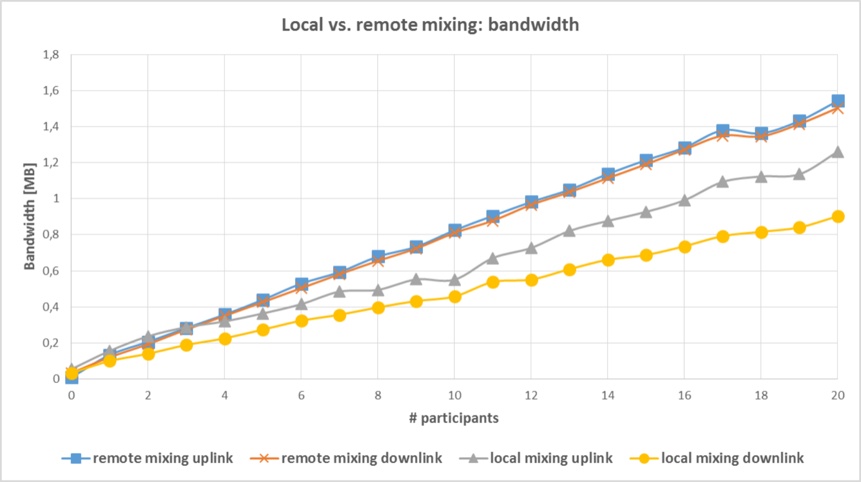
A SeleniumHQ story: testing the Janus SIP plugin



"Performance analysis of the Janus WebRTC gateway" (AWeS '15)

http://dl.acm.org/citation.cfm?doid=2749215.2749223

A SeleniumHQ story: testing the Janus SIP plugin



"Performance analysis of the Janus WebRTC gateway" (AWeS '15)
http://dl.acm.org/citation.cfm?doid=2749215.2749223

What about native tools?

- While a great improvement over "manual" testing, Selenium is still heavy...
  - Even "headless", a complete browser adds a lot of overhead
  - Limit on number of instances a single server can handle
  - Less scalability if number of servers is limited
- If UI doesn't matter, a "native" tool would be much more efficient
  - Something focused on PeerConnections, and not the rest
  - Should be controllable and programmable
  - ... and should be able to send/receive media as well!
- While some tools exist, most are very specific to target applications
  - e.g., Jitsi-Hammer, a nice traffic generator for Jitsi Videobridge
- In theory, Chrome WebRTC stack could be used for general purpose tool

What about native tools?

- While a great improvement over “manual” testing, Selenium is still heavy...
  - Even “headless”, a complete browser adds a lot of overhead
  - Limit on number of instances a single server can handle
  - Less scalability if number of servers is limited
- If UI doesn’t matter, a “native” tool would be much more efficient
  - Something focused on PeerConnections, and not the rest
  - Should be controllable and programmable
  - ... and should be able to send/receive media as well!
- While some tools exist, most are very specific to target applications
  - e.g., Jitsi-Hammer, a nice traffic generator for Jitsi Videobridge
- In theory, Chrome WebRTC stack could be used for general purpose tool

What about native tools?

- While a great improvement over "manual" testing, Selenium is still heavy...
  - Even "headless", a complete browser adds a lot of overhead
  - Limit on number of instances a single server can handle
  - Less scalability if number of servers is limited
- If UI doesn't matter, a "native" tool would be much more efficient
  - Something focused on PeerConnections, and not the rest
  - Should be controllable and programmable
  - ... and should be able to send/receive media as well!
- While some tools exist, most are very specific to target applications
  - e.g., Jitsi-Hammer, a nice traffic generator for Jitsi Videobridge
- In theory, Chrome WebRTC stack could be used for general purpose tool

How we solved this for our needs: Jattack

- Jattack = Janus Attack (or *J'attaque* ☺)
  - Controllable (via WebSockets) WebRTC client stack
  - Allows dynamic creation/monitoring of tons of WebRTC PeerConnections
- Still improving it, but already in a usable state
  - Can receive (and optionally record) media
  - Can also send media, if you provide an RTP source yourself
  - Live events on anything that happens
- Originally conceived as a homemade tool for stressing Janus
  - Might actually be used with other services as well
  - All the logic (signalling included) is in the controller
- Presented at IPTComm in Chicago just a few months ago
  - https://prezi.com/krg1esxoa6ug/jattack/
  - https://www.youtube.com/watch?v=UwNq8p0m1js

How we solved this for our needs: Jattack

- Jattack = Janus Attack (or *J'attaque* ☺)
  - Controllable (via WebSockets) WebRTC client stack
  - Allows dynamic creation/monitoring of tons of WebRTC PeerConnections
- Still improving it, but already in a usable state
  - Can receive (and optionally record) media
  - Can also send media, if you provide an RTP source yourself
  - Live events on anything that happens
- Originally conceived as a homemade tool for stressing Janus
  - Might actually be used with other services as well
  - All the logic (signalling included) is in the controller
- Presented at IPTComm in Chicago just a few months ago
  - https://prezi.com/krg1esxoa6ug/jattack/
  - https://www.youtube.com/watch?v=UwNq8p0m1js

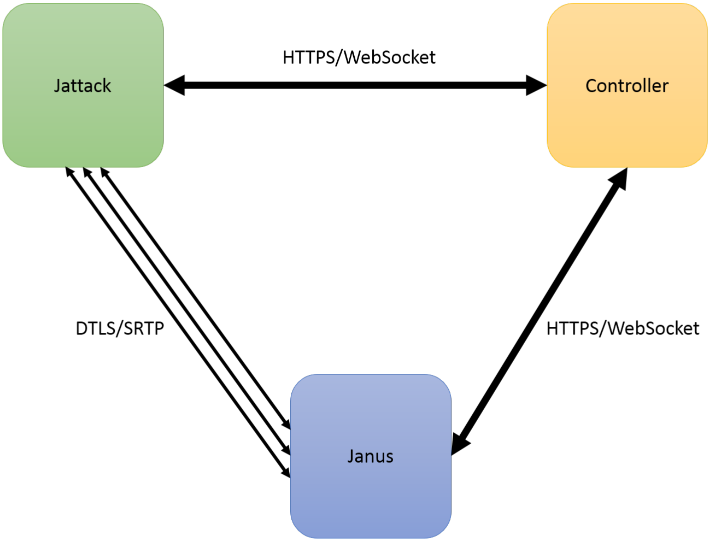
How we solved this for our needs: Jattack

- Jattack = Janus Attack (or *J'attaque* ☺)
  - Controllable (via WebSockets) WebRTC client stack
  - Allows dynamic creation/monitoring of tons of WebRTC PeerConnections
- Still improving it, but already in a usable state
  - Can receive (and optionally record) media
  - Can also send media, if you provide an RTP source yourself
  - Live events on anything that happens
- Originally conceived as a homemade tool for stressing Janus
  - Might actually be used with other services as well
  - All the logic (signalling included) is in the controller
- Presented at IPTComm in Chicago just a few months ago
  - https://prezi.com/krg1esxoa6ug/jattack/
  - https://www.youtube.com/watch?v=UwNq8p0m1js

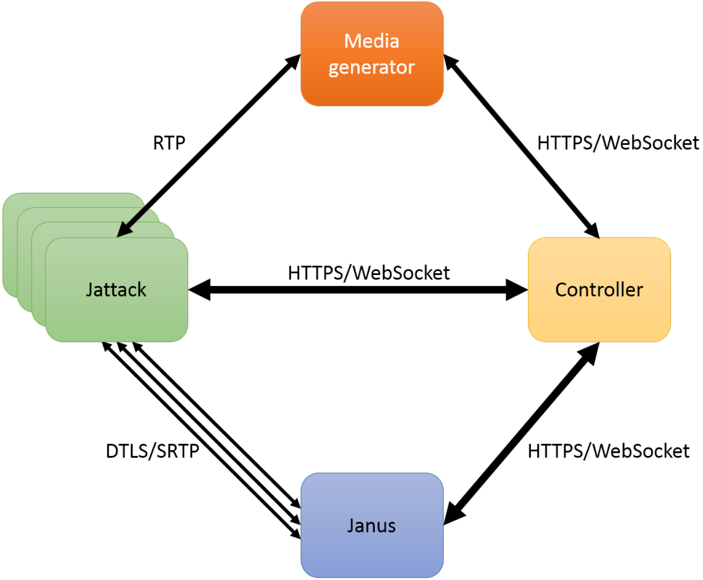
How we solved this for our needs: Jattack

- Jattack = Janus Attack (or *J'attaque* ☺)
  - Controllable (via WebSockets) WebRTC client stack
  - Allows dynamic creation/monitoring of tons of WebRTC PeerConnections
- Still improving it, but already in a usable state
  - Can receive (and optionally record) media
  - Can also send media, if you provide an RTP source yourself
  - Live events on anything that happens
- Originally conceived as a homemade tool for stressing Janus
  - Might actually be used with other services as well
  - All the logic (signalling included) is in the controller
- Presented at IPTComm in Chicago just a few months ago
  - https://prezi.com/krg1esxoa6ug/jattack/
  - https://www.youtube.com/watch?v=UwNq8p0m1js

The Jattack architecture



The Jattack architecture



Using Jattack with SIP/WebRTC Infrastructures

- A bit more complicated than Selenium...
  - You can't simply "re-use" web pages
  - Signalling needs to be implemented in a "Controller"
- ... but way lighter!
  - Based on the WebRTC core of Janus itself
  - Only does WebSockets communication and PeerConnections
  - A single machine can simulate many more users
- Most of the burden is designing the controller
  - Accept connections from one or more Jattack instances
  - Implementing the signalling (e.g., SIP over WS, Janus API, etc.)
  - Bridging SDP and candidates between server and Jattack
  - Intercepting events and handling them (e.g., dump to DB)

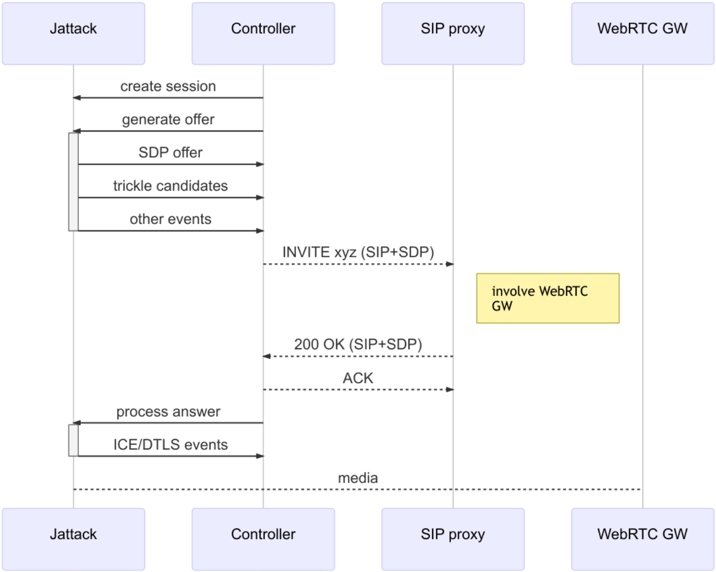
Using Jattack with SIP/WebRTC Infrastructures

- A bit more complicated than Selenium...
  - You can't simply "re-use" web pages
  - Signalling needs to be implemented in a "Controller"
- ... but way lighter!
  - Based on the WebRTC core of Janus itself
  - Only does WebSockets communication and PeerConnections
  - A single machine can simulate many more users
- Most of the burden is designing the controller
  - Accept connections from one or more Jattack instances
  - Implementing the signalling (e.g., SIP over WS, Janus API, etc.)
  - Bridging SDP and candidates between server and Jattack
  - Intercepting events and handling them (e.g., dump to DB)

Using Jattack with SIP/WebRTC Infrastructures

- A bit more complicated than Selenium...
  - You can't simply "re-use" web pages
  - Signalling needs to be implemented in a "Controller"
- ... but way lighter!
  - Based on the WebRTC core of Janus itself
  - Only does WebSockets communication and PeerConnections
  - A single machine can simulate many more users
- Most of the burden is designing the controller
  - Accept connections from one or more Jattack instances
  - Implementing the signalling (e.g., SIP over WS, Janus API, etc.)
  - Bridging SDP and candidates between server and Jattack
  - Intercepting events and handling them (e.g., dump to DB)

Jattack: a sample sequence diagram ("repeat N times")



What about other native tools?

- Unfortunately, Jattack isn't currently open source ☹
- Anyway, as we said, Chrome WebRTC stack could be used for this
  - Library would provide WebRTC stack, custom control on top
- A couple of projects to keep track of
  - https://github.com/js-platform/node-webrtc
  - https://github.com/vmolsa/webrtc-native
- Looks like they could be a good candidate
  - Both aim at providing WebRTC stack controllable by a node.js application
  - Not sure what the status for them is, though?
- Once available, the same steps described for Jattack could apply
  - e.g., Sequence Diagram or Scenario Topology

What about other native tools?

- Unfortunately, Jattack isn't currently open source ☹
- Anyway, as we said, Chrome WebRTC stack could be used for this
  - Library would provide WebRTC stack, custom control on top
- A couple of projects to keep track of
  - https://github.com/js-platform/node-webrtc
  - https://github.com/vmolsa/webrtc-native
- Looks like they could be a good candidate
  - Both aim at providing WebRTC stack controllable by a node.js application
  - Not sure what the status for them is, though?
- Once available, the same steps described for Jattack could apply
  - e.g., Sequence Diagram or Scenario Topology

What about other native tools?

- Unfortunately, Jattack isn't currently open source ☹
- Anyway, as we said, Chrome WebRTC stack could be used for this
  - Library would provide WebRTC stack, custom control on top
- A couple of projects to keep track of
  - https://github.com/js-platform/node-webrtc
  - https://github.com/vmolsa/webrtc-native
- Looks like they could be a good candidate
  - Both aim at providing WebRTC stack controllable by a node.js application
  - Not sure what the status for them is, though?
- Once available, the same steps described for Jattack could apply
  - e.g., Sequence Diagram or Scenario Topology

What about other native tools?

- Unfortunately, Jattack isn't currently open source ☹
- Anyway, as we said, Chrome WebRTC stack could be used for this
  - Library would provide WebRTC stack, custom control on top
- A couple of projects to keep track of
  - https://github.com/js-platform/node-webrtc
  - https://github.com/vmolsa/webrtc-native
- Looks like they could be a good candidate
  - Both aim at providing WebRTC stack controllable by a node.js application
  - Not sure what the status for them is, though?
- Once available, the same steps described for Jattack could apply
  - e.g., Sequence Diagram or Scenario Topology

A final consideration: can servers instead help here?

- We've talked about clients so far, but what about servers?
  - Asterisk, Freeswitch, RTC:engine, Janus, etc., all support WebRTC
  - What if we used those to "stress" other servers?
- Approach could be hybrid, e.g.:
  1. Have Asterisk #1 generate a call via AMI, and get the WebRTC SDP
  2. Use WebRTC SDP for call to server X
  3. Pass WebRTC answer from server X back to Asterisk
  4. Repeat N times, involving multiple Asterisk instances
- Basically a "controller" acting as a bridge for signalling alone
  - Media would be provided by the servers themselves
- Less straightforward, and less flexible, but still a knife in your belt
  - ... especially if you're quite familiar with a particular server!

A final consideration: can servers instead help here?

- We've talked about clients so far, but what about servers?
  - Asterisk, Freeswitch, RTC:engine, Janus, etc., all support WebRTC
  - What if we used those to "stress" other servers?
- Approach could be hybrid, e.g.:
  1. Have Asterisk #1 generate a call via AMI, and get the WebRTC SDP
  2. Use WebRTC SDP for call to server X
  3. Pass WebRTC answer from server X back to Asterisk
  4. Repeat N times, involving multiple Asterisk instances
- Basically a "controller" acting as a bridge for signalling alone
  - Media would be provided by the servers themselves
- Less straightforward, and less flexible, but still a knife in your belt
  - ... especially if you're quite familiar with a particular server!

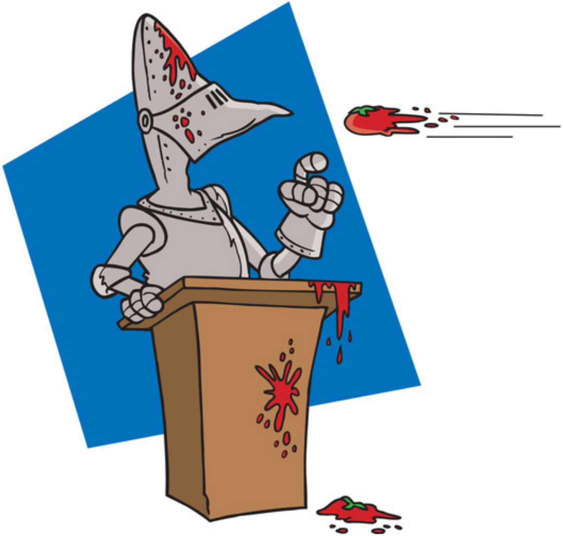
A final consideration: can servers instead help here?

- We've talked about clients so far, but what about servers?
  - Asterisk, Freeswitch, RTC:engine, Janus, etc., all support WebRTC
  - What if we used those to "stress" other servers?
- Approach could be hybrid, e.g.:
  1. Have Asterisk #1 generate a call via AMI, and get the WebRTC SDP
  2. Use WebRTC SDP for call to server X
  3. Pass WebRTC answer from server X back to Asterisk
  4. Repeat N times, involving multiple Asterisk instances
- Basically a "controller" acting as a bridge for signalling alone
  - Media would be provided by the servers themselves
- Less straightforward, and less flexible, but still a knife in your belt
  - ... especially if you're quite familiar with a particular server!

A final consideration: can servers instead help here?

- We've talked about clients so far, but what about servers?
  - Asterisk, Freeswitch, RTC:engine, Janus, etc., all support WebRTC
  - What if we used those to "stress" other servers?
- Approach could be hybrid, e.g.:
  1. Have Asterisk #1 generate a call via AMI, and get the WebRTC SDP
  2. Use WebRTC SDP for call to server X
  3. Pass WebRTC answer from server X back to Asterisk
  4. Repeat N times, involving multiple Asterisk instances
- Basically a "controller" acting as a bridge for signalling alone
  - Media would be provided by the servers themselves
- Less straightforward, and less flexible, but still a knife in your belt
  - ... especially if you're quite familiar with a particular server!

Questions? Comments?



https://twitter.com/elminiero