In the Trenches of a Globally Spanning SIP Network

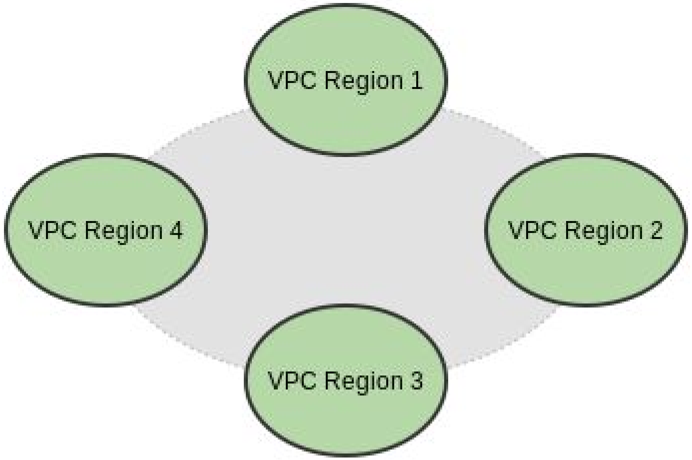
& the days spent firefighting

@borjessonjonas

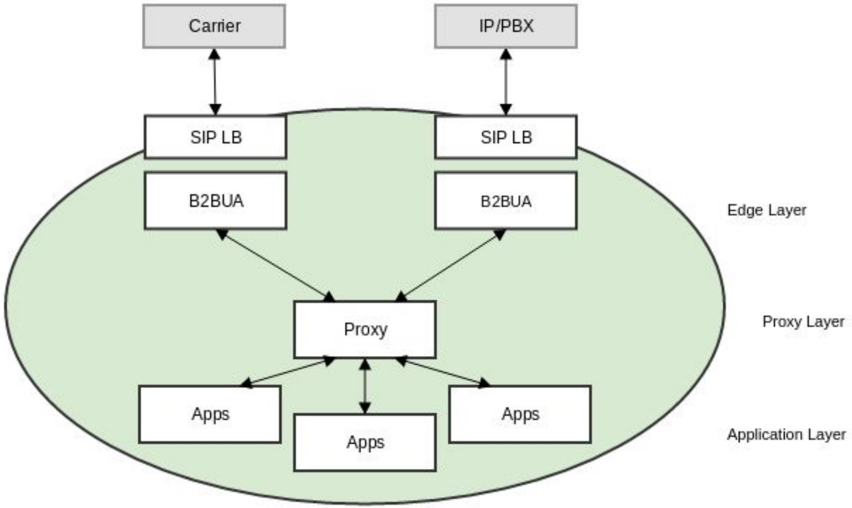
Hi, my name is Jonas

- Our SIP Network at a glance
- Loops
- Failover strategies
- Connection Management
- Registration
- Misc

- Deployed across the world
- 6 different AWS regions



- Each region deployed across multiple AZ
- Each region looks the same



Those SIP Load balancers = opensips

- Philosophy: simple & fast
    - Transaction Stateful
    - No external lookups
    - “Border Police” implemented by the B2BUA

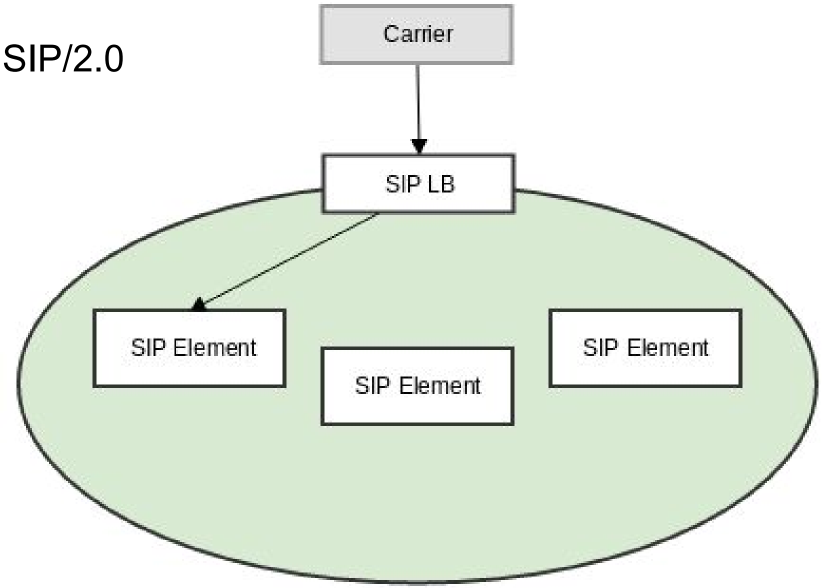
Max-Forwards:

- TTL
- Typical default 70
- Are you resetting it?

Basic SIP Load balancer

INVITE sip:hello@example.com SIP/2.0
...

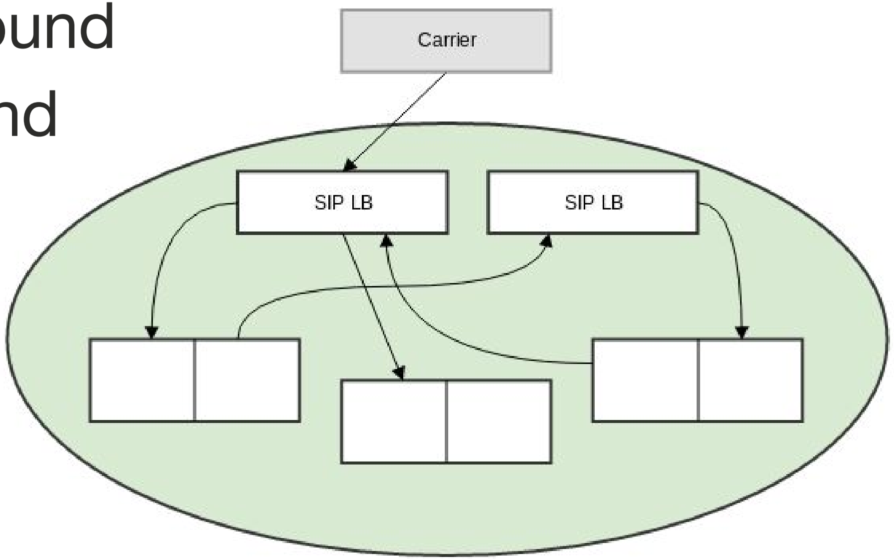
INVITE sip:hello@example.com SIP/2.0
…
Route: <sip:192.168.0.100;transport=udp;lr>



Bug in downstream element

- SIP LB marked call inbound
- B2BUA marked call outbound
- SIP LB marked call inbound

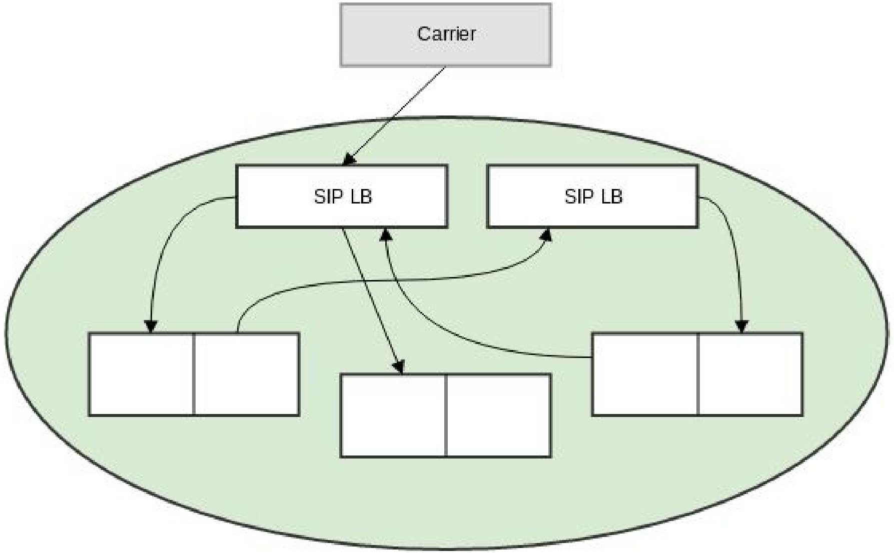
...



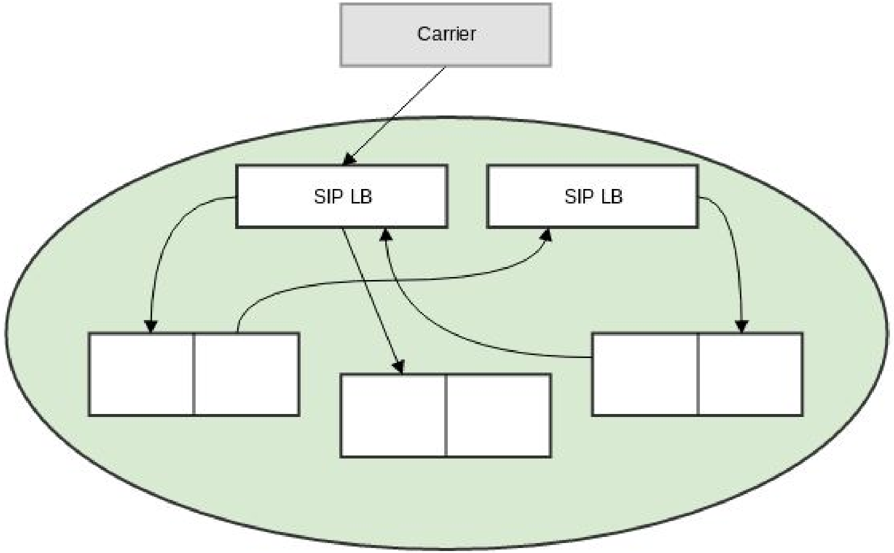
Malicious attacks

INVITE sip:hello@example.com SIP/2.0
…
Route: <sip:sip_lb_1;lr>
Route: <sip:sip_lb_2;lr>

INVITE sip:hello@example.com SIP/2.0
…
Route: <sip:b2bua_no_14;lr>
Route: <sip:sip_lb_2;lr>

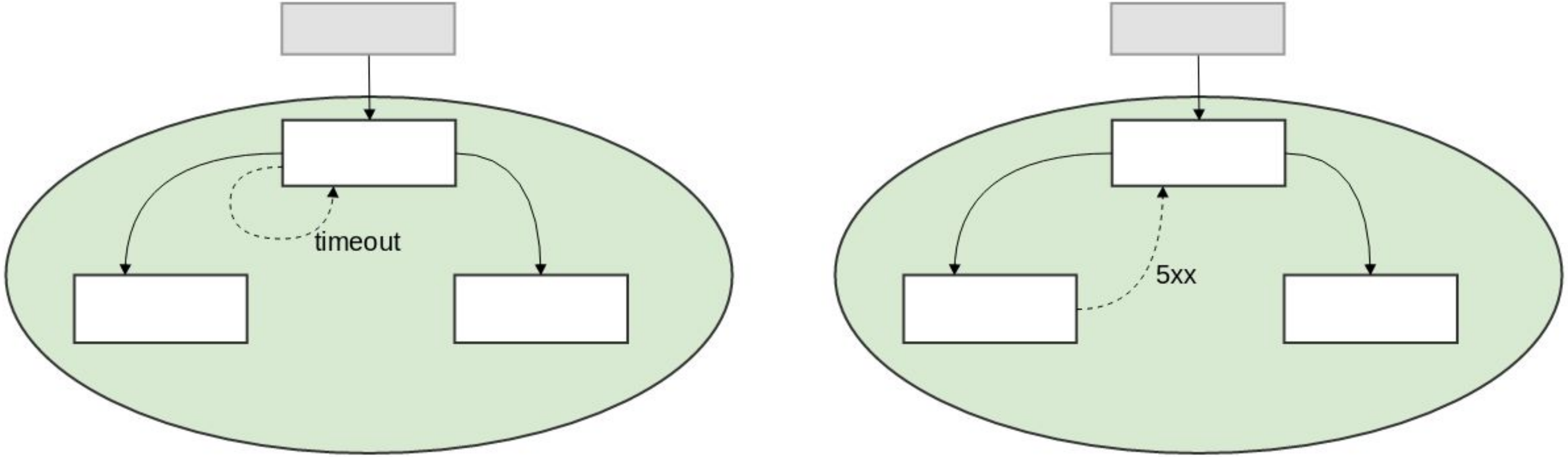


Don’t trust Route headers from external entities!



Node crash/misbehave

- Goal: need to handle it
- Strategy: Let's failover to next

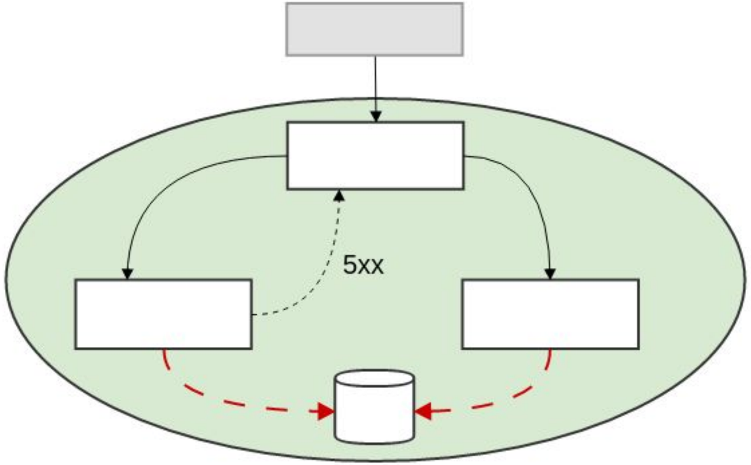


Scenario: Shared Resource is the culprit

- SQL
- NoSQL
- REST Service
- Another SIP Element
- Network partition

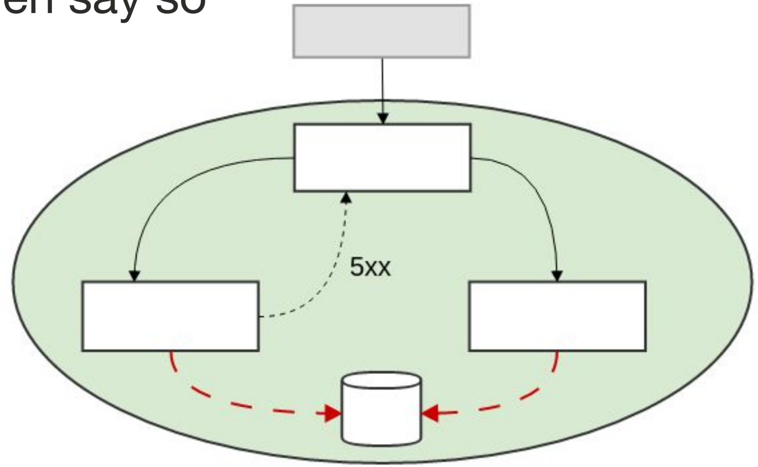
…

What do you think about our failover strategy now?



Be mindful when you failover

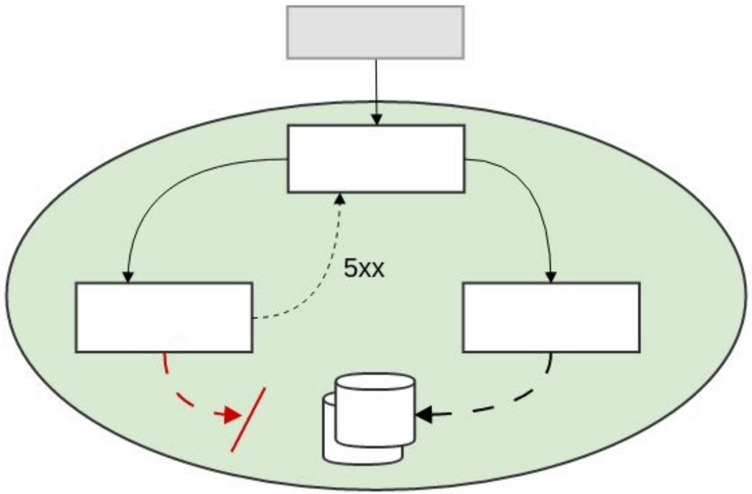
- Strategy:
    - The node closest to the problem spot probably knows best
    - Each node is responsible for exhaustively trying all options
    - If a node fails to contact XXX, then say so
    - A node will only failover on
        - 408
        - 503
        - ...



Drawback: network partition/bad network card

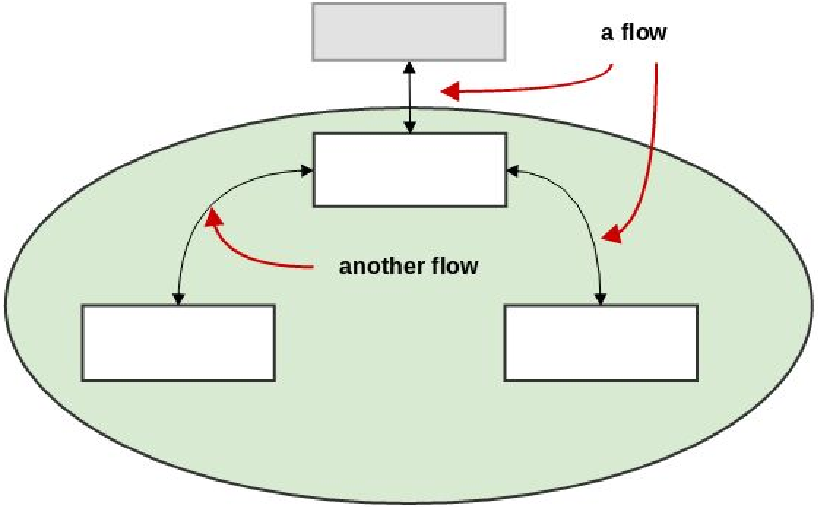
- Node is experiencing a localized network issue
- Node will signal to upstream that nope, no can do. Don’t failover
- In this case, it would have helped.

Conclusion: you have to figure out what is worse for you.



A flow:

- Represents a bidirectional communication path between two endpoints.
- Identified by: Local IP:Port + Remote IP:Port + Transport
- Has to be maintained
  - NAT & FW
  - Sending keep-alive traffic



Registration

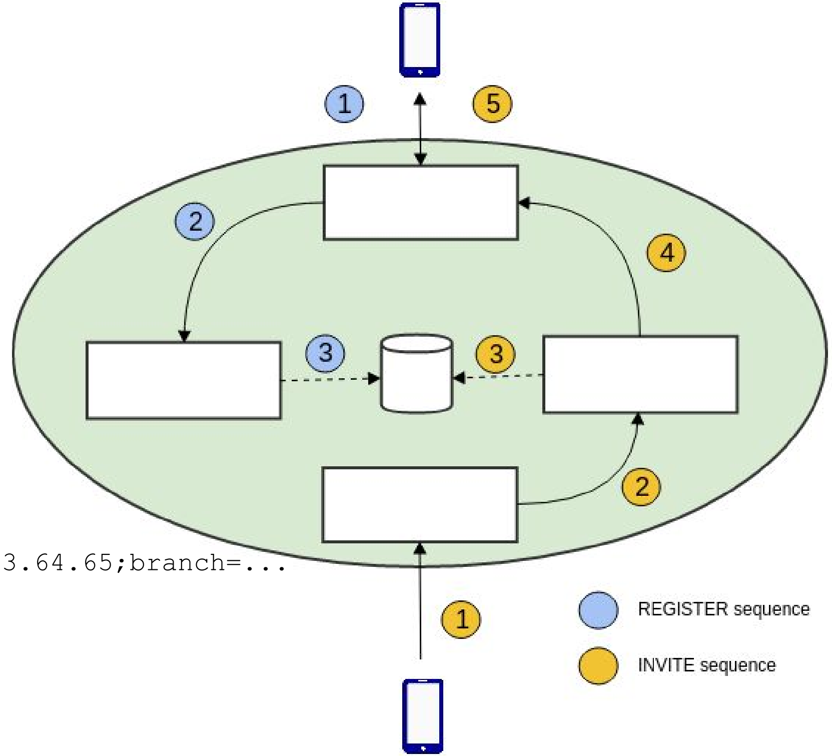
- What are we trying to achieve?

1

```
REGISTER sip:example.com SIP/2.0
...
Via: SIP/2.0/TCP 192.168.0.100;rport;branch=...
Contact: <sip:alice@192.168.0.100;transport=tcp;lr>
```

2

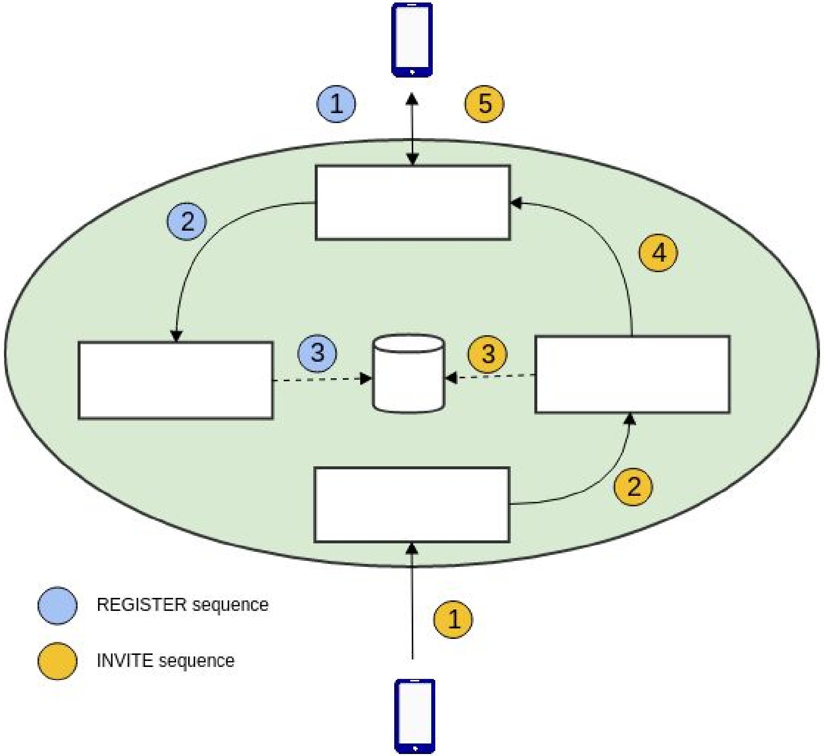
```
REGISTER sip:example.com SIP/2.0
...
Via: SIP/2.0/TCP 192.168.0.100;rport=5678;received=62.63.64.65;branch=...
Contact: <sip:alice@192.168.0.100;transport=tcp;lr>
Path: <sip:flow_token@10.36.10.11;transport=udp;lr;ob>
```



Registration

- No RFC 5626 support in opensips
- Can use force_tcp_alias
  - but: tcpconn_add_alias:
    possible port hijack attempt

  - what's going on?

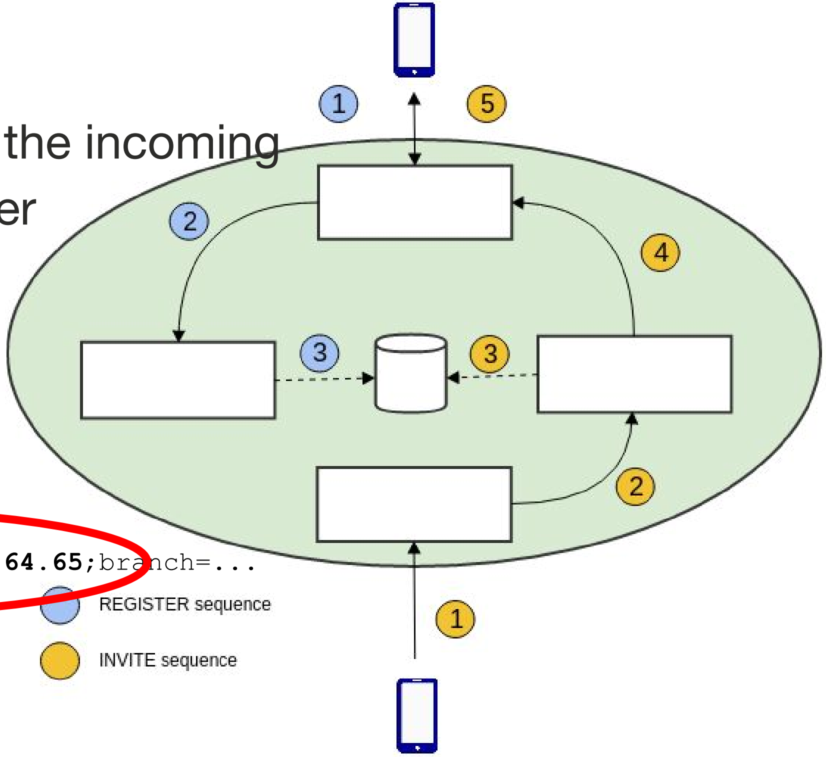


Registration

- force_tcp_alias will store the connection under a key computed from the src IP of the incoming connection + port found on the Via-header

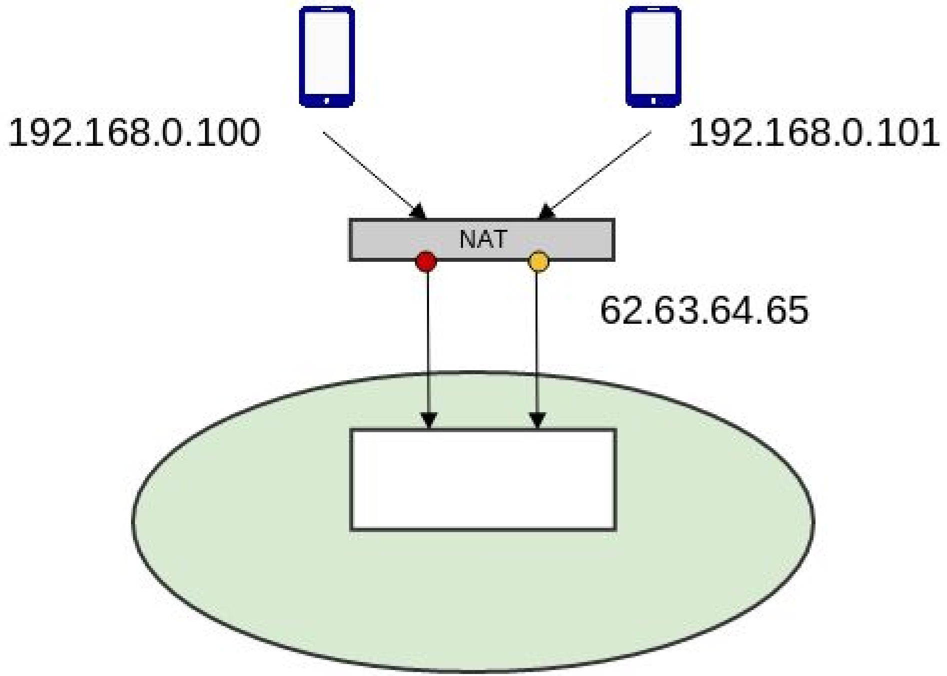
1
```
REGISTER sip:example.com SIP/2.0
...
Via: SIP/2.0/TCP 192.168.0.100;rport;branch=...
Contact: <sip:alice@192.168.0.100;transport=tcp;lr>
```

2
```
REGISTER sip:example.com SIP/2.0
...
Via: SIP/2.0/TCP 192.168.0.100;rport=5678;received=62.63.64.65;branch=...
Contact: <sip:alice@192.168.0.100;transport=tcp;lr>
Path: <sip:flow_token@10.36.10.11;transport=udp;lr;ob>
```



Registration

- Users behind same NAT => same “tcp_alias”
- First wins



- Note: you must also “fix” the Contact header.

Maintain the Flow

- Keep-alive traffic
  - Use TCP Keep alive
  - But, doesn’t always work. AT&T e.g. will kill your connection anyway
  - Client has to send a payload (double CRLF) as ping

- iOS App
  - What happens when it is backgrounded?
  - Conclusion: server has to send double CRLF

(opensips ask: please add server side double CRLF support)

Registration Flood

- Huge issue, it's the perfect DDoS
- Client has to play along
  - Honor 302 on REGISTER (not standard, we do this for our clients. Facilitates connection migration)
  - If lose connection, wait randomly between 0 - X seconds
  - Honor Retry-After headers (server could send a 500 e.g)
  - Re-cycle the connection after X hours
- Server side:
  - Limit the total no of connections/ip
  - Throttle incoming connections
  - Add ability to issue 302s
  - Retry-After headers

Fragmentation

- Are you building a SIP based smartphone app?
- Do you use ICE?
- Are your SDPs massive?
- Using TCP? (I certainly hope you are!)

- tcp_max_msg_chunks

TCP Connection Timeout

- Different in 1.11 than 1.8

Monitor Everything

- opensips
  - we log all requests and responses in a parsable friendly manner
  - pub those stats to dashboards

Carrier Edge | US	Carrier Edge | EU	Carrier Edge | BR
XYZ / s	XYZ / s	XYZ / s

Carrier Edge | US	Carrier Edge | EU	Carrier Edge | BR
0.62	0.42	0.22

Capture Everything

- tshark
- voipmonitor
- we push to S3
- tools for downloading, parsing and drawing (heavily use pkts.io)

KEEP THINGS SIMPLE

Thanks!