

Cross-Layer Telemetry Support in Linux Kernel

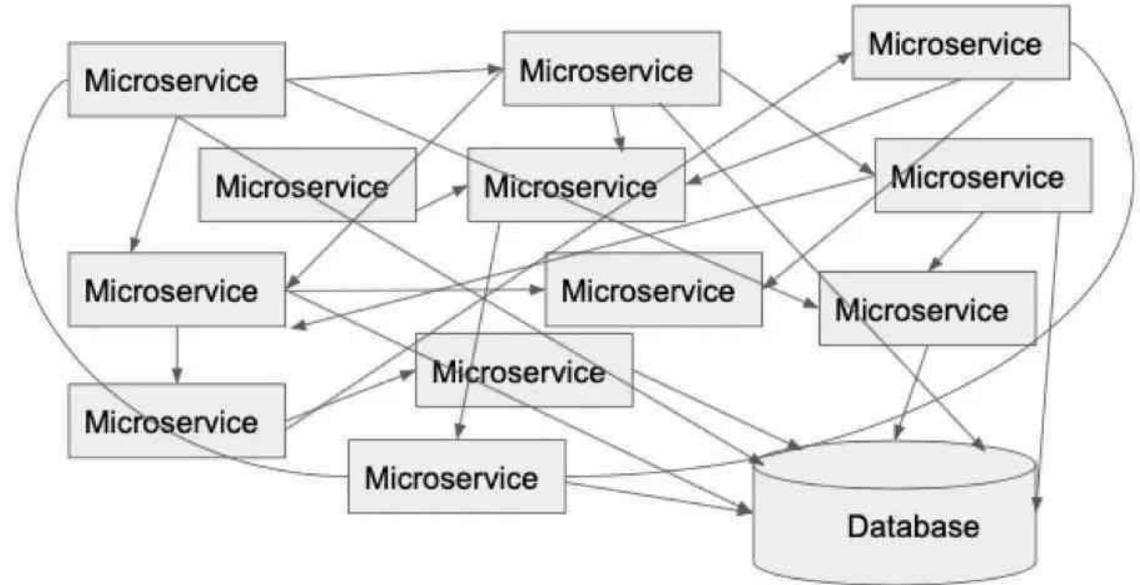
<https://github.com/Advanced-Observability/cross-layer-telemetry>

Justin Iurman, Benoit Donnet
<justin.iurman@uliege.be>

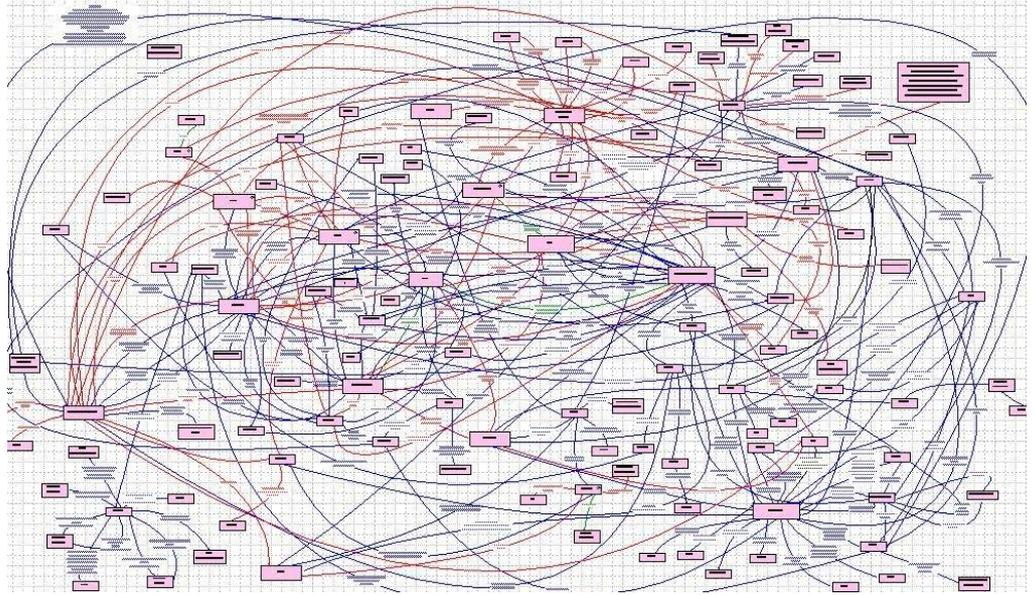
Netdev 0x16
October 28, 2022, Lisbon (Portugal)

From Monolithic to Microservice architecture

- + flexibility
- + high reliability
- + independently deployable
- debugging challenges



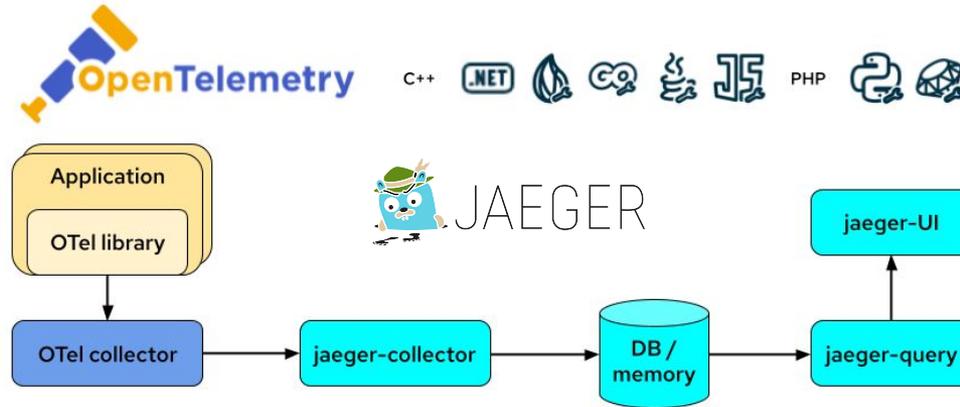
How about that?



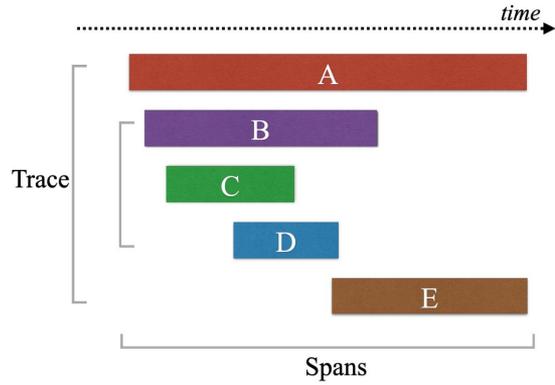
Application Performance Management (APM)

Solution:

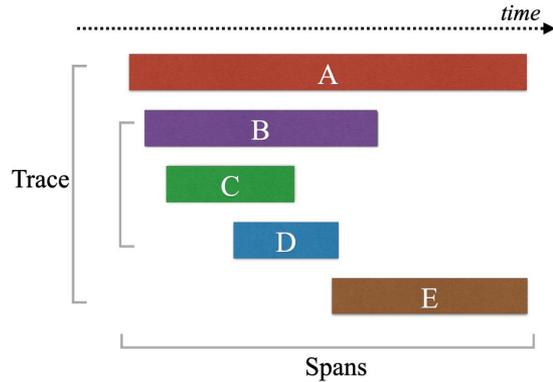
- Distributed tracing (e.g., OpenTelemetry)
- Very useful for (spaghetti-but-not-only) microservices



Tracing world – traces and spans



Tracing world – traces and spans



```
@app.route('/api/generate_pdf', methods=['GET'])
def generate_pdf():
    with tracer.start_as_current_span('generate_pdf') as parent:
        with tracer.start_as_current_span('check_permissions') as child:
            check_permissions()
        with tracer.start_as_current_span('get_pdf') as child:
            get_pdf()
```



Wait a minute...

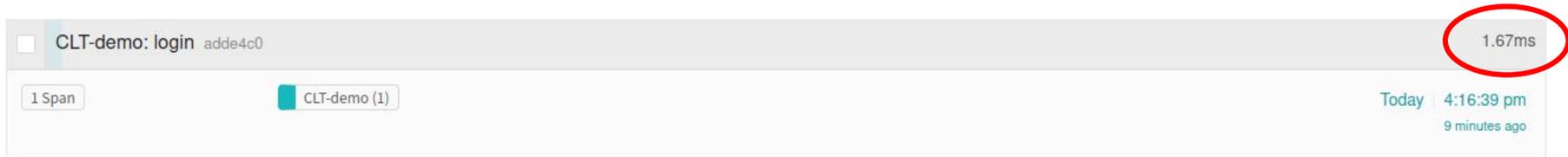
What if the monitored block of code



Monitoring an HTTPS request

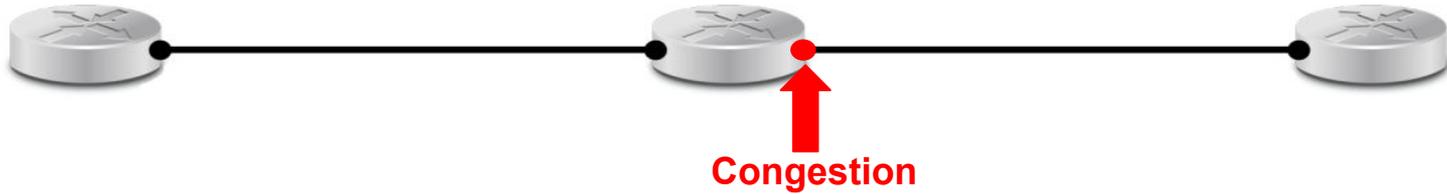
Example: login API

```
@app.route('/api/login', methods=['POST'])
def login():
    with tracer.start_as_current_span('login') as span:
        https.request('POST', '/login', body, headers)
        resp = https.getresponse().read()
```



Monitoring an HTTPS request

Let's simulate a delay somewhere on the path...



Is it the app, the server, the DB, a network issue, ...?

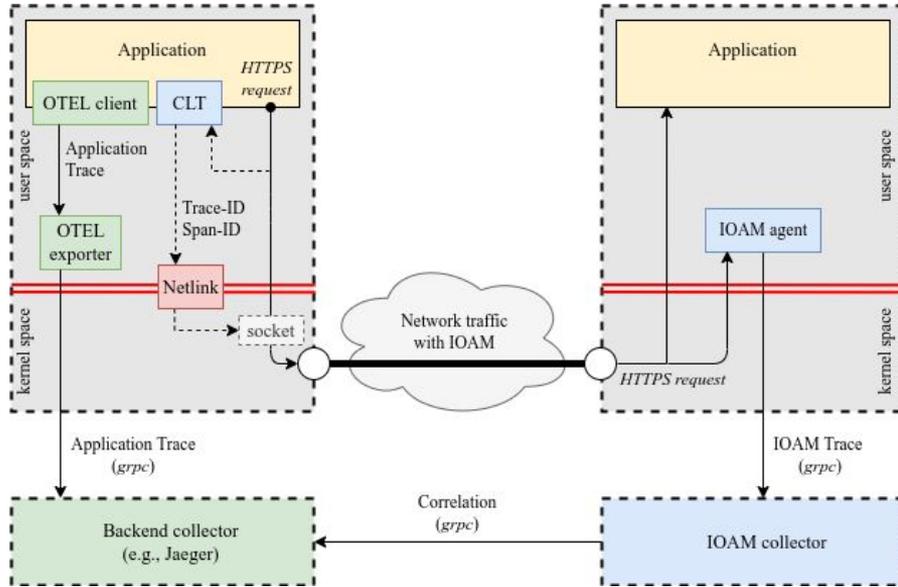
⇒ All we know is that it's slow... **why** ?!

Cross-Layer Telemetry

“Cross-Layer Telemetry (CLT) makes the entire network stack (L2→L7) visible to monitoring tools, instead of the classic application level visibility”.

Cross-Layer Telemetry

“Cross-Layer Telemetry (CLT) makes the entire network stack (L2→L7) visible to monitoring tools, instead of the classic application level visibility”.

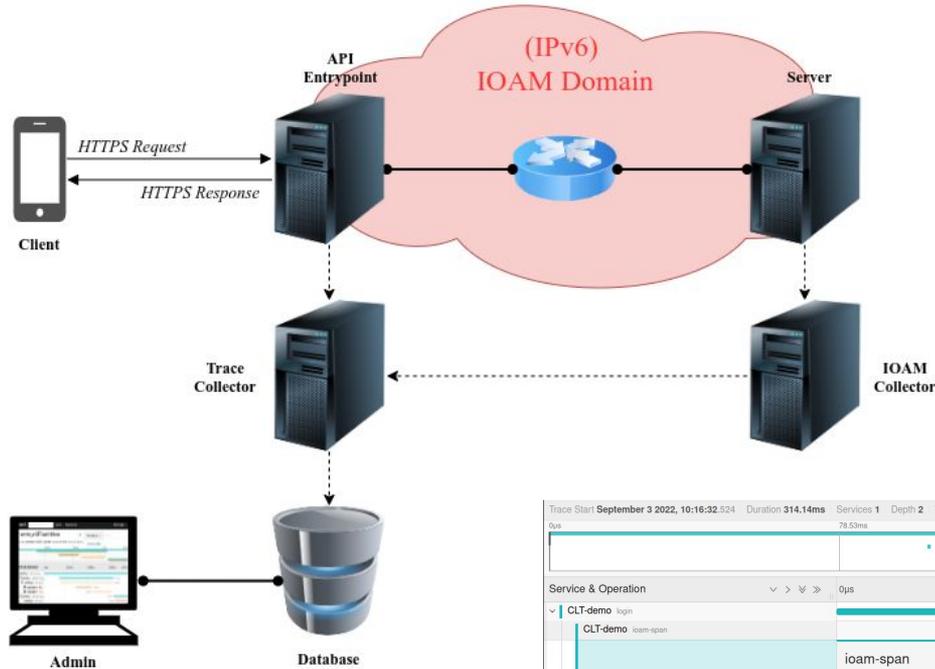


Patch attempt

- Quite straightforward (see <https://github.com/Advanced-Observability/cross-layer-telemetry/blob/main/CLT.patch>)
- Add 2 fields (128 and 64 bits for the trace and span IDs) to:
 - struct sock
 - struct sk_buff
- Add a netlink call to pass IDs from userspace (copy IDs to the corresponding socket)
- Copy IDs from socket to skb in:
 - tcp_sendmsg_locked
 - (udp6_sendmsg)
- Insert IDs within IOAM (tmp solution)



OpenTelemetry/Jaeger on steroids



Trace Start: September 3 2022, 10:16:32.524 | Duration: 314.14ms | Services: 1 | Depth: 2 | Total Spans: 3

Service & Operation	0µs	78.53ms	157.07ms	235.6ms	314.14ms
CLT-demo login					
CLT-demo ioam-span					

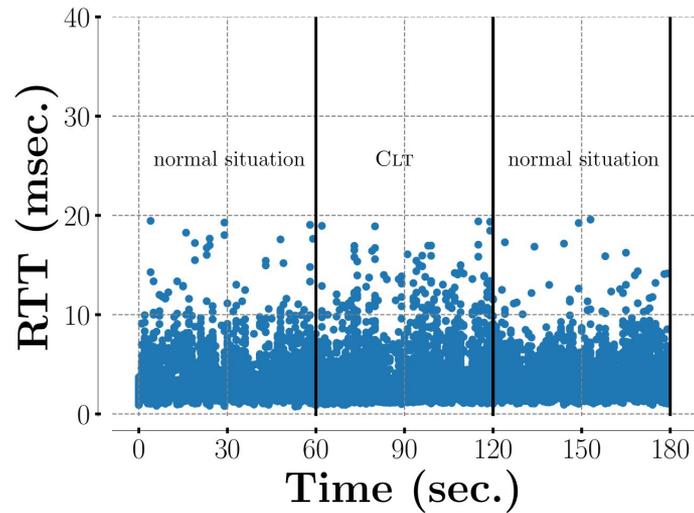
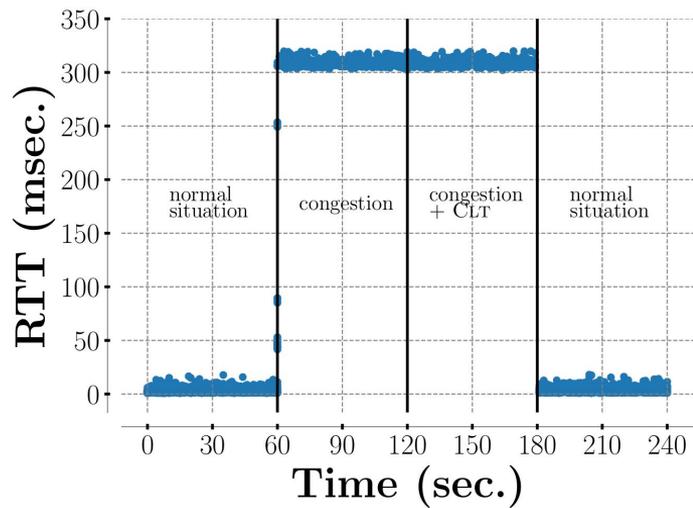
Service: CLT-demo | Duration: 15µs | Start Time: 102.47ms

ioam-span

Tags

internal.span.format	jaeger
ioam_namespace123_node1	HopLimit=64; Id=1; IngressId=65535; QueueDepth=0; (Entrypoint)
ioam_namespace123_node2	HopLimit=63; Id=2; IngressId=21; QueueDepth=1446; (Router)
ioam_namespace123_node3	HopLimit=62; Id=3; IngressId=31; QueueDepth=0; (Server)
otel.library.name	ioam-tracer

Performances



IETF work

“Carrying a Generic Identifier in IPv6 packets” [I-D.draft-ieurman-6man-generic-id]

<https://datatracker.ietf.org/doc/draft-ieurman-6man-generic-id>

→ potential solution to CLT standardization instead of using IOAM to carry IDs

Conclusion

- Useful
- Any interest to have support in the kernel?
- Thoughts on patch design?

Thank you

<https://github.com/Advanced-Observability/cross-layer-telemetry>