Protected Audience on-device vs server side

Fabian Höring
f.horing@criteo.com

2020 – Turtledove on-device

- The user visits an advertiser page
- Ad tech buyer is called to provide bidding script, ads and IG data
- Browser joins Interest Group calling joinAdInterestGroup
- At a later stage, the user visits a publisher page
- Seller calls runAdAuction, fully run in the browser leveraging IG data

Protected Audience on-device

PROS	CONS
• Private by design, users own their data • Auction is happens on-device and cannot be manipulated	• Campaign Budget capping is not possible as there is always a delay • Very high latency as advertising has significant workloads

Now - Real time Key Value service

- Buyer can provide a trusted bidding signals URL and key in the Interest group
- Trusted bidding signals are executed during runAdAuction and results is injected in generateBid
- Execution happens inside a trusted server (TEE) to prevent any data leaking out of this server

KV Service trusted server

- Side-effect free
- No network, disk access, timers, or logging
- Look up in memory state and send back result
- Keys/values uploaded by ad tech with offline process

Protected Audience on-device with KV Service

PROS

- Campaign Budgeting can be handles
- Other expensive computation linked to user data can be of sourced server side
- Simple API, input key, result JSON object

CONS

- TEE brings more complexity and additional infra cost compared to on-device
- Latency still high (less on device, server side call)

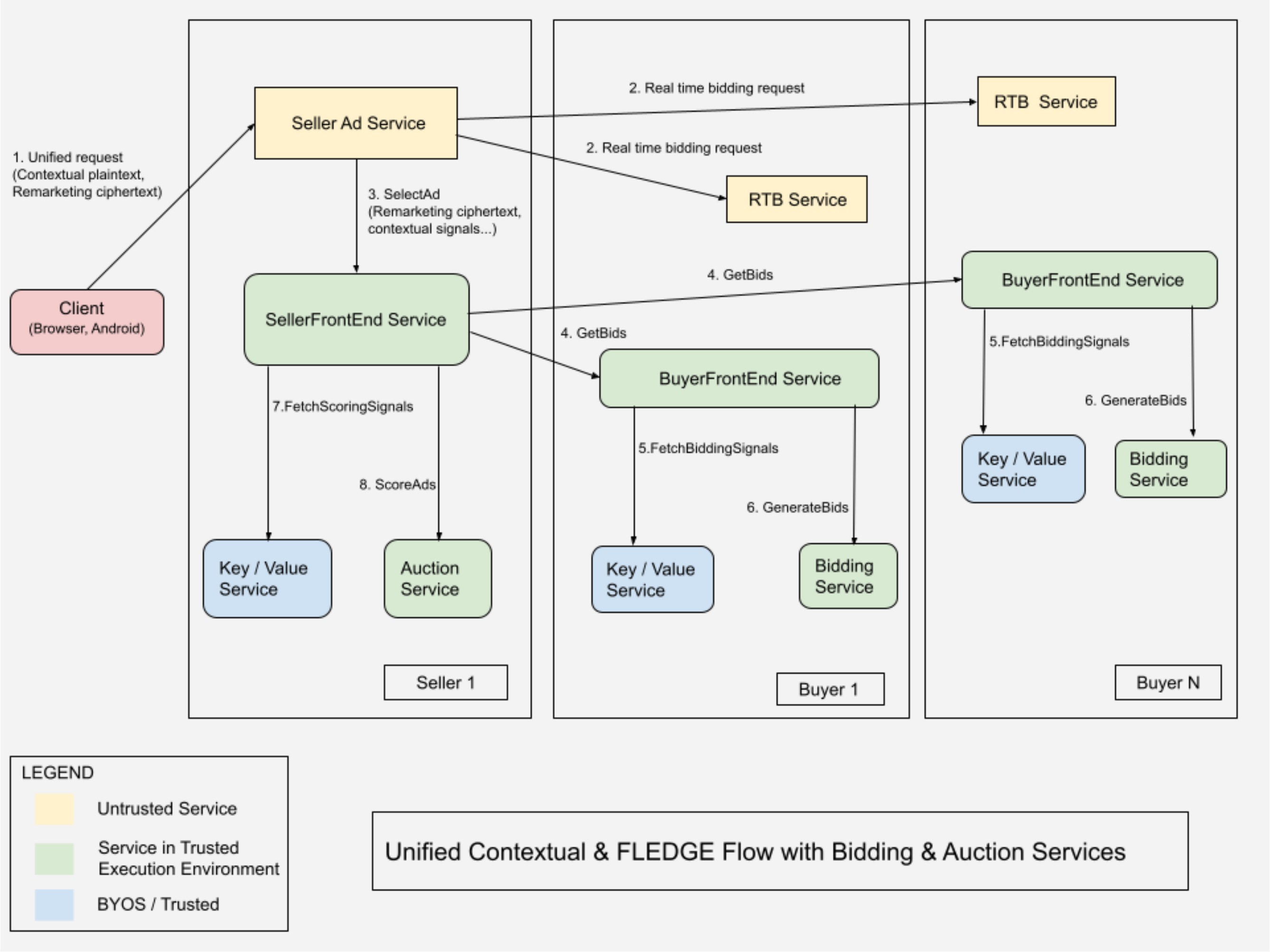
>2023 - Bidding and Auction services

- Interest Group tagging In browser as before
- Seller calls gets auction blob from browser by calling getInterestGroupAdAuctionData
- Sends auction blob to servers by calling fetch
- Gets response and complete auction in browser by calling runAdAuction
- Rendering and FF reporting happens as usual

Bidding and Auction service

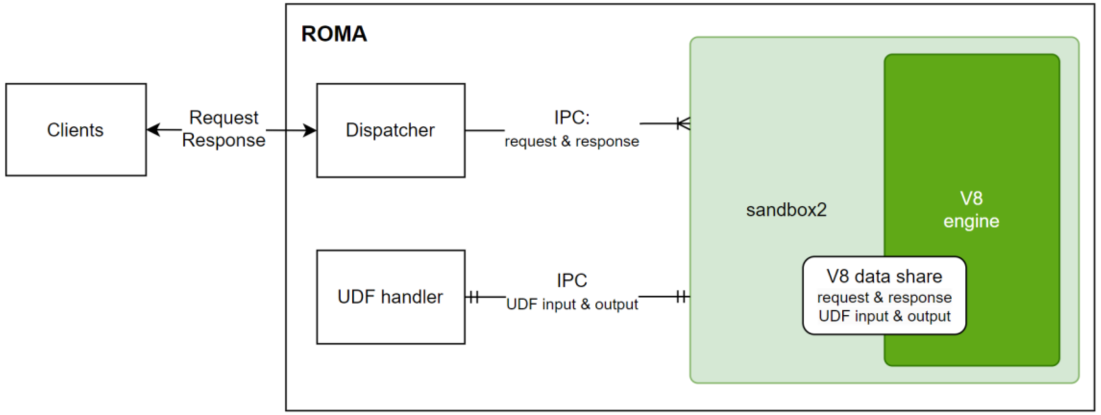
PROS	CONS
• Reduced latency, better allocation of resources and auction fairness as computation can be scaled by buyers and seller server side	• More introduced complexity as Seller and Buyer calls need to be synchronized server side • Potentially higher latency due to very high blob payload sent to servers

Server side part



Same components as on device

ROMA engine



- No Open Source requirement, ad tech can execute UDFs
- Sandbox, currently using V8 JS engine inside Sandbox V2
- Dispatcher, pre-allocated workers
- UDF handler

ROMA engine

PROS	CONS
• Allows to run proprietary custom code • Execution engine can be stable	• Sandboxing overhead • Only JS & WASM allowed

Criteo KV service ROMA engine benchmark

Locally compiled KV server: 1 instance, 16 cores, 16 GB with a very basic implementation

- it reads the keys from the request
- it queries the in-memory datastore to get the values of those keys
- it replies with those values
- C# asp.net vs Google key/value service with same input & output

Results

- KV service implementation can handle 1000s of queries per second with ms second latency
- Asp.net can handle 10 times more queries with better mean response times
- c# WASM can handle 1 QPS (file size 30 MB), WASM doesn't seem like a real alternative for managed runtimes

Efficient In memory caching

- Currently all data lookups from bidding script to KV state need to be serialized/de-serialized
- For real time bidding server must reply within 50ms
- **Optimizations here actually matter** (where we might not care much elsewhere)
- Network calls and IPC should be reduced or batched
- Over 30 caches in our Prod systems on PA POC

Efficient In memory caching, examples

- Efficient filtering, sending back all campaigns by country & all campaigns by domain and doing the intersection in the bidding layer
- ML Inference Sidecar, Inference side will require IPC, overhead might be significative, times 5 with internal ONNX benchmarks

Server Infra cost might quickly get out of control

- JS ROMA vs Native c# => **x5**
- No shared memory => **High** (30 caches prod system)
- ML inference side car => **x5** on ML inference (internal ONNX benchmark)
- TEE and encrypted networking => **+20%** (usual symmetric encryption overhead)

Future work – server side

- More work is needed to mitigate potential infra cost increase
  - Shared memory
  - Inlining data structures
  - New ROMA BringYourOwn binary execution engine to allow custom languages beyond JS & WASM

- Common method of benchmarking, improvements should be trackable easily, web load tests

Future work – on device

- Flexibility to move payload on-device server side and back
- Ability to AB test on-device vs server side for direct performance/latency/cost metrics
- Continue exploring on device (parallelization rollout at scale, more metrics on resource allocation, CPU watchdogs)

Questions