My Research

Outline

1. ICDCS’20: Win with what you have.
2. CRII’21: With Uncertainty Comes Opportunity: Providing Best-Effort Edge Services with Uncertain and Limited Resource
3. Polymorphic Web Service: Just rejected by ATC’23.
4. Preparing for CAREER

Win with What You Have: QoS-Consistent Edge Services with Unreliable and Dynamic Resources

Zheng Song, Eli Tilevich
zhesong@umich.edu, tilevich@cs.vt.edu

Who we are?

Zheng Song, Assistant Professor, CIS@UMDearborn

- Edge/mobile computing
- Opportunistic systems
- QoS optimization/Resource Allocation
- A Ph.D. alum of Dr. T.’s Software Innovations Lab@VT

Eli Tilevich, Professor, CS@VT

- Systems end of software engineering
- Distributed system and middleware
- CSE / music informatics

Research Vision

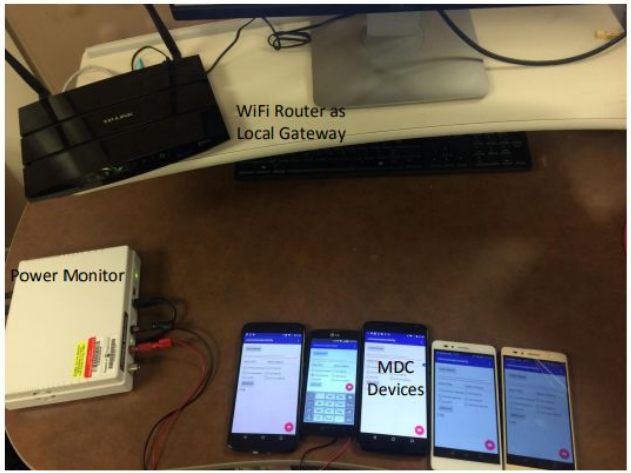
Provide QoS-guaranteed Services

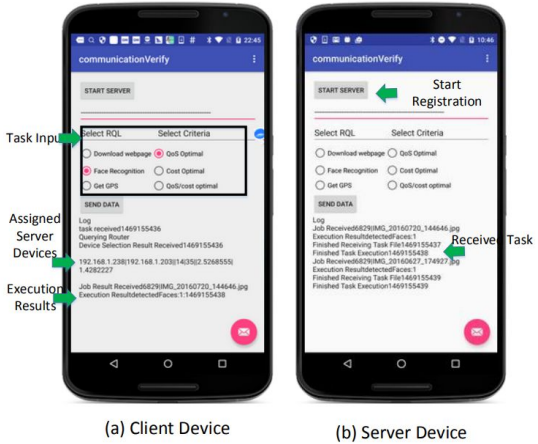
in Opportunistic Edge Environments

with Self-Organized Edge Devices

Our Contributions to Mobile Edge Computing

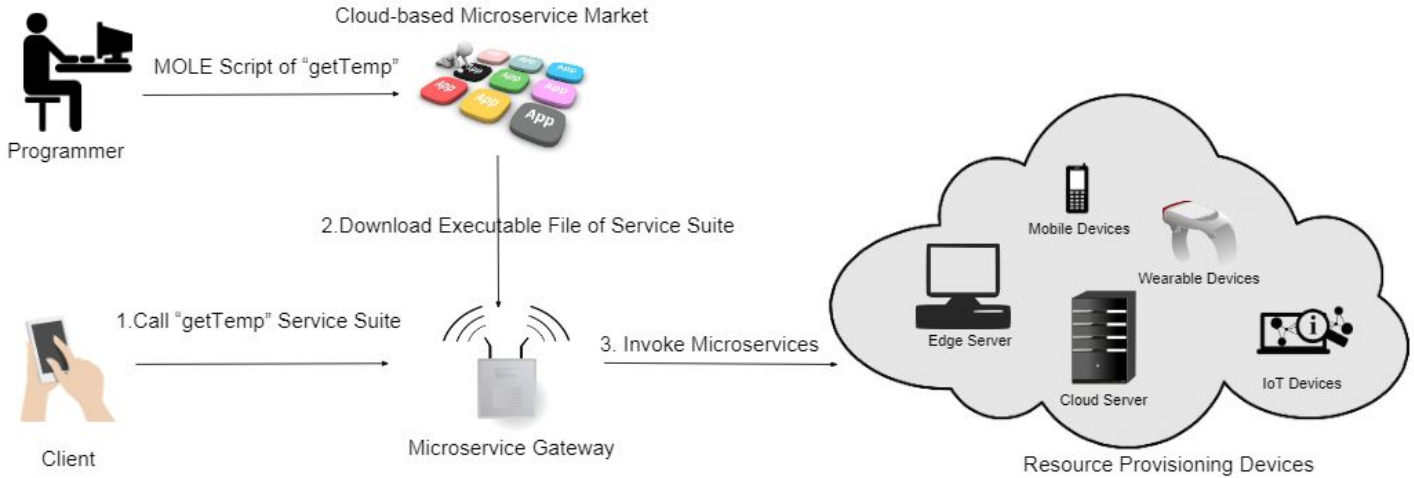
1. PMDC: *IEEE CLOUD'18*
   - Remote execution on nearby mobile devices
   - Gateway for pairing execution demand and resource supply





Our Contributions to Mobile Edge Computing

2. MOLE: *IEEE EDGE'19* Best Paper
   - Provide Services on Edge Devices
   - Improve Reliability/Latency

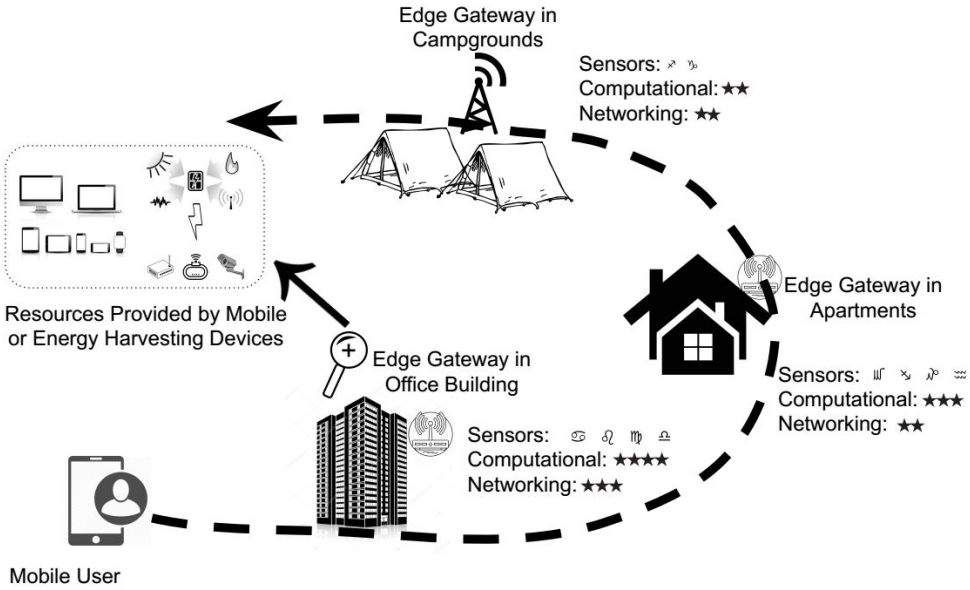


Outline

1. Motivating Example
2. Related Works
3. Solution Overview
4. Evaluation and Results
5. Lessons Learnt

Problem Statement

- Resources in Opportunistic Edge Environments
  - Heterogeneous
  - Scarce
- How to best satisfy service QoS requirements in pervasive edges?
  - High reliability (i.e., >97%)
  - Medium latency (i.e., <100ms)
  - Low resource consumption (cost)



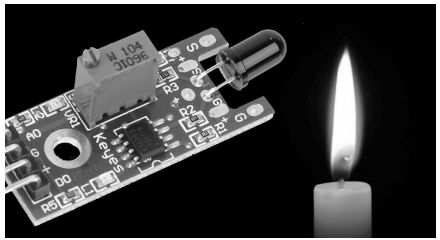
Existing Approaches

1. Guarantee QoS in Clouds:
    - redundant resources, allocated at runtime (serverless computing)
    - Replicate redundant resources at edge?
2. Offload to nearby edges/cloud:
    - Remote resources cannot provide local context / networking

Secret Weapon: Equivalent Functions

For example, detecting fire:

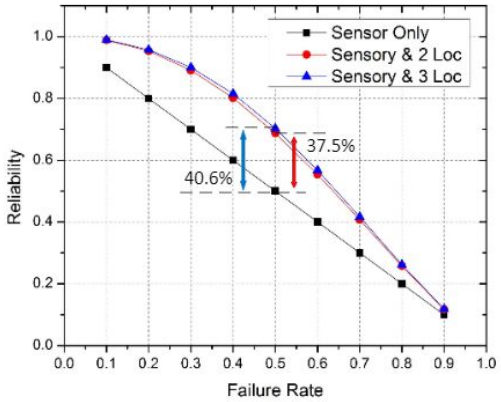
- smoke by a surveillance camera
- smoke by smoke sensors;
- flame by flame sensors;
- the change of CO/CO2 level by gas sensors;
- the temperature change by a temperature sensor.



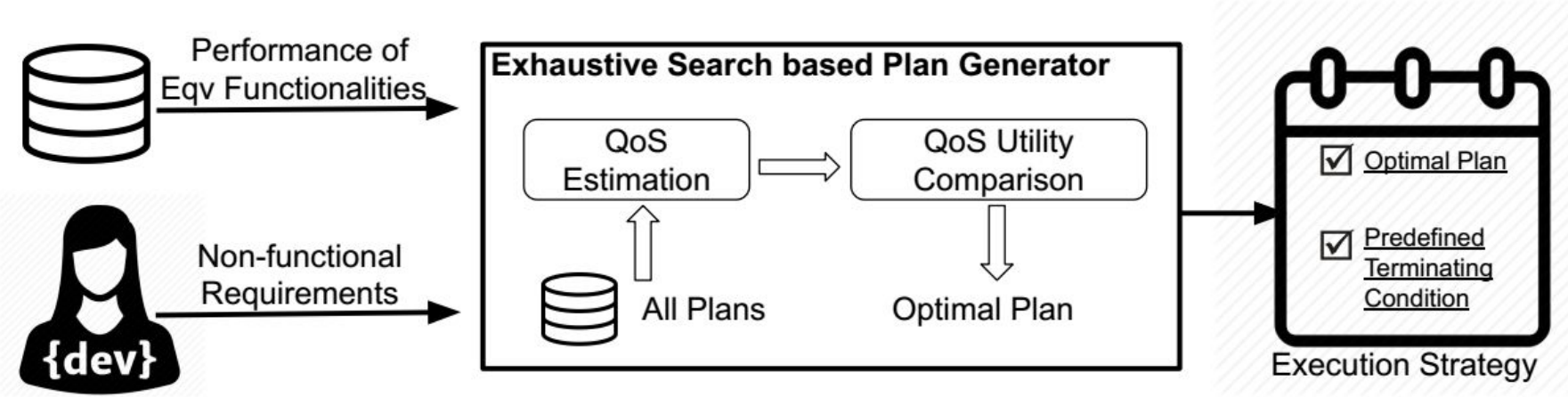
Expressing Combined Execution of Equivalent Functions

- Programming support for specifying how to execute equivalent functions
- Qualitatively improve QoS (trading one QoS aspect for another)
  - Fail-over: improve reliability, higher latency
  - Speculative parallel: improve reliability, lower latency, higher cost
- **How to quantitatively maximize the QoS satisfaction level of an edge service?**

```
Service getTemp {
   MS: getTempSensorReading {
      device: select("Sensor.Temperature")
      on.success: ret temp
      on.fail: getTempbyLocation
   }
   MS: getTempByLocation {
      device: select("Internet")
      req: location
      set: ep.max_retry to 3
      on.success: ret temp
   }
   MS: getLocationByGPS {
      device: select("Location.GPS_PROVIDER")
      on.success: ret loc as location
   }
   MS: getLocationByCellID {
      device: select("Location.NETWORK_PROVIDER")
      on.success: ret loc as location
   }
}
```



Solution Overview



Research Challenges

Given a set of equivalent functions and a QoS requirement, how to execute them?

⇒ Can we :

- Given equivalent functions, find all possible strategies for their combined executions
- Given their QoS, estimate the QoS of these strategies?
- Given the QoS estimations, find out a strategy that best fit the requirement?

Finding all Execution Plans for Eqv. Functionalities

24 game: given 4 numbers in the range from 1 to 9, binary operators (+, -, *, /), and parentheses, form an arithmetic expression that equals to 24.

- Step 1: put 9 digits into 4 slots: P(9,4)
- Step 2: put any of 4 operators into 3 slots between digits; 4^3
- Step 3: process parentheses, alter the precedence of 3 operator slots.

n	2	3	4	5	6
distinct plans	5	31	305	4471	87545

Table 2: Execution Plans for Equivalent Sets of Size *n*

Non-functional Performance Estimation

Different terminating conditions, different calculation methods;

Basic rules: 1) No folding. First calculate d=(a-b), then calculate d*c

2) Start time and end time of each equivalent functionality:

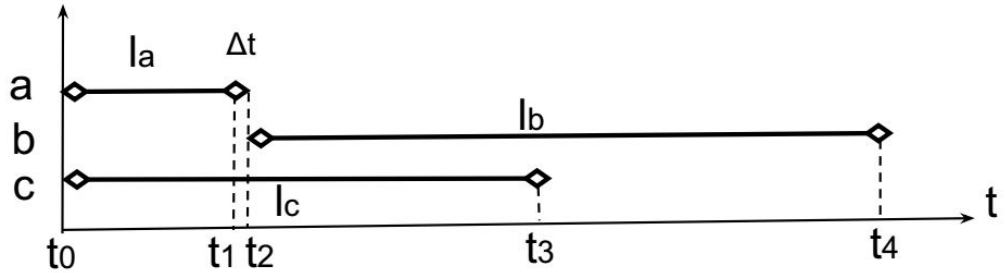


Figure 6: Execution Timeline for $(a-b)*c$

QoS Utility Index

Features:

- increase when a QoS attribute improves
- increase faster while the attribute is being satisfied, slower afterwards

$$u_n(s) = \begin{cases} -k\dfrac{|q_n(s) - Q_n|}{Q_n}, \text{if } q_n(s) \preceq Q_n \\ \dfrac{|q_n(s) - Q_n|}{Q_n}, \text{if } q_n(s) \succ Q_n \end{cases} \quad \forall n \in \mathcal{N}, k > 1$$

System Design

Feedback Loop + Automated Orchestration Strategy Generation

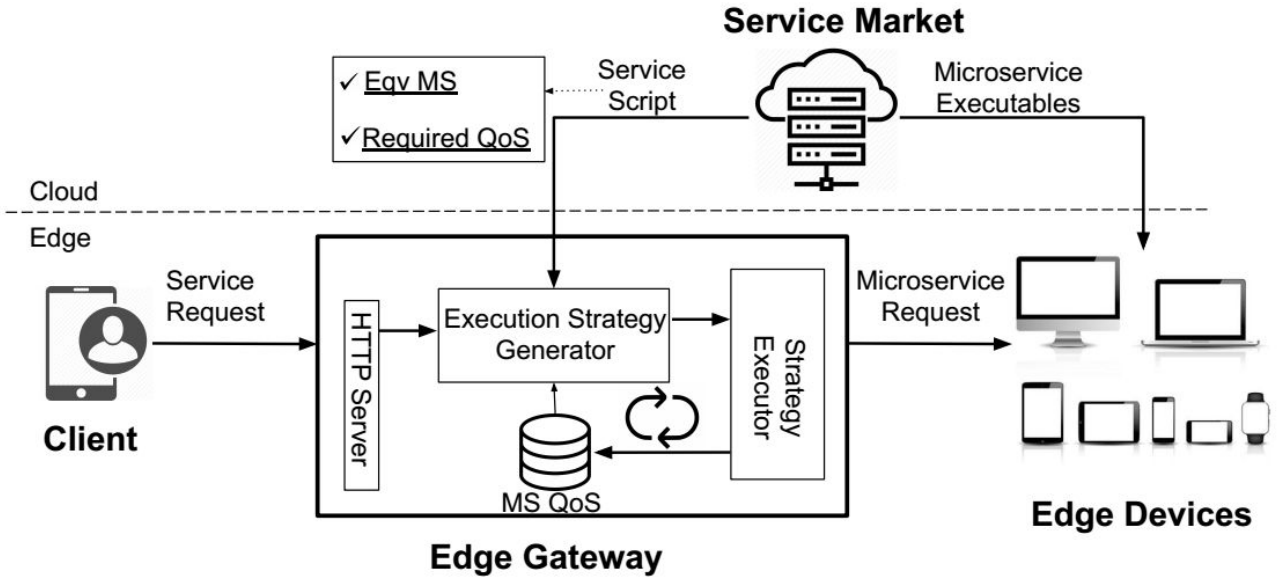


Fig. 2: System Design for Provisioning Edge Services

Programming Model

1. Direct Specification:

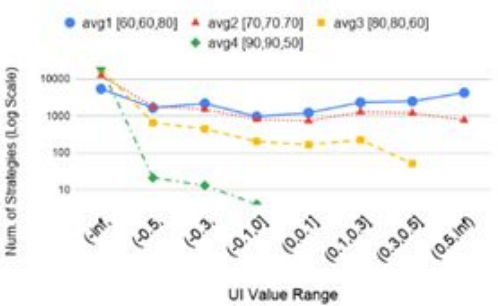
```
eqvFuncs := {a, b, c}
s := strategy(
    executionPlan := a * ( b - c ),
    terminatingCondition := results.maxSD(1)
)
return s.execute(execParameter of eqvFuncs)
```

2. Automated generation:

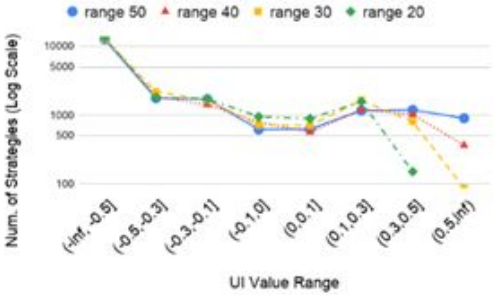
```
eqvFuncs := {a, b, c}
//set non-functional characteristics
a.setNonFuncCharac(reliability, latency, cost)
b.setNonFuncCharac(reliability, latency, cost)
c.setNonFuncCharac(reliability, latency, cost)
strategy s := someGenerator.generate(
    eqvFuncs,
    NonFuncRequirements := (reliability>90,w_l=high)
)
return s.execute(execParameter of equFuncs)
```

1. Do these strategies really matter for QoS

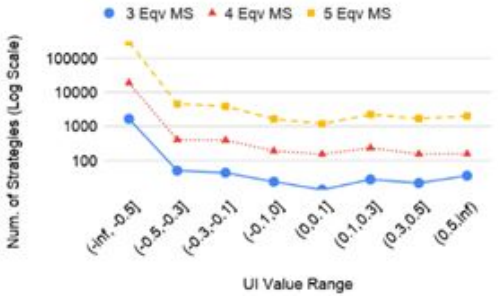
More equivalent functions, higher QoS variations → bigger advantages of the optimal strategy over others



(a) Exp1: Varying avg [c, l, r]



(b) Exp2: Varying QoS Range(Δ)

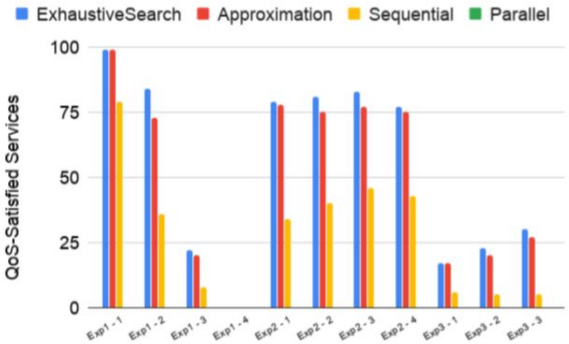


(c) Exp3: Varying Number of Eqv. Microservices

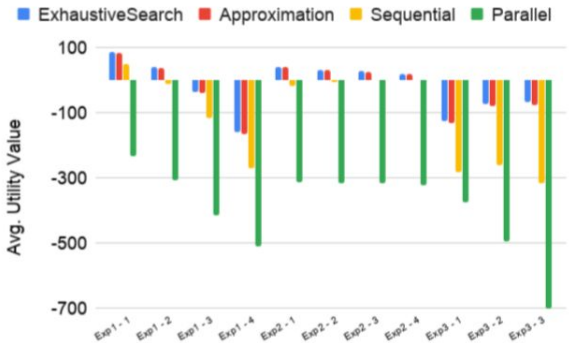
2. Do the generated strategies improve QoS?

increase the ratio of QoS-satisfied services by 2~2.6 X

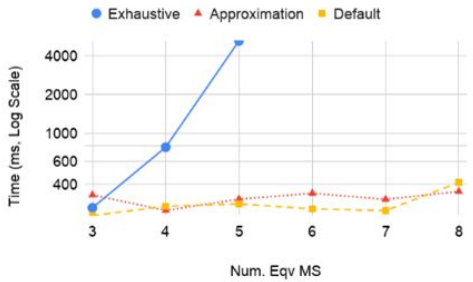
Exhaustive search can be used for <5 eqv functions, while approximation applies for more eqv functions



(d) Number of Services with Fully Satisfied QoS of Different Generated Strategies



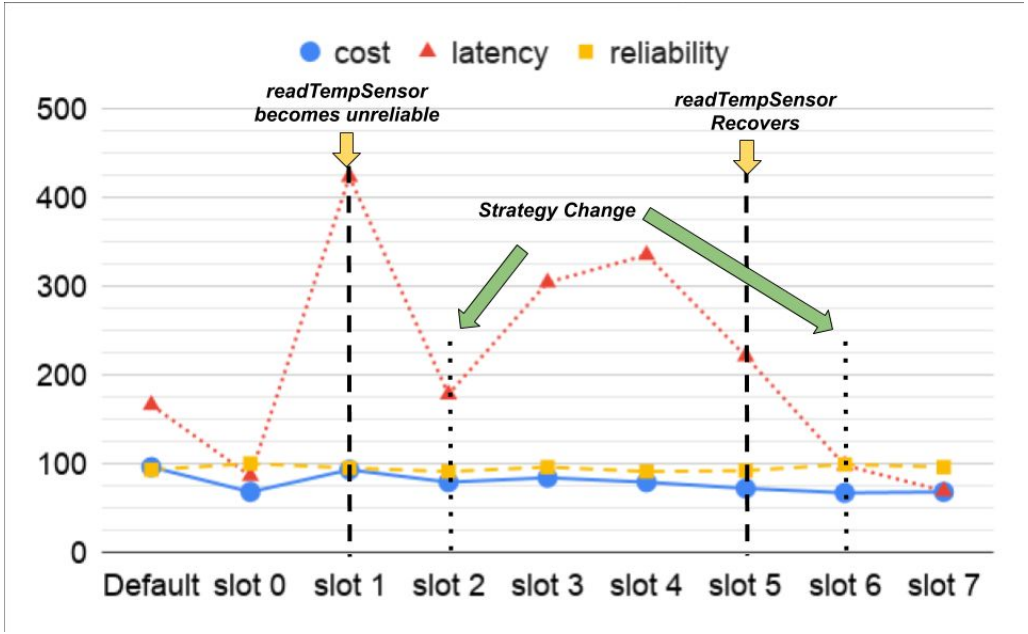
(e) Average Utility Values of Generated Strategies



3. Does the solution work in a realistic environment?

outperforms MOLE by 31% in cost, 52% in latency, 4% in reliability

adaptive to resource fluctuations



Lessons Learned

1. The combined execution of equivalent functions can improve QoS
2. Their orchestration strategies make a tangible difference, rendering existing template-based orchestrations less efficient
3. The customized orchestrations can quantitatively improve QoS

In pervasive edges, orchestrating equivalent functions can best satisfy QoS.

CRII

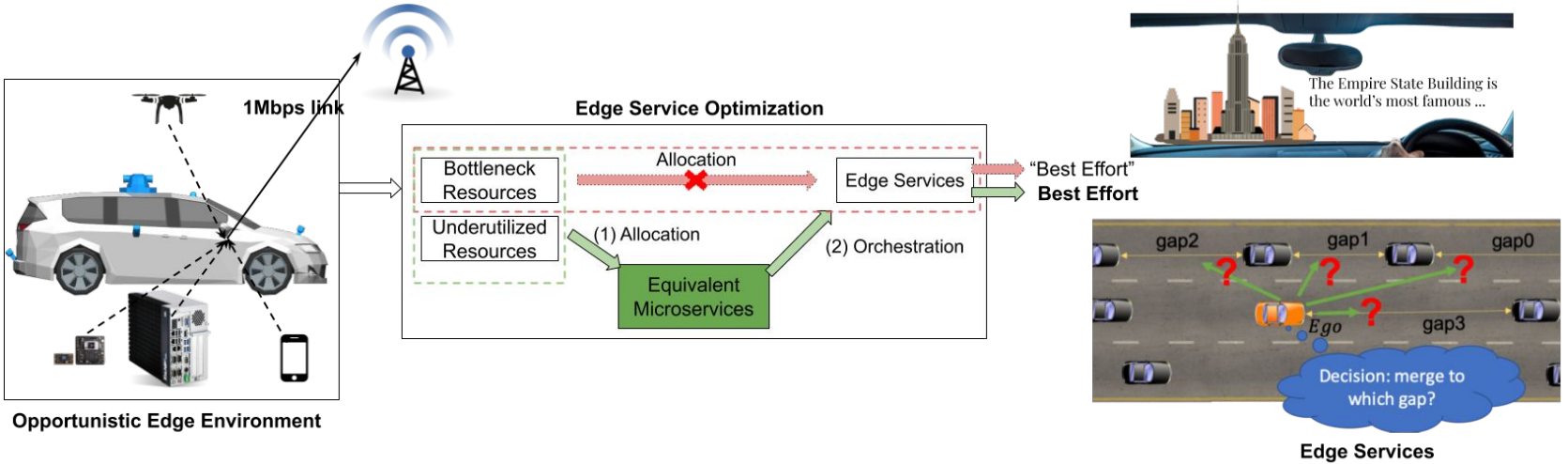
https://www.nsf.gov/awardsearch/showAward?AWD_ID=2104337&HistoricalAwards=false

Award Abstract # 2104337 CRII: CNS: With Uncertainty Comes Opportunity: Providing Best-Effort Edge Services with Uncertain and Limited Resources

The goal of this project is to create a runtime system for providing best-effort edge services. Different from clouds with manageable and redundant resources, many edge environments are opportunistic and feature uncertain and limited resources. As a result, edge services cannot guarantee Quality of Service (QoS) for their clients, which is a critical obstacle of creating current and emerging edge computing applications. Existing solutions fall short, as they optimize the overall QoS by allocating bottleneck resources competed for by services. In contrast, this project is based on the insight that the uncertainty of edge environments also brings opportunity so the equivalent functions that do not consume the same resources can satisfy the same service request. Hence, this project will explore how to systematically improve QoS by dynamically orchestrating equivalent functions that consume underutilized resources.

Current Research Focus --- Middleware

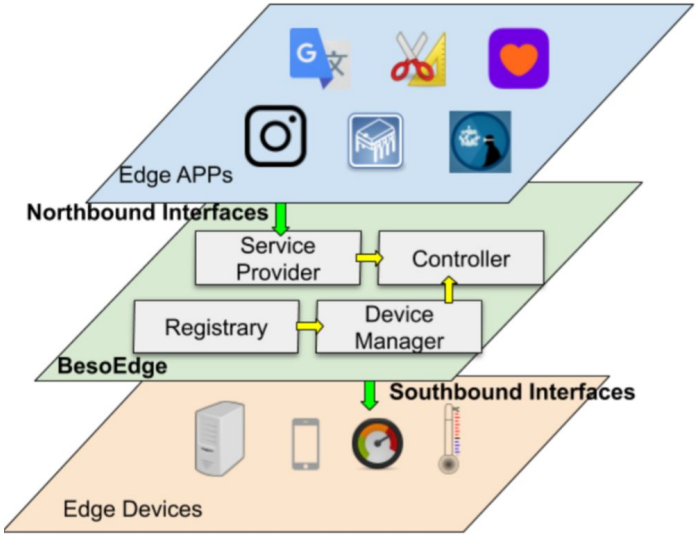
BesoEdge: a runtime system for providing best-effort edge services over self-organized edge devices

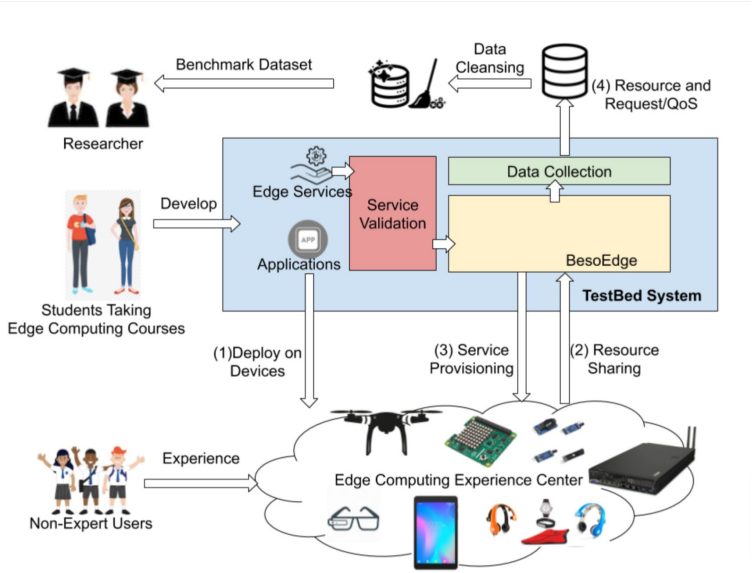


Differences from ICDCS

ICDCS's focus: Resources already allocated for each function → Change how to orchestrate them

CRII: Resources → How to allocate resources for each func. + Orchestration plan

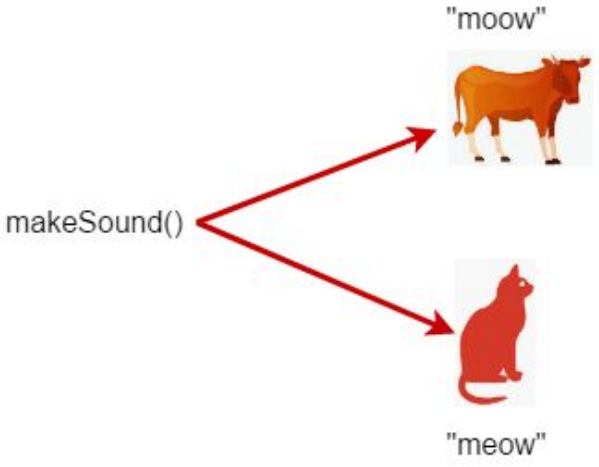




Polymorphic Web Service

What is a polymorphism?

In computer science, a polymorphic object is an object that is capable of taking on multiple forms. The kind of polymorphism the object undergoes depends on *when* the object takes its form and *what part* of the object is transforming.



Web Service Usage Model in Mobile Applications

- QoS matters!
  - Reliability, latency, availability, etc.
  - Impact user’s experience
  - Theoretically bound by Service Level Agreement (SLA)

- For invoking a web service in mobile apps:
  - Manually pick one service and hard code into the APP
  - APP developers have no control after releasing

Limitations of the Current Model

Maintaining Satisfactory QoS for mobile APPs is challenging:

1) Most web services don’t provide SLA guarantees

2) QoS may fluctuate over location and time:
   a) Hard to manually pick one service that fits your needs
   b) one service may not always providing satisfactory performance in all contexts (location and time)

Equivalent Translation Web Services

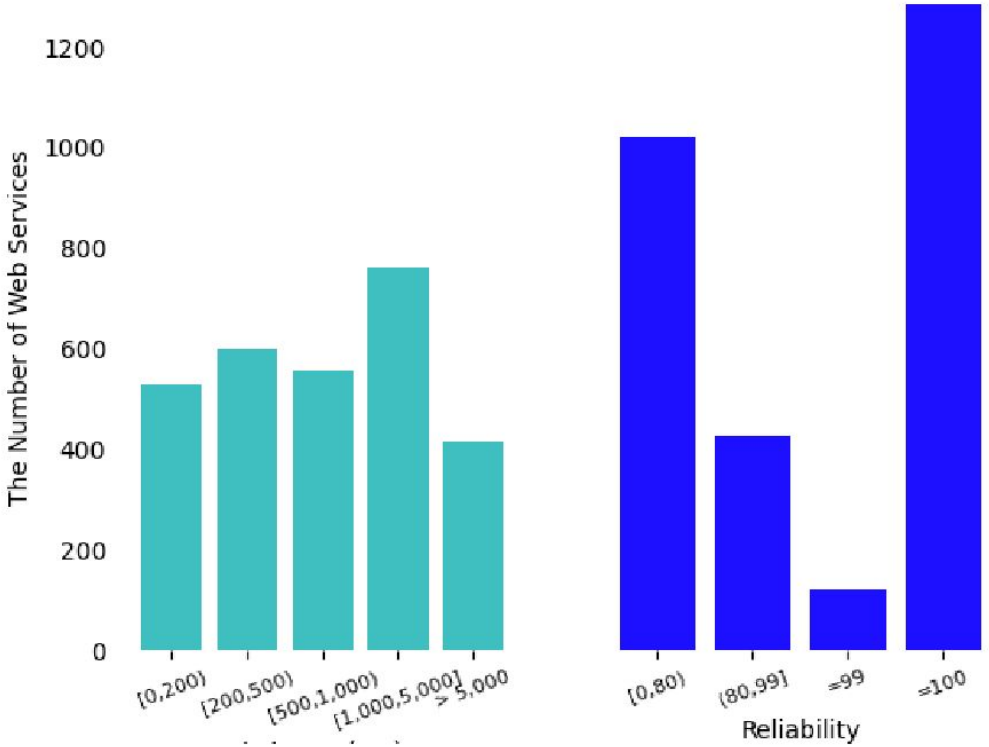
	Cost	Latency	Reliability
Just Translated	$8.5/Mon	3,159ms	99%
Google Translate	$20.0/Mon	636ms	90%
Deep Translate	$9.00 /Mon	1,942ms	99%
Text Translator	$25.00 /Mon	97ms	100%

Data take from RapidAPI website https://rapidapi.com/hub

Study: How Equivalent Services Impact QoS?

1. Data Collection
    a. RapidAPI website
    b. Real invocations
2. Data Analysis
    a. Is SLA a good solution to guarantee use experience?
    b. Can preselected One web service always outperform to its peers?
    c. Can combined invocation of Multiple equivalent APIs improve performance?

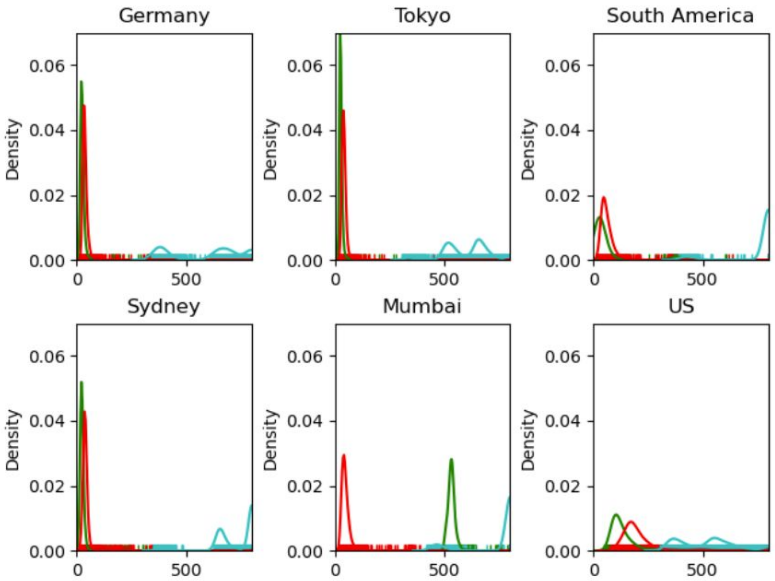
Is SLA a good solution to guarantee use experience?



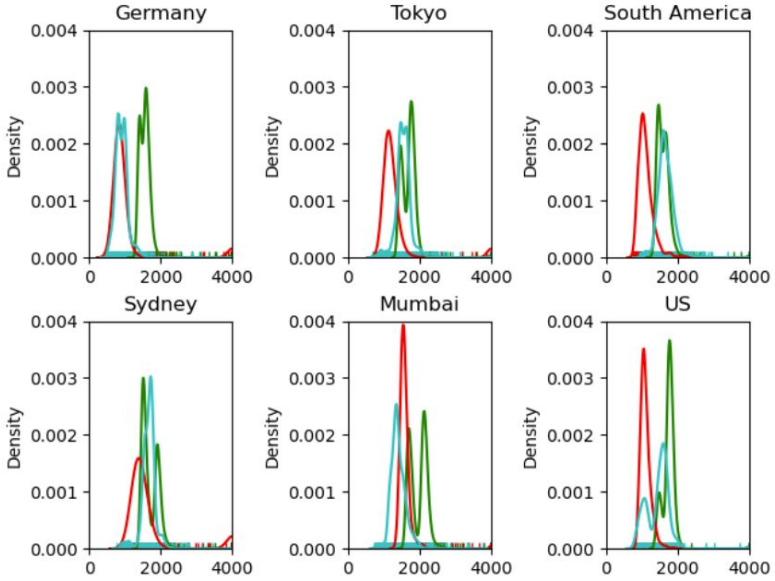
1. Only 1416/5784 have .99 of reliability
2. Only 3/5784 has legitimate SLA documents

SLA is NOT a good solution to guarantee use experience

Can preselected web service always outperform to its peers?



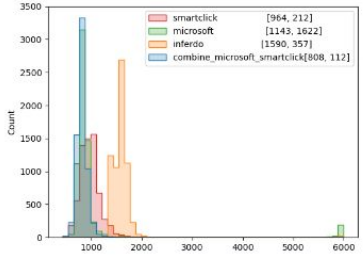
Latency Distribution of IP2Location APIs



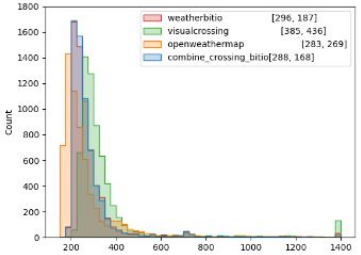
Latency Distribution of Face Detection APIs

Observation 2: Preselect web service Cannot always outperforms its peers in different contexts

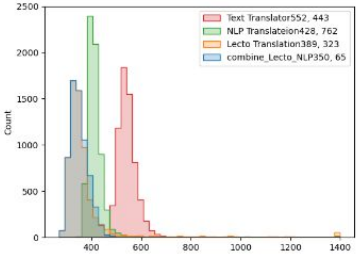
Can combined invocation of Multiple equivalent APIs improve performance?



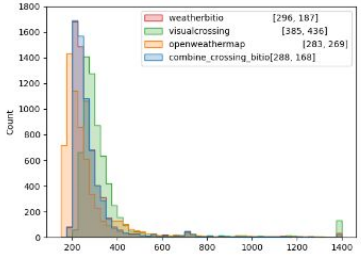
(a) Case 1: Face Detection invoked from Germany



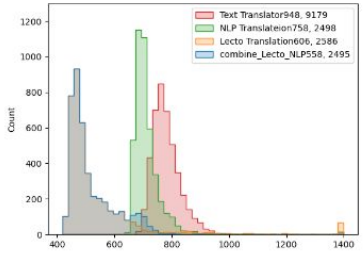
(c) Case 3: Language Translation invoked from South America



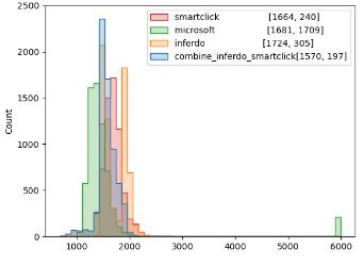
(e) Case 5: Translation Combinations in Germany



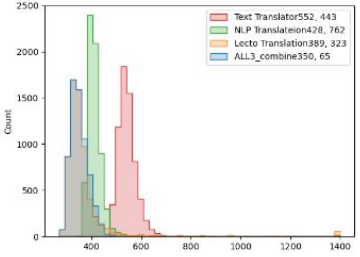
(g) Case 6: Weather Forecast invoked from MI, US



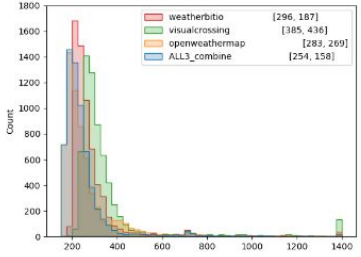
(b) Case 2: Language Translation invoked from Tokyo



(d) Case 4: Weather Forecast invoked from Mumbai



(f) Case 5: Translation Combinations in Germany

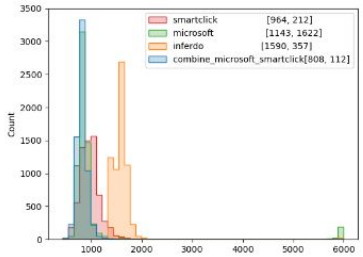


(h) Case 6: Face Detection invoked from Sydney

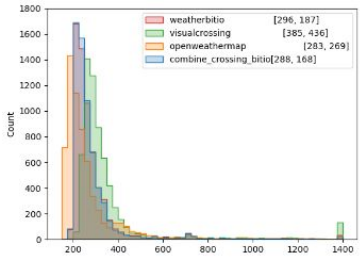
Fig. 4: Latency Distribution of Invoking Multiple Web Services in the Speculative Parallel Fashion.

```
a * b * c * d * e; //speculative parallel
```

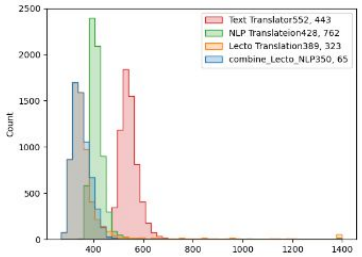
Can combined invocation of Multiple equivalent APIs improve performance? — YES (avg improve rate: 31% on avg latency, 42% on std) (max improve rate: 83% on avg latency, 96% on std)



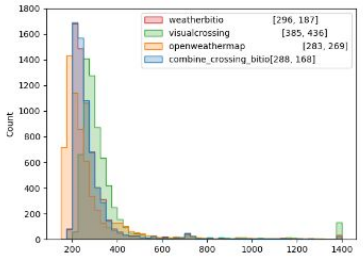
(a) Case 1: Face Detection invoked from Germany



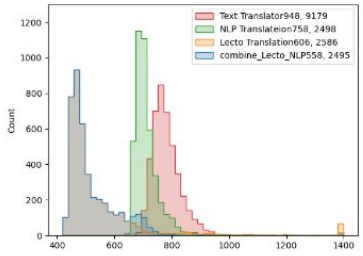
(c) Case 3: Language Translation invoked from South America



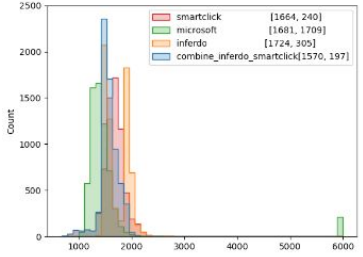
(e) Case 5: Translation Combinations in Germany



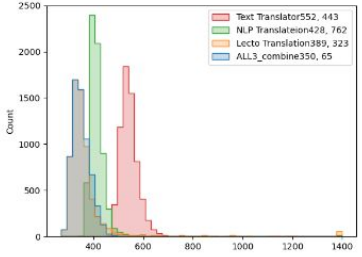
(g) Case 6: Weather Forecast invoked from MI, US



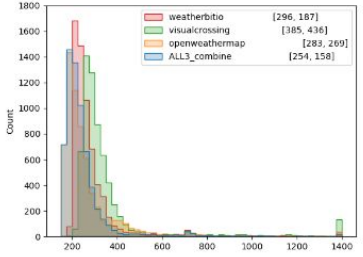
(b) Case 2: Language Translation invoked from Tokyo



(d) Case 4: Weather Forecast invoked from Mumbai



(f) Case 5: Translation Combinations in Germany

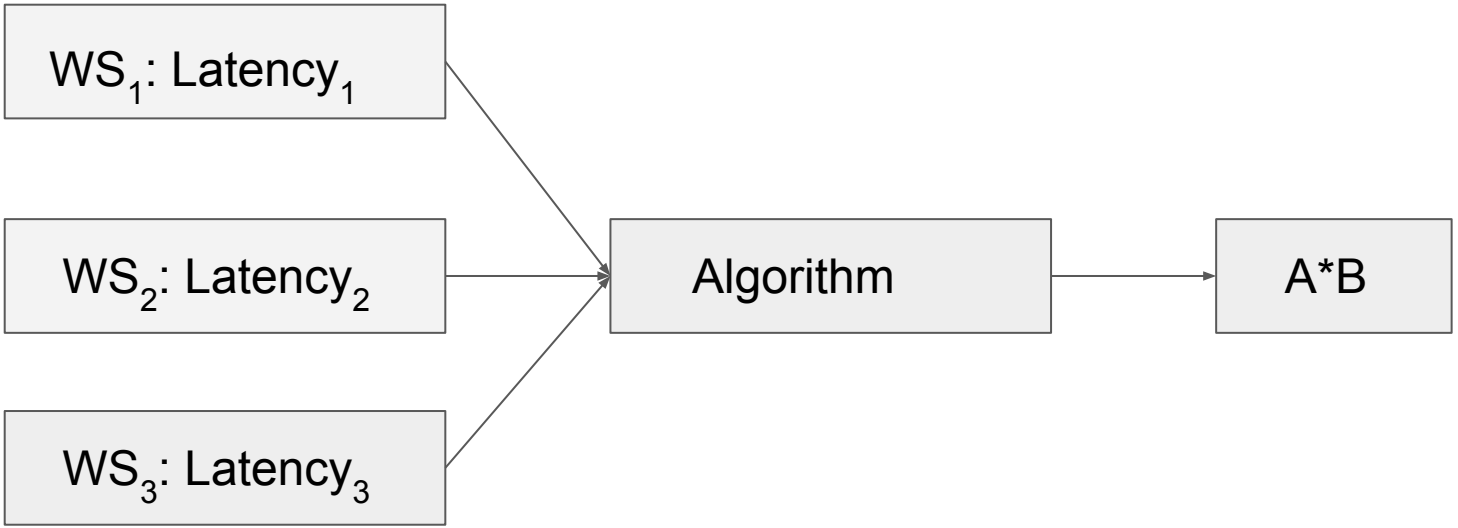


(h) Case 6: Face Detection invoked from Sydney

Fig. 4: Latency Distribution of Invoking Multiple Web Services in the Speculative Parallel Fashion.

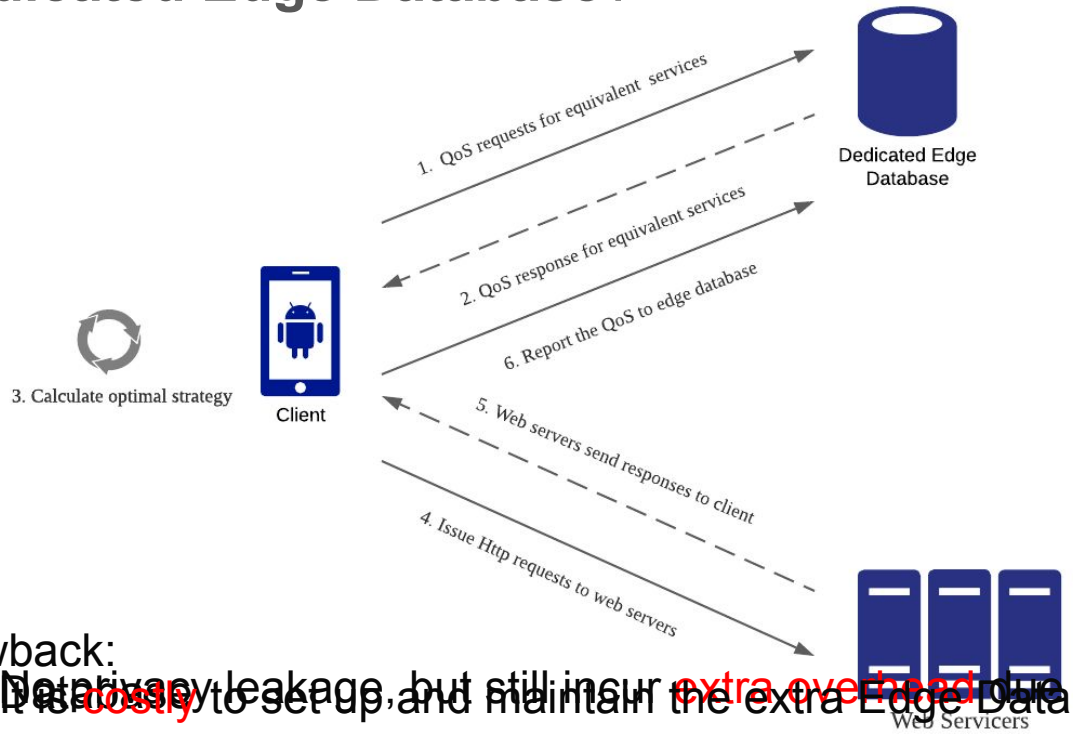
How to Determine which Services chosen to combine?

A*B
A*C
B*C
A*B*C



How to store and retrieve updated QoS information?

Dedicated Edge Database?



Drawback:

1. Data privacy leakage, but still incur extra overhead due to interact with the Edge
2. It is costly to set up and maintain the extra Edge Database

Outline

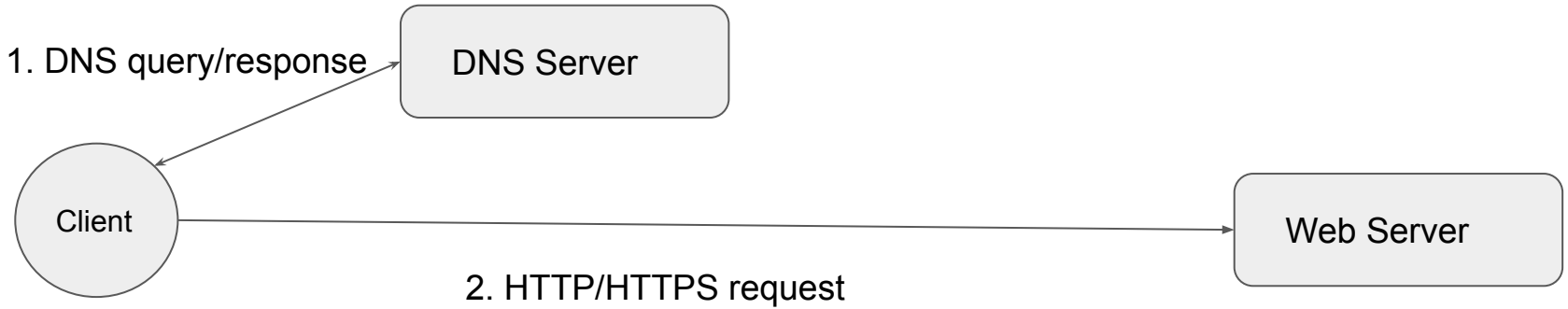
1. Background and Problem Motivation
2. **Our Solution**
3. Implementation
4. Tentative Evaluation

Background: DNS

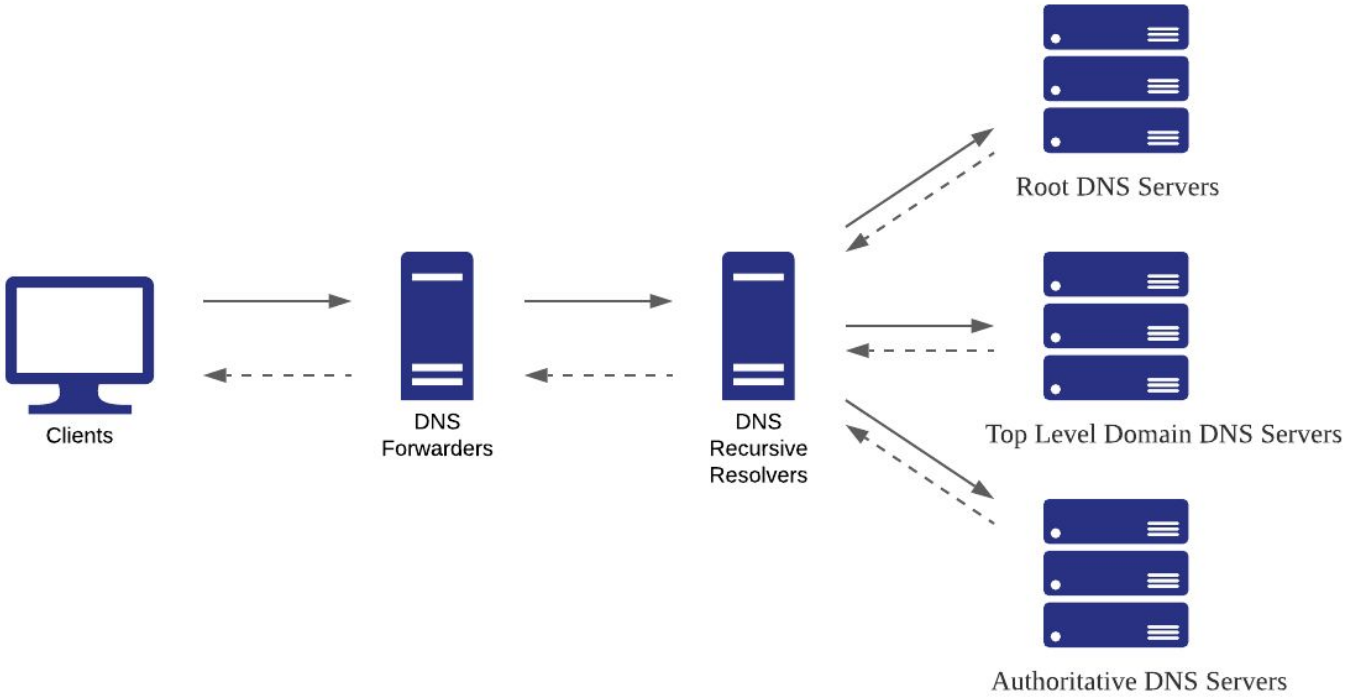
1. Responsible for convert domain name to IP address

   www.google.com → 142.251.32.4

2. All the http/https requests involve the above DNS process

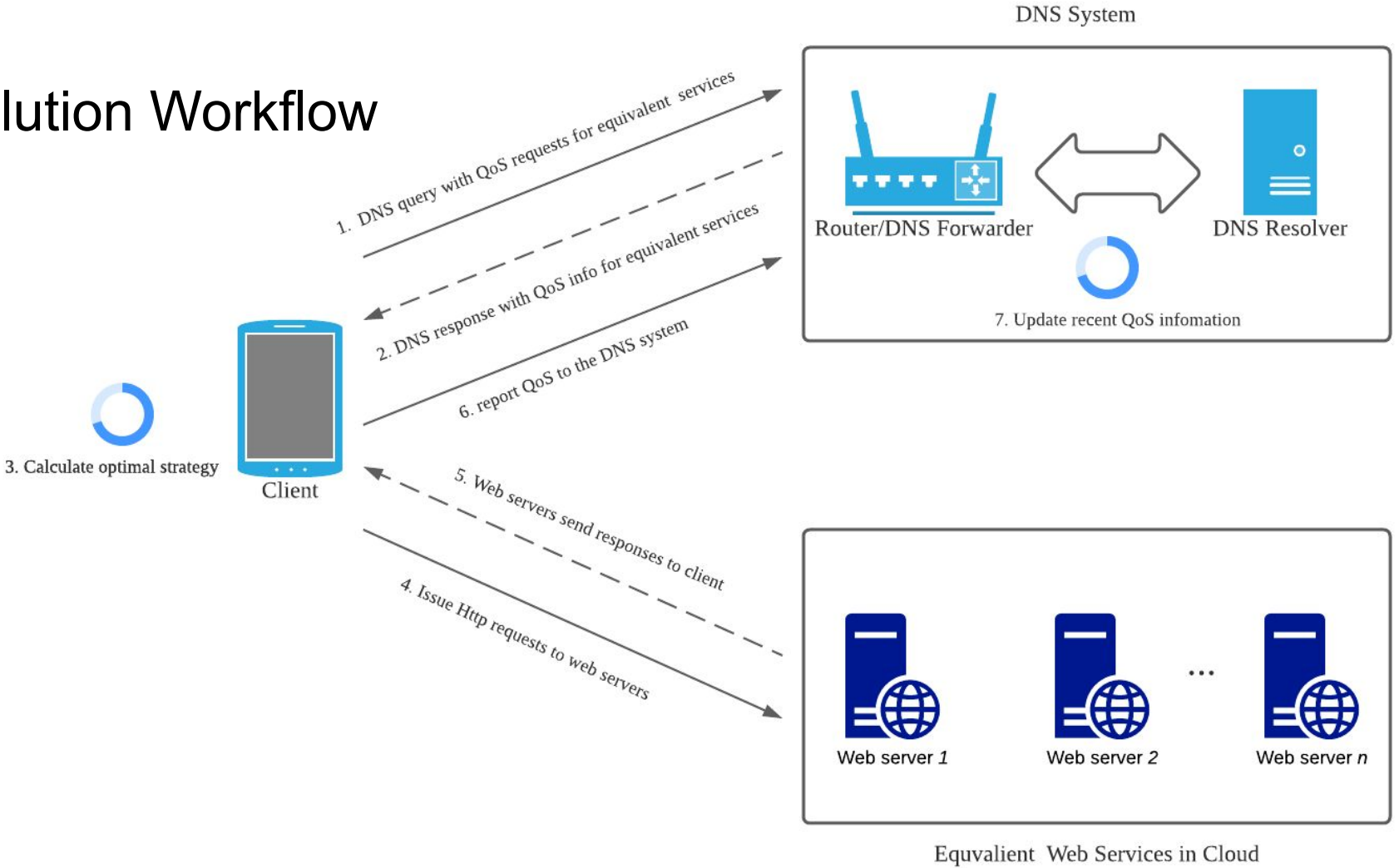


Modern DNS Infrastructure



DNS Infrastructure with Multiple Layers

Solution Workflow

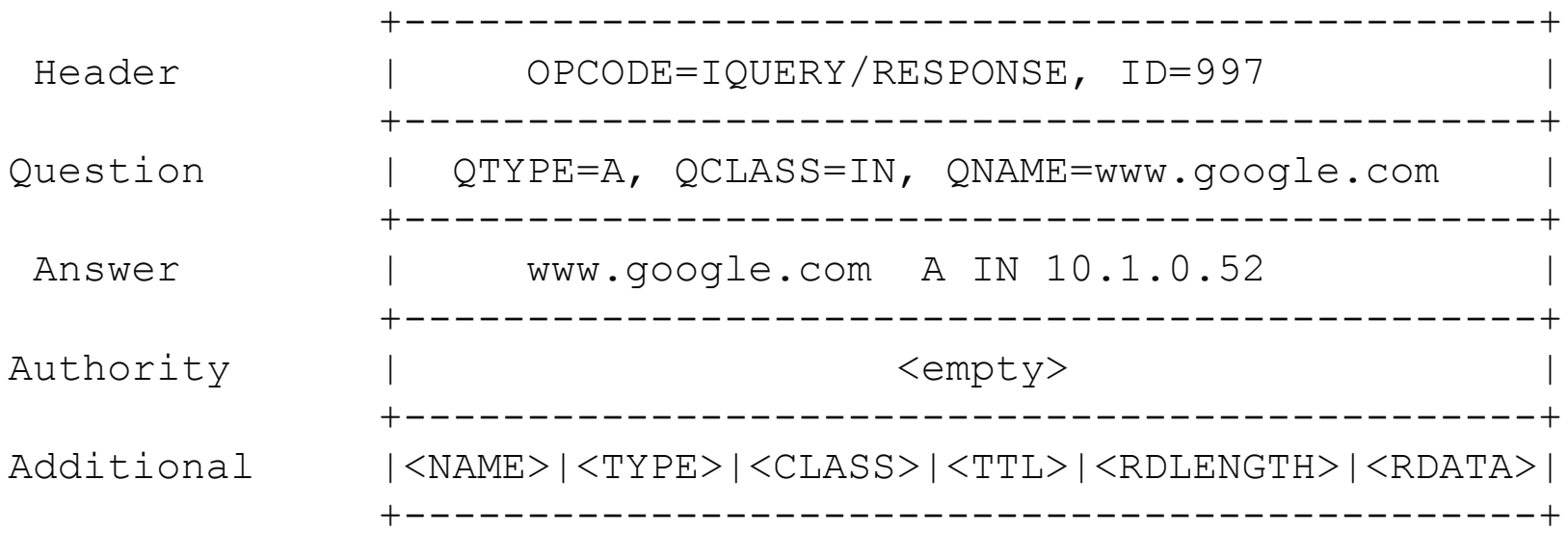


Solution Designs

1. Piggybacking into DNS packets to minimize QoS query overhead
2. Retrofitting existing DNS system to store and retrieve QoS information
3. A mobile runtime to orchestrate equivalent web services by utilizing QoS
4. A reputation model to prevent
    a. attackers from injecting malicious QoS data
    b. encourage sharing

Design Details: DNS Packet Piggyback

DNS Request / Response ⇔ Location-specific Service QoS Inquiry/ Result



Design Details: DNS Packet Piggyback

DNS Request / Response ⇔ Location-specific Service QoS Inquiry/ Result

```
                +-----------------------------------------------+
 Header         |     OPCODE=IQUERY/RESPONSE, ID=997            |
                +-----------------------------------------------+
Question        |  QTYPE=A, QCLASS=IN, QNAME=www.google.com     |
                +-----------------------------------------------+
 Answer         |     www.google.com  A IN 10.1.0.52            |
                +-----------------------------------------------+
Authority       |                     <empty>                   |
                +-----------------------------------------------+
Additional      |<NAME>|<TYPE>|<CLASS>|<TTL>|<RDLENGTH>|<RDATA>|
                +-----------------------------------------------+
```

Additional Record Format/Definition for QoS Query/Response:

```
<NAME>     ::= HOSTNAME
<TYPE>     ::= QoS
<CLASS>    ::= QUERY/RESPONSE
<TTL>      ::= NULL
<RDLENGTH> ::= The Length of RDATA
<RDATA>    ::= <CLASS> == QUERY ? HASH(URL): <Latency, Reliability>
```

Design Details: DNS Packet Piggyback

DNS Request / Response ⇔ Location-specific Service QoS Inquiry/ Result

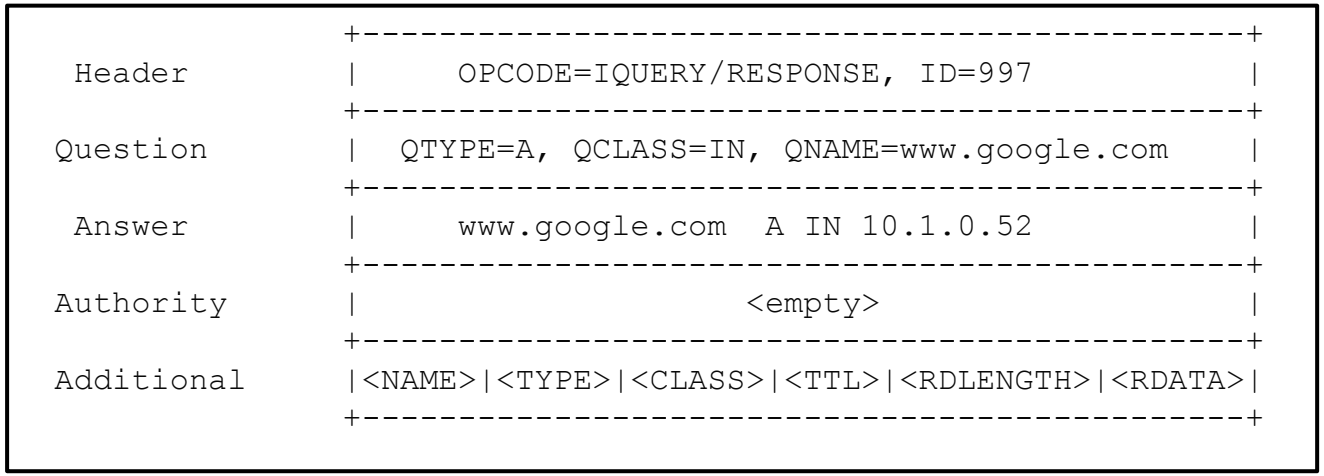
```
                +-----------------------------------------------+
 Header         |     OPCODE=IQUERY/RESPONSE, ID=997            |
                +-----------------------------------------------+
Question        |  QTYPE=A, QCLASS=IN, QNAME=www.google.com     |
                +-----------------------------------------------+
 Answer         |     www.google.com  A IN 10.1.0.52            |
                +-----------------------------------------------+
Authority       |                     <empty>                   |
                +-----------------------------------------------+
Additional      |<NAME>|<TYPE>|<CLASS>|<TTL>|<RDLENGTH>|<RDATA>|
                +-----------------------------------------------+
```

Additional Record Format/Definition for QoS Query/Response:

```
<NAME>     ::= HOSTNAME
<TYPE>     ::= NetReq
<CLASS>    ::= QUERY/RESPONSE
<TTL>      ::= NULL
<RDLENGTH> ::= The Length of RDATA
<RDATA>    ::= <CLASS> == QUERY ? HASH(URL): DATA OBJ
```

Design Details: DNS Packet Piggyback

DNS Request / Response ⇔ Location-specific Service QoS Inquiry/ Result



Additional Record Format/Definition for QoS Query/Response:

<NAME> 0 bytes
<TYPE> 2 bytes
<CLASS> 2 bytes
<TTL> 4 bytes
<RDLENGTH> 2 bytes
<RDATA> 16 bytes or 12 bytes

extra 24Bytes is acceptable:

- Extension Mechanisms for DNS (EDNS) support larger DNS packet without truncation
- Median DNS response size is about 150 Bytes, more than 99.8% of DNS less than 1230Bytes

Design Details: QoS Aggregation within DNS System

- After service requests, client report QoS to DNS system via UDP packages
- Aggregate QoS info at Forwarder and Resolver
  - Forwarder: directly connected to users, accurate QoS, less data
  - Resolver: connected to forwarder, less accurate QoS, more data
  - Forwarder will always report QoS to resolver, which will aggregate QoS from multiple forwarders
  - Why not authoritative DNS server involved? Too far away from the user.

- Forwarder may periodically retrieve QoS from resolver, depending on
  - Number of users attached to a forwarder
  - Latency between forwarder and resolver

- How is service QoS data stored?
  - Hash table: <hash value of service, latency>

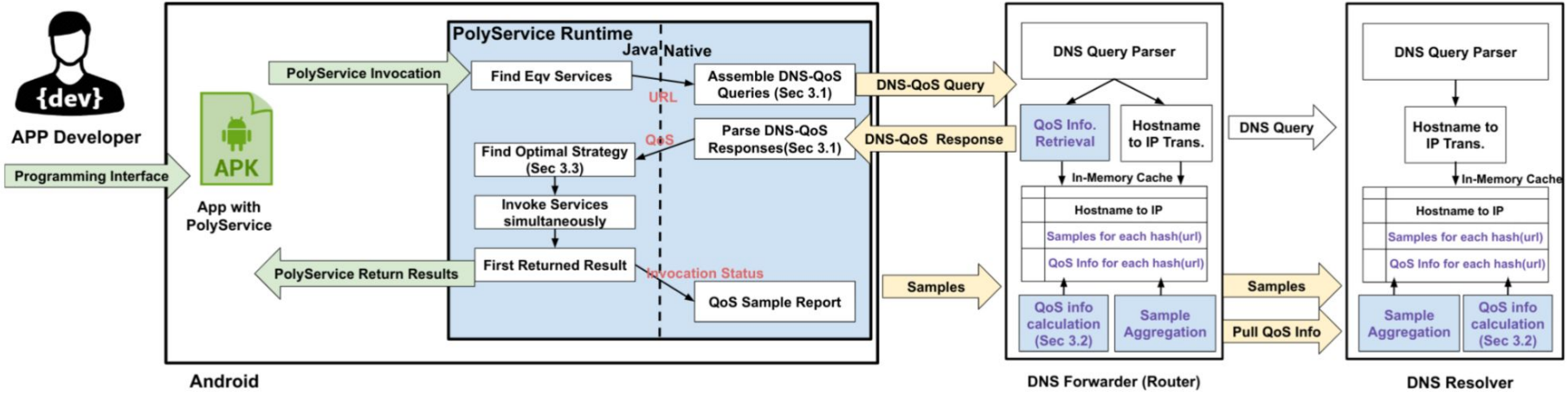
Programming Interface

Example: Translation

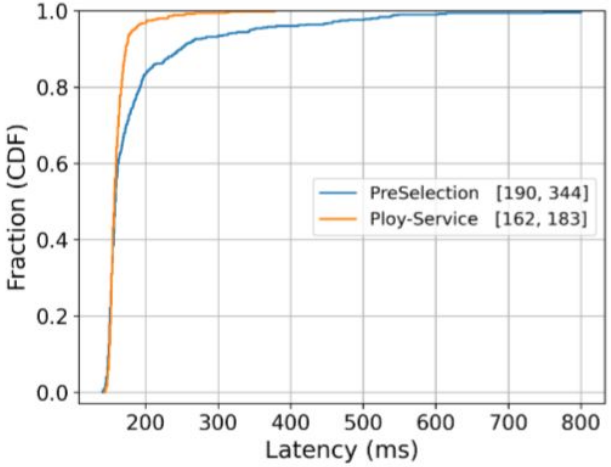
```
// Define TransServ as Poly-Service
Class TransServ extends PolyService{
//TransServ input: (sentence, target language)
//TransServ output: Translation result
    String sentence, lang;
    EqvService LT, NT, TT;
//Declare Equivalent Services and append to Eqv Set
//``token" refers to the authentication keys of services
    @Override
    public void Init(String... args){
        LT=new EqvService("LectoTrans", token1);
        NT=new EqvService("NLPTrans" ,  token2);
        TT=new EqvService("TextTrans",  token3);
        this.addEqvService(LT, NT, TT);
//If no DNS support available, execute ``LT" by default
        this.setDefaultService(LT);
//Construct HTTP requests with given inputs
        LT.connectInput((sentence, lang)->{...});
        NT.connectInput((sentence, lang)->{...});
        TT.connectInput((sentence, lang)->{...});
//Process HTTP responses and return result
        LT.connectOutput(()->{... return result});
        NT.connectOutput(()->{... return result});
        TT.connectOutput(()->{... return result});}
//Implement interface Input to get user's input data
    @Override
    public void Input(String... s){
        sentence=s[0];lang=s[1]; } }

//Example of invoking TransServ
TransServ client = new TransServ();
//Input: Translate ``hello world" to ``Spanish"
client.Input("Hello World", "es");
//Output result: ``Hola Mundo''
String result = client.invoke();
```

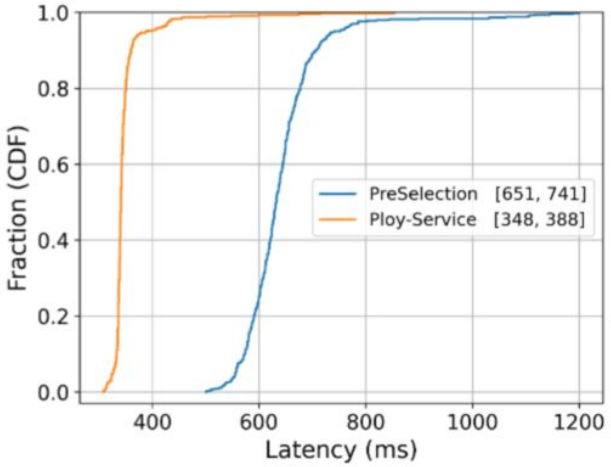
Reference Implementation



Evaluation Results:



(a) APP 1: Weather



(b) APP 2: Translation

Evaluation Results

QoS Metric	Failure (%)			Cost			Average Latency (ms)			Tail Latency (ms)		
Paradigm	**Pre**	**Opt.**	**ALL**	**Pre**	**Opt.**	**ALL**	**Pre**	**Opt.**	**ALL**	**Pre**	**Opt.**	**ALL**
Weather	0.00	0.00	0.00	1.00	1.40	3.00	275.55	266.63	257.99	417.63	389.06	387.06
Translation	0.12	0.00	0.00	1.00	1.71	3.00	560.64	474.77	469.74	778.38	589.45	586.45
Face Det	1.06	0.01	0.00	1.00	1.85	3.00	1272.88	1096.77	1093.74	1460.37	1343.54	1335.95
Flight	0.00	0.00	0.00	1.00	1.60	3.00	1178.66	1067.70	1063.34	1285.80	1243.2	1228.62
Hotel	0.00	0.00	0.00	1.00	1.40	3.00	727.47	714.50	712.17	1157.87	1048.52	1037.55
IP2Loc	0.01	0.00	0.00	1.00	1.60	3.00	80.00	54.30	54.13	123.60	92.40	92.20
Avg.	0.20	0.00	0.00	1.00	1.59	3.00	682.53	612.45	608.51	870.60	784.36	777.97

Scenario	Latency
Regular App + Unmodified DNS	1.4744ms
Regular App + Modified DNS	1.4972ms
PSF App + Unmodified DNS	1.5309ms
PSF App + Modified DNS	1.5351ms

Review

The paper presents "service polymorphism" and evaluates an implementation of that idea. Service polymorphism (S3) is the idea that a client application may potentially invoke any acceptable instance of a given kind of web service, e.g., "weather forecast" or "language translation," and can improve the QoS that it attains by dynamically selecting the service instances (one or more) to which it sends its requests. The work is motivated by a measurement study that shows that service-response latency varies across instances of a service, across client locations, and over time (S2). The paper presents a model (S3.3, S3.4) for a client to choose the set of services instances to which it will send a request, based on recent measurements of the instances' reliability and latency and the cost of sending multiple requests. When multiple instances are chosen, copies of the request are sent concurrently, and the first response is accepted by the client. Service latency information is shared across nearby clients through extensions to DNS (S3.2): "[a] resolver aggregates the QoS samples [reported by clients], calculates the QoS of each service, and shares it" with clients and other nearby resolvers.

The paper then presents PSF (S4), an implementation of service polymorphism for Android, with the DNS components implemented within BIND and OpenWrt. The paper evaluates the potential benefits and costs of PSF's model based on service-trace data (collected in the measurement study) and summarizes (S5.1): the strategy implemented by PSF "can reduce [the client-perceived rate of request failure] from 0.2% to 0%, and reduce mean and tail latency [of client-perceived request responses] by 10.26% and 9.90% respectively for all clients, while invoking 59% more services." The paper also summarizes an experiment that compares the client-perceived response latency for applications making service requests with and without PSF. In brief, the clients utilizing PSF experienced reduced latency (S5.2): at the tail, when the instance utilized by the non-PSF application is already the fastest, and at points in the latency CDF when the instance utilized by the non-PSF application is not the fastest.

Review

Strengths

- The topic of the paper---improved QoS for clients of web-based services---is relevant to ATC.
- The paper presents a model that clients can utilize to dynamically select the instances of a general web service (e.g., "weather forecast") to which the client will send a request, in order to minimize the latency of receiving a response (i.e., improve QoS). The model is driven by the measured latency and reliability of recent requests from "nearby" clients.
- The model uses local DNS servers to maintain QoS information about service instances and share it among "nearby" clients. This extension of the DNS infrastructure is clever.
- The model, called "service polymorphism," is implemented by a system called PSF.
- The paper evaluates the benefits of the model and PSF. In brief, PSF (1) decreases client-perceived latency, (2) often sends a single request to multiple instances, and (3) imposes reasonable resource costs (CPU, memory, latency) on local DNS servers.

Weaknesses

- A primary way that service polymorphism reduces client-perceived latency is by sending requests to multiple service instances. This is a potentially high cost on the service instances (S5.1, "invoking 59% more services") for a potentially small reduction in client-perceived latency (S5.1, "reduce mean and tail latency by 10.26% and 9.90% respectively")---all for services without clearly defined SLOs.
- Modeling the latency of web services with exponential distributions (S3.3) is not clearly well-justified. The collected measurement data (S2.3, Figure 3) suggests that the latency of a service for a given client and time frame is is not exponentially distributed from some minimum time

Review

Comments for authors

The idea of enhancing mobile applications to be able to consult any of several "equivalent" web services, called "service polymorphism" in this paper, is interesting, relevant to ATC, and of obvious potential benefit to the users of those applications. If one service instance is slow or unavailable, the ability to consult a different service instance allows the application to maintain high utility. The focus of this paper is on choosing the service instances---one or more---that will reduce the latency of request responses, but as the paper also points out, service polymorphism is also useful for improving client-perceived service reliability.

The idea of utilizing the edge DNS infrastructure for sharing latency information among "nearby" clients is clever in my opinion, and novel to my knowledge. While it seems to me that a client could do without this infrastructure---by simply doing some initial and periodic probing on its own, as mentioned by the paper in S7 (references 29 and 31)---it is interesting to leverage the DNS for sharing latency data among nearby clients, potentially eliminating the need for a client to send its own probes.

A primary way that service polymorphism reduces client-perceived latency is by sending single requests to multiple service instances, i.e., asking multiple services for the same information. The client accepts the first response it receives. The general idea of issuing multiple copies of a request to reduce latency and improve reliability is a well-known technique in distributed systems (e.g., Zaharia et al., "Improving MapReduce Performance in Heterogeneous Environments," OSDI 2008). Because this technique can impose potentially high costs on the servers and the network, however, it should be used sparingly by clients.

Cost/Benefit Trade-off

This leads me to wonder about the costs and benefits of the method presented in the paper. In S5.1, the paper summarizes an analysis of the method over collected service trace data: the method that "[invokes] one or multiple services with the best QoS based on our QoS comparison algorithm" can "reduce failure from 0.2% to 0%, and reduce mean and tail latency by 10.26% and 9.90% respectively for all clients, while invoking 59% more services." To me, this seems like a high cost---a great increase in load at the service instances---to pay for a relatively small benefit to the client. As shown in Table 1, the latencies of requests to pre-selected "best" services are already low, e.g., an average of 275ms for a weather request. The method is the paper can reduce the average to 266ms, at a cost of making 40% more requests to weather services. Is this a good cost/benefit trade-off? Is it a good cost/benefit trade-off for the other types of services?

The answer to whether or not a trade-off is "worthwhile" is very contextual, of course---worthwhile to whom, what is the penalty for greater latency, which resources are limited, etc.---but I make two observations from my understanding of the paper.

The first is that, for the services studies in the paper, the apparent benefit of lower latency to the client is minimal. E.g., I don't see that it matters very much to the user that the average latency of flight-info requests is reduced from 727.47ms to 714.50ms.

The second is that, if I understand correctly, most of the latency-reduction benefit of the paper's method can be had for less impact on the services through initial client-side server selection, i.e., probing (S7). I.e., if a client simply probes to find the best current service and then uses that service only, it can get good average response latency while probably paying some cost in tail latency. This is basically the situation shown in Figure 9a. A client can periodically probe to re-evaluate its service-instance choice, possibly triggered by observed changes in its context or other events. It is not hard to imagine that the overhead of initial and/or periodic probing, in terms of extra requests sent to services, would be less than the 40-85% overhead (Table 1) reported for the method presented in this paper. One could also imagine leveraging the edge DNS for clients to share recent probing data, reducing the need for each client to probe.

The apparent high cost (to services) and low benefit (to clients) of the paper's method is the reason why I recommend that this paper not be accepted to ATC.

Preparing for CAREER

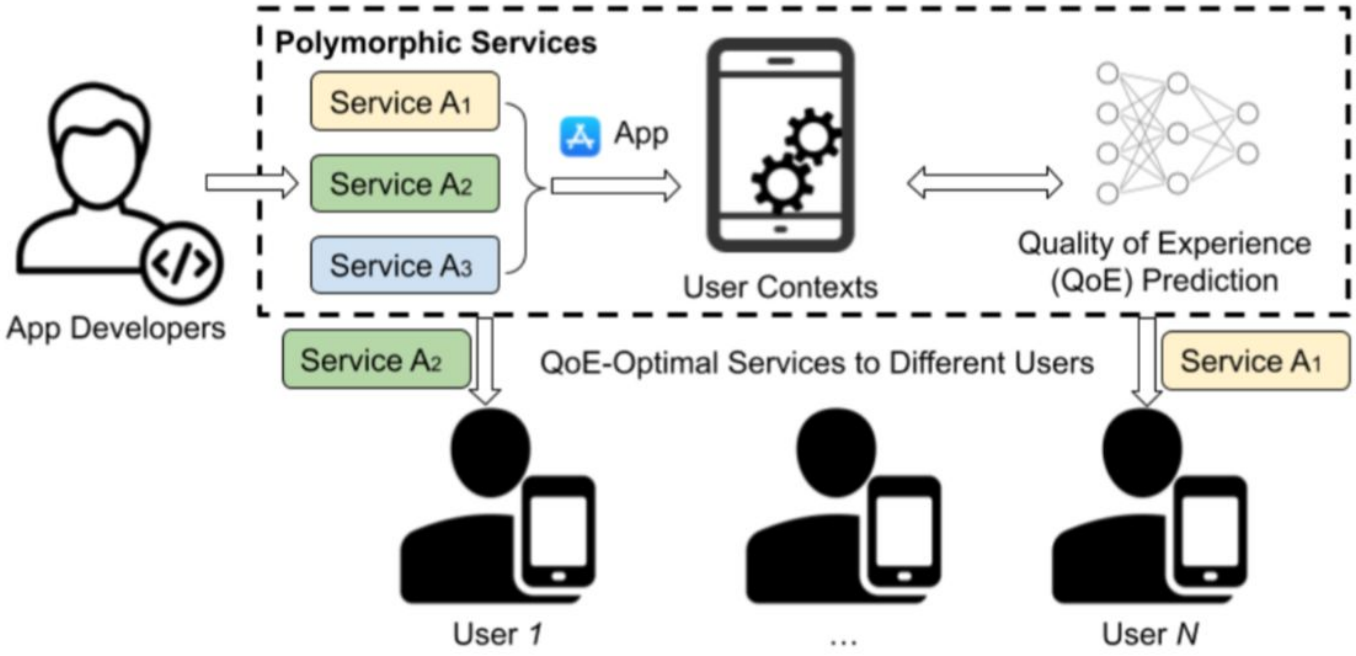
Polymorphic Service:

1. Web Service for QoS Enhancement.
2. Web Service for QoE Enhancement
3. IoT Services

SHF:Small: Polymorphic Services: Toward Socially Sustainable Service-Based Software

This proposal addresses a lack of social sustainability in service-based software by creating a novel development framework referred to as \emph{polymorphic services}. Socially sustainable software equitably delivers the same functionality to a diverse set of users, irrespective of their dissimilar individual traits, including gender, race, national origin, and socioeconomic status. For example, facial image recognition should work equally well for users that come from all racial backgrounds. However, some facial recognition services have been observed as discriminating against Black users by delivering much lower accuracy to them than to other users. The proposed polymorphic services would solve this problem by delivering the same facial recognition services to the target users by different means, thereby eliminating the aforementioned discrepancy in accuracy.

Workflow


Polymorphic Services
Service A1
Service A2
Service A3
App
User Contexts
Quality of Experience (QoE) Prediction
App Developers
Service A2
QoE-Optimal Services to Different Users
Service A1
User 1
...
User N