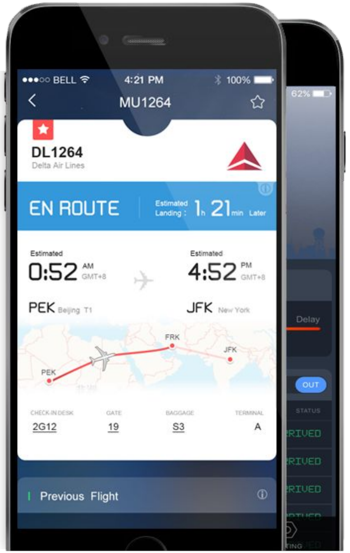
Service Polymorphism: Serve Distributed End Users Dissimilarly to Improve Performance for Service-Oriented Applications

Student: Zhengquan Li
Advisor: Dr. Zheng Song
Committee: Drs. Bacha, Esthete, and Xu.
CIS @University of Michigan, Dearborn

Web Services are Widely Used in Real Life

- Weather Forecast, Face Recognition, Flight Information…
- Wide used in various applications



QoS (Quality of Service) matters!E.g., Latency, reliability…

- Amazon: “Every 100ms of latency costs 1% in sales”

- TABB Group: “Broker could lose as much as $4 million in revenues per millisecond if its electronic trading platform was only 5ms behind the competition”

- Google: “Extra 500ms in search page generation time dropped traffic by 20%”

How Current Service Invocation Paradigm Guarantee QoS?

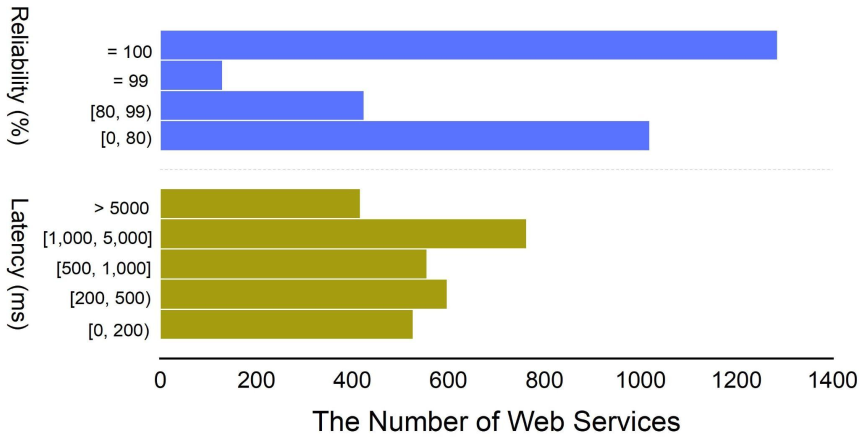
1. Preselect one service and Hardcode it into the application
2. Rely totally on SLA (Service Level Agreement) to guarantee the service' QoS

```python
def bets():
    import requests
    url = "https://api-football-v1.p.rapidapi.com/v3/odds/bets"
    headers = {
        'x-rapidapi-host': "api-football-v1.p.rapidapi.com",
        'x-rapidapi-key': TOKEN
        }
    response = requests.request("GET", url, headers=headers)
    responsep = response.json()['response']
    for x in range(0,len(responsep)):
        print(responsep[x])
```

```javascript
const BASE_URL = `https://youtube-v31.p.rapidapi.com`;


export default function fetchFromApi(uri) {

    //приема динамичен uri, който се закача на BASE_URL и ползва горните options
    return fetch(BASE_URL + uri, options).then((res) => {
        if (!res.ok) {
            //ако статуса е в неуспешния диапазон - 4xx - 5xx, хвърли тази грешка
            throw new Error(`Error: ${res.status}. Message: ${res.message}`);
        }
        return res.json();
    });


}
```

Problem: Current Paradigm Cannot Guarantee QoS

- Study Object: RapidAPI, the largest web services hub in the world
- **Observation 1**: Only 3 services out of 5784 on RapidAPI provide QoS guarantees in terms of SLA.
- **Observation 2**: The services' QoS is not satisfactory



Web Services' Latency/Reliability from RapidAPI

Opportunity: Widely Existing Equivalent Services

- Equivalent Services: The services with same functionality but different QoS

Weather APIs

Browse the best premium and free APIs on the world's largest API Hub. Read about the latest API news, tutorials, SDK documentation, and API examples. RapidAPI offers free APIs all within one SDK. One API key. One dashboard.

Open Weather Map

Get weather and weather forecasts for multiple cities.

Verified ✓

0.5 | 56 ms | 100%

Visual Crossing Weather

Visual Crossing Weather API provides instant access to both historical weather records and weather forecast data

9.8 | 368 ms | 100%

WeatherAPI.com

WeatherAPI.com is a powerful fully managed free weather and geolocation API provider that provides extensive APIs that range

9.9 | 265 ms | 98%

Weather

Current weather data API, and Weather forecast API - Basic access to the Weatherbit.io Weather API.

9.9 | 327 ms | 100%

Open Weather

Get weather data for any location on the globe immediately with our superb API

9.6 | 1,075 ms | 94%

Top Hotel APIs

Top Image Search and Image Recognition APIs

Top Proxy APIs

Top Transportation APIs

Best Text to Speech APIs

Opportunity: Equivalent Services’ QoS

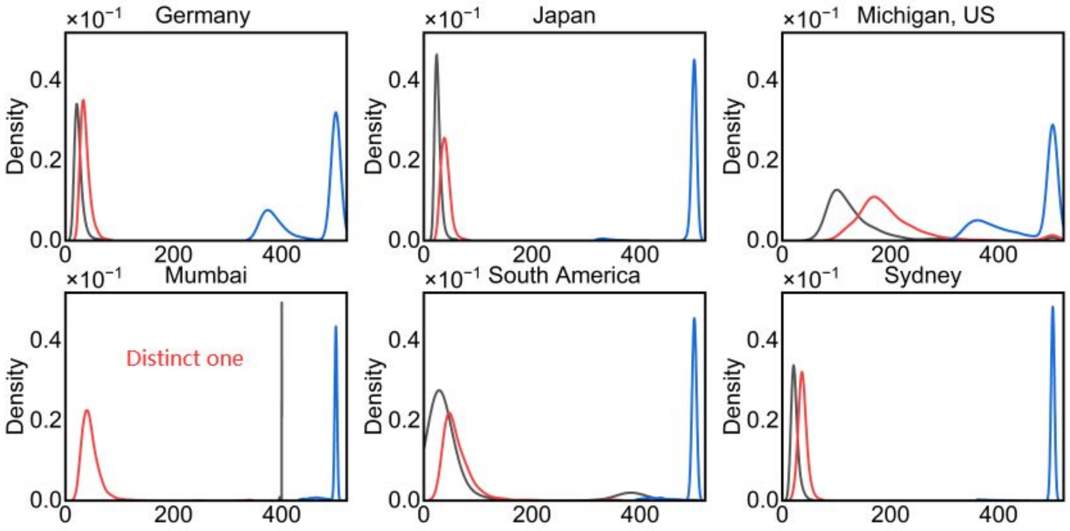
- Study Setup: Invoking six set of equivalent services in seven locations

			Germany	Tokyo	Sydney	Mumbai	Michigan	South America	Silicon Valley
Functionality	*Service Name*	*Runs*	*Avg1, Std1*	*Avg2, Std2*	*Avg3, Std3*	*Avg4, Std4*	*Avg5, Std5*	*Avg6, Std6*	*Avg7, Std7*
Weather	OW	32k	89, 324	337, 268	405, 239	261, 314	283, 269		
	VC	32k	434, 339	747, 296	952, 403	1322, 2036	385, 435		
	WO	32k	382, 65	712, 61	917, 136	818, 134	296, 187		
Trans.	LT	37k	389, 325	606, 2586	754, 272	640, 228	395, 337	453, 321	684, 759
	NT	37k	428, 762	758, 2498	941, 2722	1185, 155	199, 73	367, 99	545, 465
	TT	37k	552, 444	948, 9177	987, 812	1221, 465	300, 407	498, 1269	709, 644
FaceDet.	ID	36k	1590, 358	1763, 3356	1723, 307	1994, 274	2539, 19K	1557, 300	1591, 291
	MS	36k	1143, 1621	422, 1606	1681, 1709	1606, 508	1241, 3370	731, 136	1084, 208
	SC	36k	964, 215	1803, 7820	1664, 243	1490, 242	2171, 19K	1523, 261	1720, 299
IP2Loc.	IG	32k	28, 117	27, 35	26, 65	534, 52	148, 171		
	IL	32k	39, 62	41, 33	56, 1243	49, 48	213, 174		
	IT	32k	645, 327	650, 487	797, 310	977, 345	545, 258		
Flight	FR	630	3687, 385	4197, 690	2697, 712	6292, 607	2475, 434		
	FT	630	344, 260	1432, 378	1711, 206	1524, 225	880, 246		
	FA	630	1279, 505	1405, 704	1595, 928	1857, 812	1180, 710		
Hotel	BC	450	380, 575	730, 554	858, 833	908, 840	760, 403		
	GS	450	1579, 1329	1859, 1602	2076, 1807	2371, 1395	1599, 1642		
	PL	450	2092, 886	2435, 895	2390, 1021	2525, 1056	2153 ,660		

Table 1. The Detailed Statistics of Web Service Invocations

Opportunity: Equivalent Services' QoS

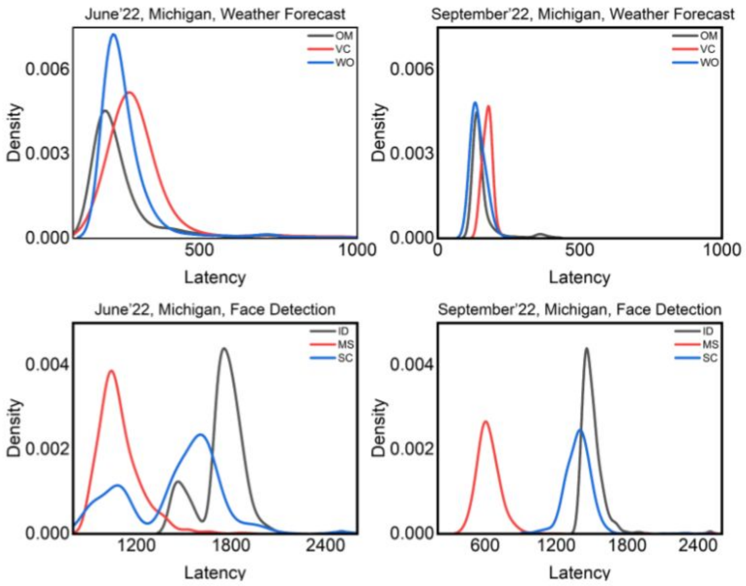
Observation 1: Among a set of equivalent services, the QoS ranking could change over different locations.



Latency Distribution of IP2Location APIs (different colors represent different web services)

Opportunity: Equivalent Services' QoS

Observation 2: Among a set of equivalent services, the QoS ranking could change over different time.



Services' Latency Distribution in Different Time

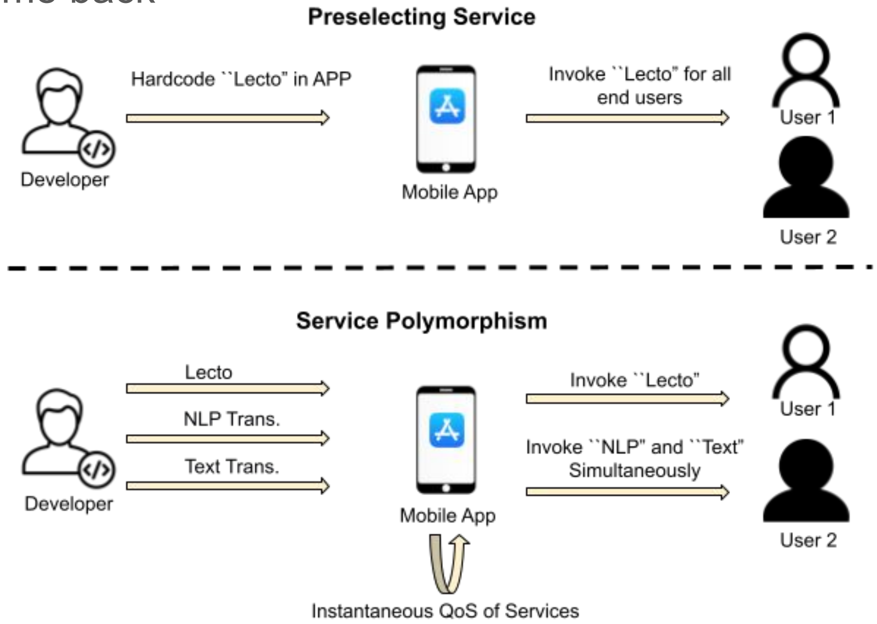
Opportunity: Equivalent Services' QoS

- **Observation Summary**: For a set of equivalent services, their QoS rankings could fluctuate over spatial and temporal context, making it impossible to pre-select one service that consistently outperforms its peers in different invocation contexts.

- **Inspiration: Leverage the dynamics and complementation of equivalent services' QoS**
  - ❑ Assign multiple equivalent services in the applications
  - ❑ Invoke one or more services to provide optimal QoS according to the context

Quick View of Service Polymorphism Philosophy

- **Invoke different service(s) for users in different context (location/time).**
  - ❑ Invocation strategy: A, B, C, A*B, A*C, B*C, A*B*C
  - ❑ Speculative parallel (*): Simultaneously invoke multiple services and fetch the result that first come back



Outline

- Introduction
- **Design**
  - **Challenges**
  - **Solutions**
- **Implementation**
- **Evaluation**
- **Conclusion**

Challenge #1: Find the QoS-optimal Invocation Strategy

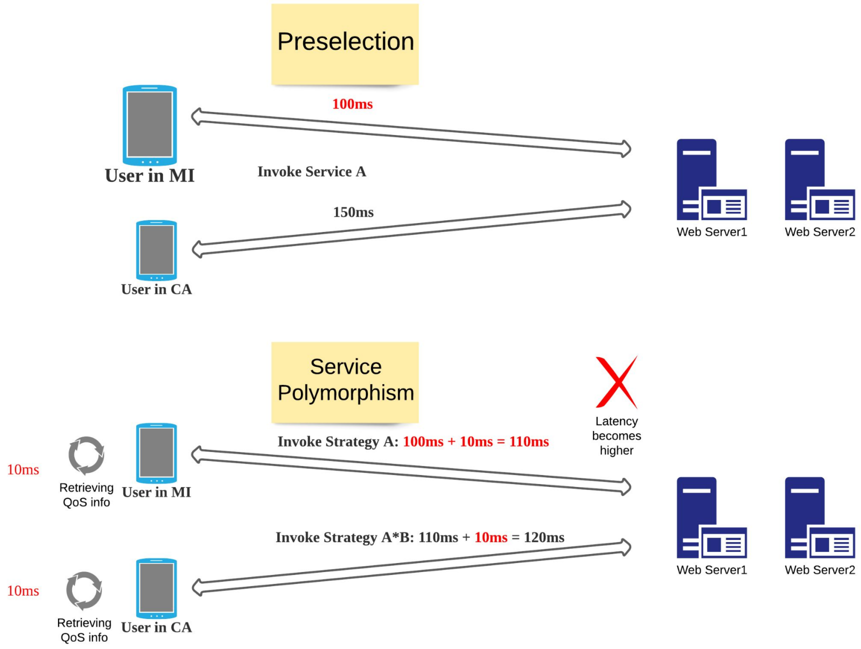
- **QoS is a multi-dim vector: latency (L), reliability (R), cost (C)**

- **Calculate the QoS of the invocation strategy?**
    - Given QoS: L, R, C of A, B, C
    - A*B, A*C, B*C, A*B*C
    - What is L(A*B) … L(A*B*C) ?
    - What is R(A*B) … R(A*B*C) ?
    - What is C(A*B) … C(A*B*C) ?

- **Choose the QoS optimal strategy by comparing the QoS of candidates**

Challenge #2: Minimize Overhead

- Finding optimal strategy requires QoS information of each services
- The overhead of retrieving QoS should be minimum



Outline

- Introduction
- Design
  - Challenges
  - Solutions
- Implementation
- Evaluation
- Conclusion

Estimating the QoS of Combined Invocation

- **Assuming there are n equivalent services**
  - Reliability: $\{R_1, R_2, \ldots, R_n\}$
  - Latency: $\{L_1, L_2, \ldots, L_n\}$
- **Reliability of Combined Invocation:**
  - The speculative parallel invocation of n services only fails when all constituent services fail

$$R = 1 - \prod_{i}^{i=n} (1 - R_i)$$

- **Cost of Combined Invocation:**
  - Invoking more services, issuing more http requests, leads to more network traffics
  - We consider the number of services as the cost
  - C=|*W*|, where *W* represents the services in the invocation strategy

Estimating the QoS of Combined Invocation

- The service latency should be a distribution []
- Latency: $\{L_1, L_2, \ldots, L_n\}$
- As speculative parallel pick the results that comes back first, so
  - $L_c = \min\{L_1, L_2, \ldots, L_n\}$
  - The distribution function of L_c :

$$\begin{aligned}\mathbf{P}\{L_c \le x\} &= \mathbf{P}\{\min\{L_1, L_2, \ldots, L_n\} \le x\} \\ &= 1 - \mathbf{P}\{\min\{L_1, L_2, \ldots, L_n\} > x\}\end{aligned}$$

After some algebra manipulations

$$\begin{aligned}&\mathbf{P}\{\min\{L_1, L_2, \ldots L_n\} > x\} \\ &= \mathbf{P}\{L_1 > x, L_2 > x, \ldots, L_n > x\} \\ &= \mathbf{P}\{L_1 > x\} \times \mathbf{P}\{L_2 > x\} \times .. \times \mathbf{P}\{L_n > x\} \\ &= \prod_{i=1}^{n} \mathbf{P}\{L_i > x\} = \prod_{i=1}^{n} (1 - \mathbf{P}\{L_i \le x\})\end{aligned}$$

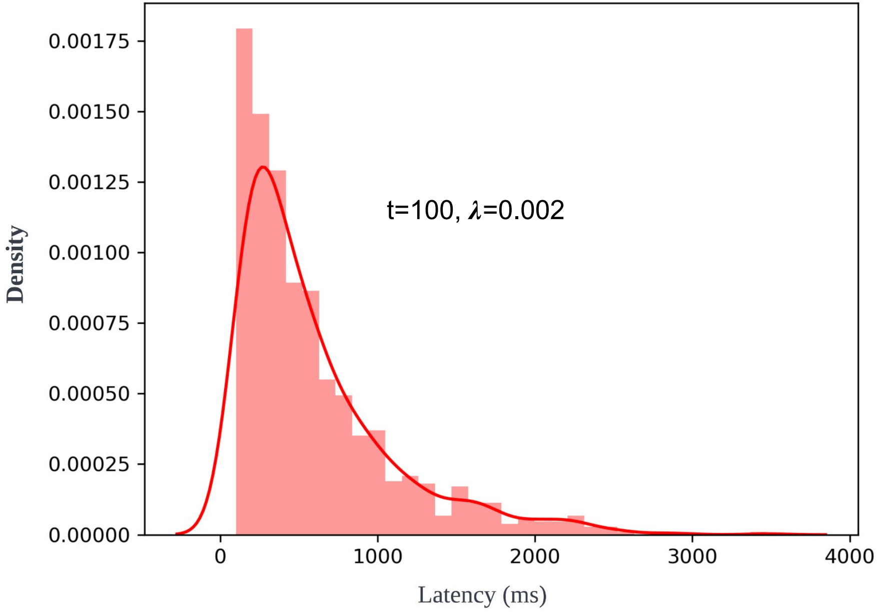
$$\overbrace{\mathbf{P}\{L_c \le x\}}^{\text{CDF of Combine Invocation}} = 1 - \prod_{i=1}^{n} (1 - \overbrace{\mathbf{P}\{L_i \le x\}}^{\text{CDF of each constituent service}})$$

Estimating the QoS of Combined Invocation

- **Approximating each services' latency: two-parameter exponential distribution**

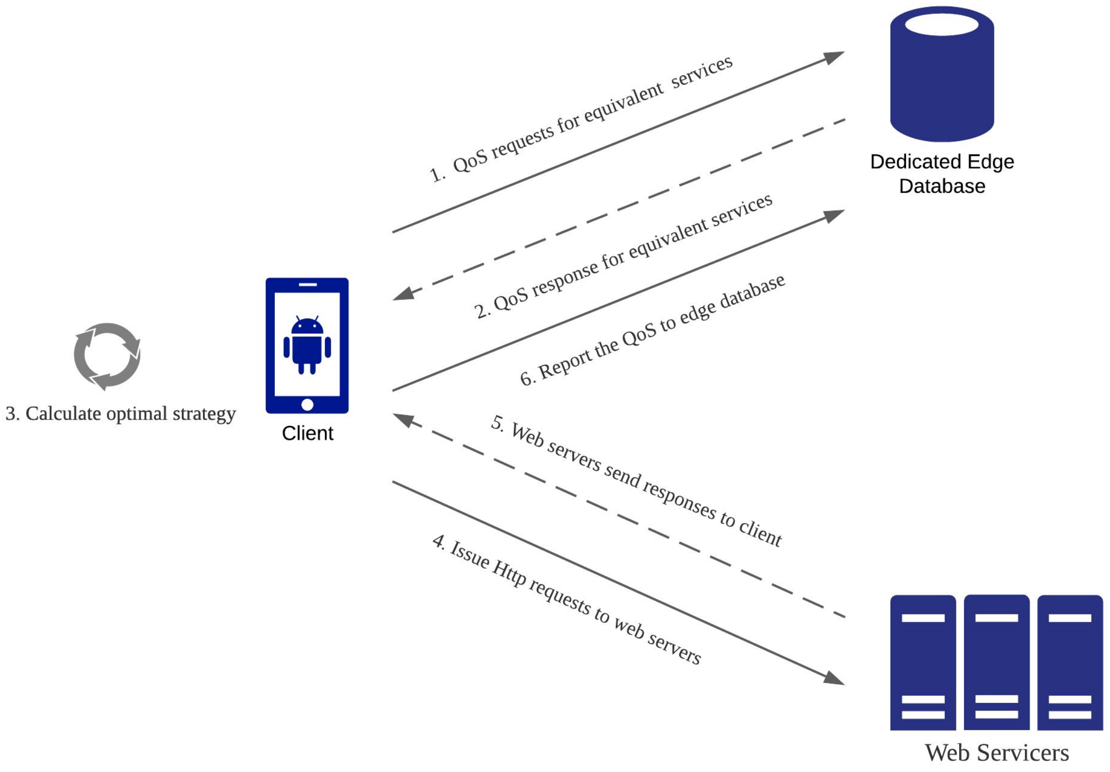
$$f_i(x) = \begin{cases} 0, & x < t \\ \lambda e^{-\lambda(x-t)}, & x \geq t \end{cases}$$

$$P\{L_i \leq x\} = F_i(x) = \begin{cases} 0, & x < t \\ 1 - e^{-\lambda(x-t)}, & x \geq t \end{cases}$$

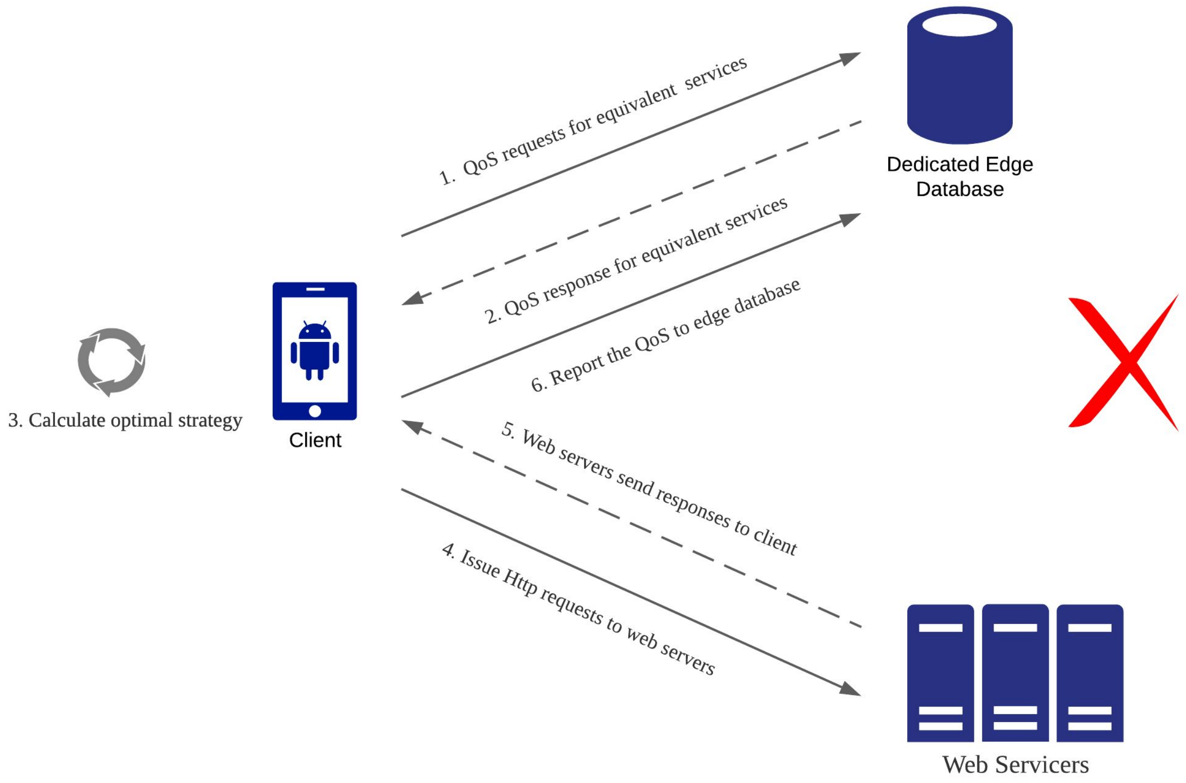


- Why choose exponential distribution ?
  - Fitness
  - Computation friendly

Store and Retrieve QoS information: Edge Database



Store and Retrieve QoS information: Edge Database



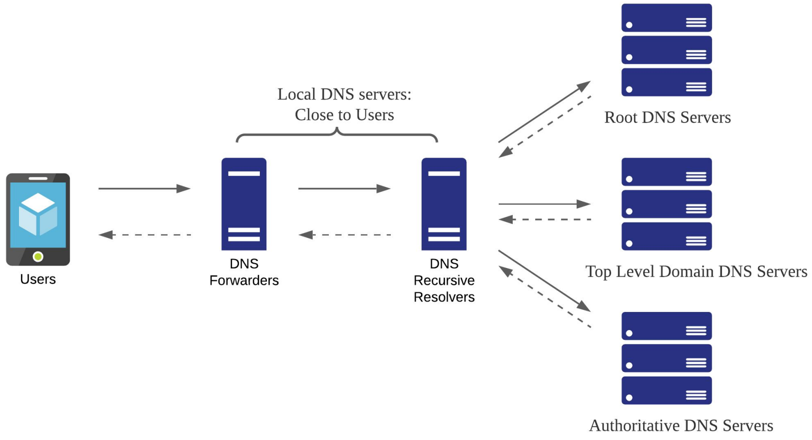
1. Much Overhead
2. Deploy extra infrastructure

Piggyback QoS onto DNS packet to Minimize Overhead

Inspiration source: All the http/https requests involve the DNS procedure

Web Server

HTTP(s) Process



DNS Infrastructure with Multiple Layers

Piggyback QoS onto DNS packet

- Overview: Use Local DNS servers to preserve QoS information and piggyback it onto DNS packet

```
              +---------------------------------------------+
 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>|
              +---------------------------------------------+
```

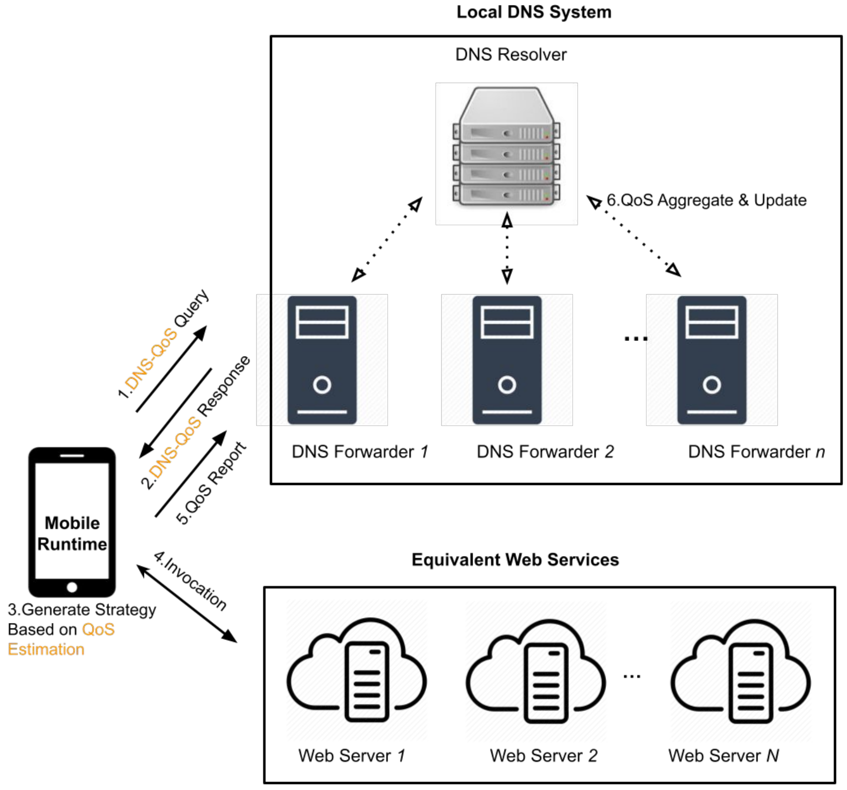
DNS packet format

```
<NAME>     ::= NULL
<TYPE>     ::= QOS
<CLASS>    ::= QUERY/RESPONSE
<TTL>      ::= NULL
<RDLENGTH> ::= The Length of RDATA
<RDATA>    ::= <CLASS> == QUERY ? MD5(URL) : <Latency, Reliability>
```

Additional Record Format/Definition for QoS Query/Response

Piggyback QoS onto DNS packet

- Born with context (local dns servers geographically close to users)
- No cost to deploy extra infrastructures
- No extra overhead for interacting with other infrastructures



Outline

- Introduction
- Design
  - Challenges
  - Solutions
- Implementation
- Evaluation
- Conclusion

Implementation: on Android

- **Programming model: Ploy-Service**
  - Native library (*2500 lines of code*): modify *c-ares* DNS library to assemble and parse DNS-QoS query/response
  - Java library (*1000 lines of code*): programming interface and strategy generation

```
//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();
```

Invoking TransServ Poly-Service

```
// Define TransServ as Poly-Service
Class TransServ extends PolyService{
//TransServ input: (sentence, target language)
//TransServ output: Translation result
    String sentence;
    String 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, 1);
        NT=new EqvService("NLPTrans" ,  token2, 5);
        TT=new EqvService("TextTrans",  token3, 10);
        this.addEqvService(LT, NT, TT);
//If no DNS support available, execute ``GT" by default
        this.setDefaultService(GT);
//Construct HTTP requests with given inputs by lambda
      funcs
        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]};
}
```

Implementing Translation Poly-Service Suite

Implementation: on Android

- **Strategy Generation Algorithm Based on QoS Estimation**

Input: $\mathcal{M}$: Equivalent Web Services Set

Input: θ: Reliability Threshold

Output: IS: Invocation Strategy

$\mathcal{S} \leftarrow$ FindInvocationStrategies $(\mathcal{M})$

$\mathcal{T} \leftarrow$ AutoSortedSet$< Float, Float >, Strategy >>$

$\mathcal{T}$.SetComparator(CompareQoS)

for $i \leftarrow 0$ to $|\mathcal{S}| - 1$ **do**

  $c, l, r \leftarrow$ EstimateQoS$(\mathcal{S}(i))$

  if $r >= \theta$ **then**

    $\mathcal{T}$.put$(< c, l >, \mathcal{S}(i))$

  end if

end for

$IS \leftarrow \mathcal{T}$.first().value()

return IS

function COMPAREQOS$(< c_1, l_1 >, < c_2, l_2 >)$

  if $abs(l_1, l_2) < 5$ **then**

    $c_1 < c_2$? return -1 : return 1

  else

    $l_1 < l_2$? return -1 : return 1

  end if

end function

function ESTIMATEQOS(s:STRATEGY)$(\{c, l, r\})$

  $c \leftarrow 0, l \leftarrow 0, r \leftarrow 1, L \leftarrow \{\}$

  $c \leftarrow |s|$

  for $i \leftarrow 0$ to $|s| - 1$ **do**

    $L \leftarrow L \cup \{s(i).t, s(i).\lambda\}$

    $r \leftarrow r * (1 - s(i).r)$

  end for

  $l \leftarrow \Omega(L)$ ▷ Minimum of exponential distributions

  $r \leftarrow 1 - r$

  return c, l, r

end function

Implementation: on DNS servers

- DNS Forwarders: modify *Dnsmasq* DNS software
  - Mainstream router operating system, OpenWRT, use it as default DNS library
  - To support handle DNS-QoS query
  - To receive QoS reports
- DNS Resolvers: modify *BIND9* DNS software
  - Widely used in today's internet carriers, ISPs, various organizations
  - To aggregate QoS reports from forwarders
  - To disseminate the reports to nearby forwarders

BIND
Berkeley Internet Domain Name

1500 lines of C code for implementing the DNS server software

Outline

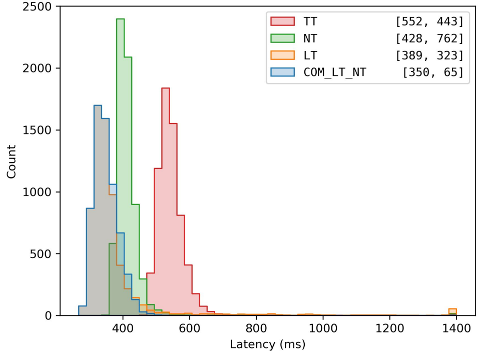
- Introduction
- Design
  - Challenges
  - Solutions
- Implementation
- Evaluation
- Conclusion

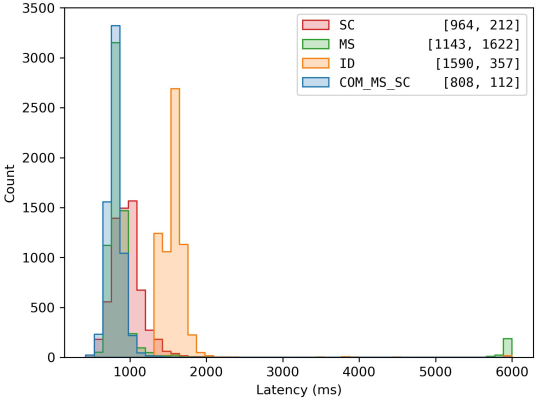
Trace-based Simulation

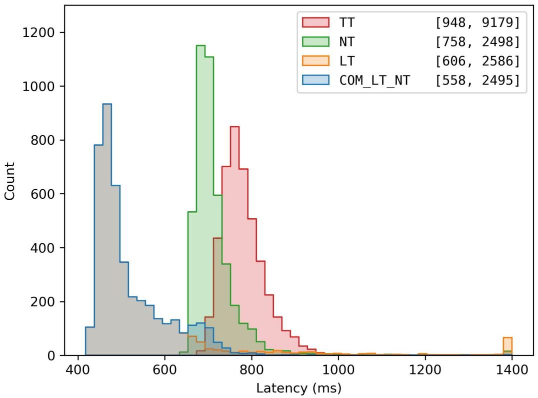
- **Combined invocation does improve QoS**
- **Invoke simply all services is not reasonable**

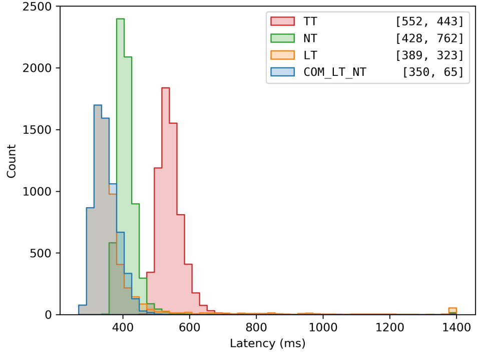
Invocation Strategy	Reliability	Cost	Latency	STD
Pre-selected	99.62%	1.00	100%	100%
Optimal	99.97%	1.97	85.3%	58.0%
Invoke-all	99.97%	3.00	85.2%	57.7%

Table 1. Performance of Different Invocation Strategies



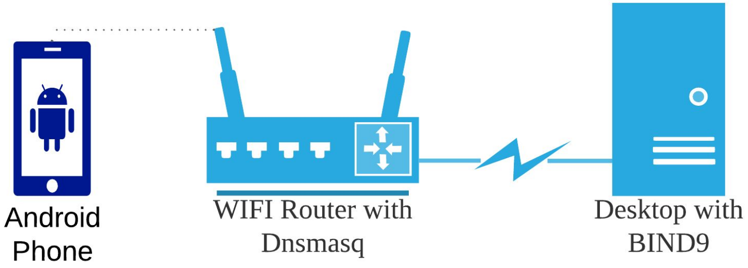






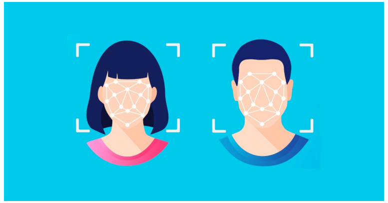
Poly-Service's Performance in Real Testbed

- User/Client: Android Mobile Smartisan R2
  - Android 11
  - Qualcomm Snapdragon 865 CPU and 16GB RAM
- DNS Forwarder: Home WiFi Router GL-MT1300
  - OpenWRT 21.02
  - MT7621A @880MHz CPU and 256MB Memory
- DNS Resolver: a Desktop
  - Ubuntu 16.04
  - Intel i7-4790 CPU @3.60GHz and 16GB Memory



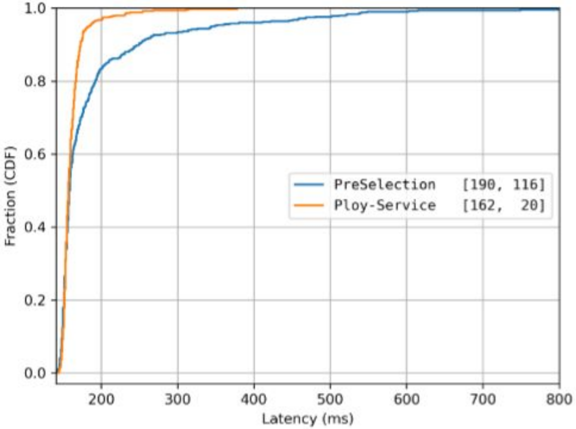
Poly-Service's Performance in Real Testbed

- **Developed three typical Android APPs**
- **Weather Forecast**
    - Three equivalent services: *OW, VC, WO*
    - Preselected: *OW*
- **Language Translation**
    - Three equivalent services: *TT, NO, LT*
    - Preselected: *LT*
- **Face Detection**
    - Three equivalent services: *ID, MS, SC*
    - Preselected: *MS*



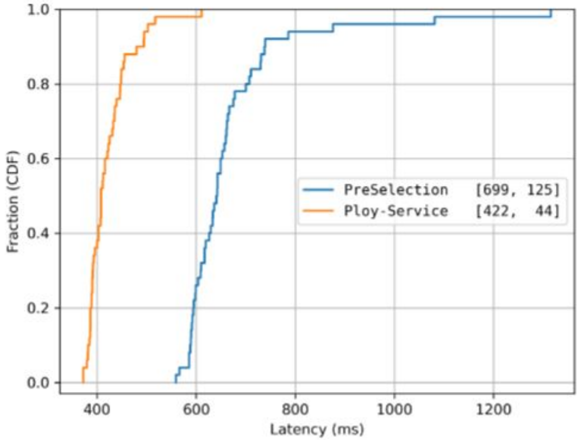
Poly-Service's Performance in Real Testbed

14% latency reduce, 〈 OW * WO 〉



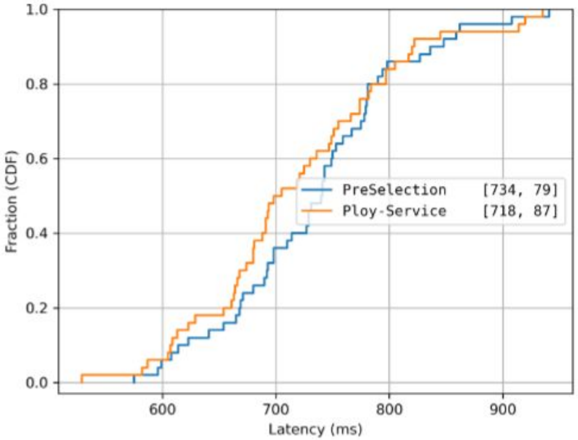
(a) APPs 1: Weather Forecast

36% latency reduce, 〈 NT * TT 〉



(b) APPs 2: Language Translation

2% latency reduce, 〈 MS * SC 〉



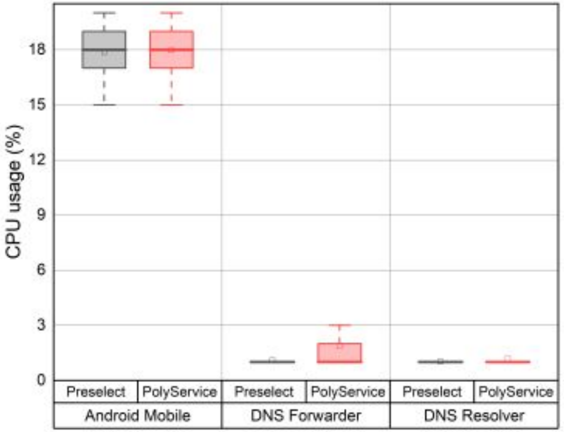
(c) APPs 3: Face Detection

Latency Distribution of APPs based on PreSelection and Ploy-Service

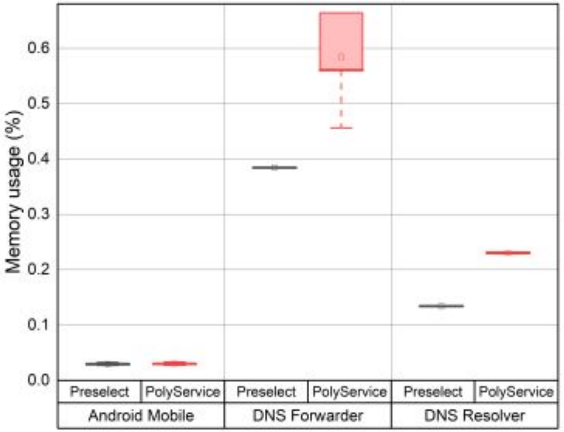
Compatibility & Overhead Assessment

Cases	Latency Overhead
Preselection-based APP to original DNS	1.26ms
PolyService-based APP to original DNS	1.30ms
PolyService-based APP to modified DNS	1.41ms

Table 2. The Latency Overhead for Different DNS Servers in the Wild



(a) CPU usage

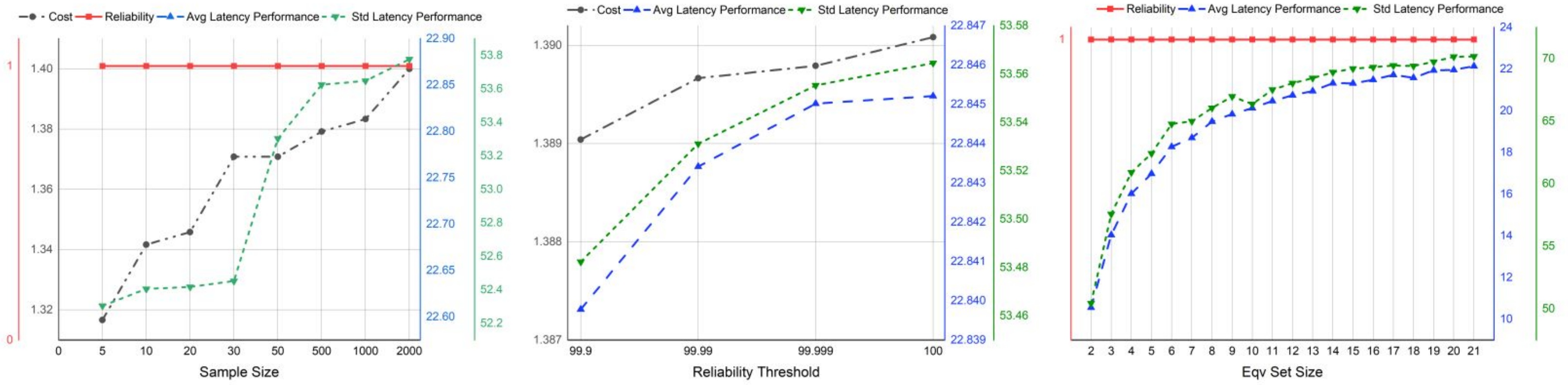


(b) Memory usage

The Resource Consumption of Poly-Service on Android Mobile, DNS Forwarder and DNS Resolver

Sensitivity Analysis

- Some key hyper-parameters in our solution:
  - QoS Sample Size, Reliability Threshold, Equivalence Set Size



(a) The Impact of Sample Size (b) The Impact of Reliability Threshold (c) The Impact of Equivalence Set Size

Poly-Service's Parameters' Impact on QoS Performance Compared to Preselection

Discussion

- **Previous works on modifying DNS provide a practical vision**
  - Microsoft Research: solve client-Local DNS mismatching
  - DEW: DNS as proxy to achieve HTTP interaction
  - ASAP: new internet protocol based on modify hostname of DNS packet
- **Privacy & Security considerations**
  - Our solution can co-exist with current DNS security mechanisms
    - E.g., DNSSEC, DNS over TLS, DNS over HTTPS
  - The security of QoS reporting and sharing can be improved by existed mechanisms
    - E.g., Reputation Model
- **Limitations**
  - The correlation between the services' latency. E.g., network congestion

Outline

- Introduction
- Design
  - Challenges
  - Solutions
- Implementation
- Evaluation
- Conclusion

Conclusion & Future Work

- **Current service invocation paradigm is hard to provide satisfactory QoS**
- **Service Polymorphism is a novel paradigm that deliveries better QoS**
  - Significantly Improve QoS, especially in latency performance
  - DNS piggybacking to minimize extra overhead
  - Carefully consider all the services' users worldwide
- **Future work directions**
  - More comprehensively study the web services' performance worldwide. E.g., using RIPE-Atlas
  - Can DNS be used to other purpose? E.g., caching

Thank you!