Systemizing and Mitigating Topological Inconsistencies in Alibaba's Microservice Call-graph Dataset

Darby Huye, *Lan Liu, and Raja Sambasivan*
Tufts University
ACM/SPEC ICPE 2024

Everything is microservices

Modern day distributed applications often built with microservices

- Decomposing applications' functionalities into many services that coordinate over well-defined APIs to process requests
- E.g. Google, Meta, Bytedance, Alibaba, etc.

Lots of interest in doing research on microservices

But, little is known about concrete characteristics of microservice deployments at companies…

Alibaba released microservice call graph datasets*

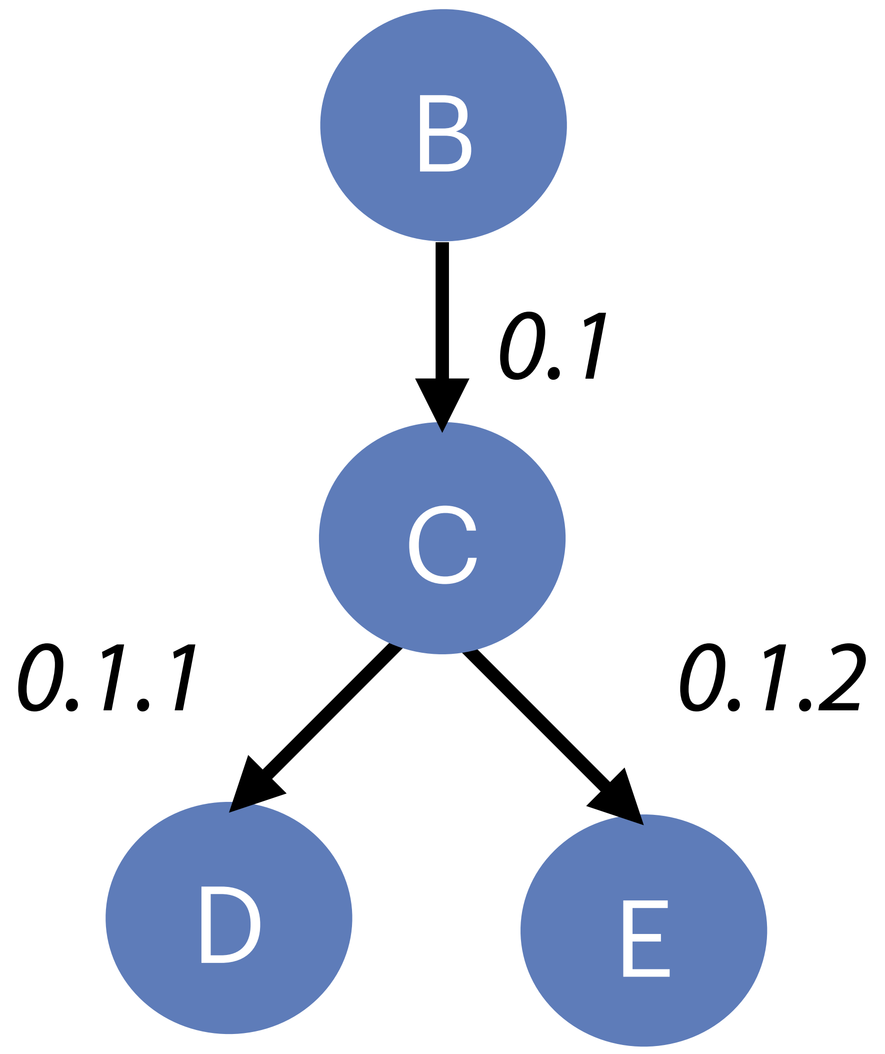
Call graphs are trees showing path of request being processed by services

Alibaba released two tabular call graph datasets:

- 2021: 12-hour time period, 20 million traces
- 2022: 13-day time period, ~13 billion traces

Datasets are popular:

- 1.5k stars on GitHub, 137 papers cite the original paper
- Comparing microservice architectures [Google SOSP '23], simulating realistic workloads and applications[Mbench] for profiling tail latency [Erms ASPLOS'23], testing QoS recovery mechanisms[Nodens ATC'23], and latency distribution predictions [LatenSeer]

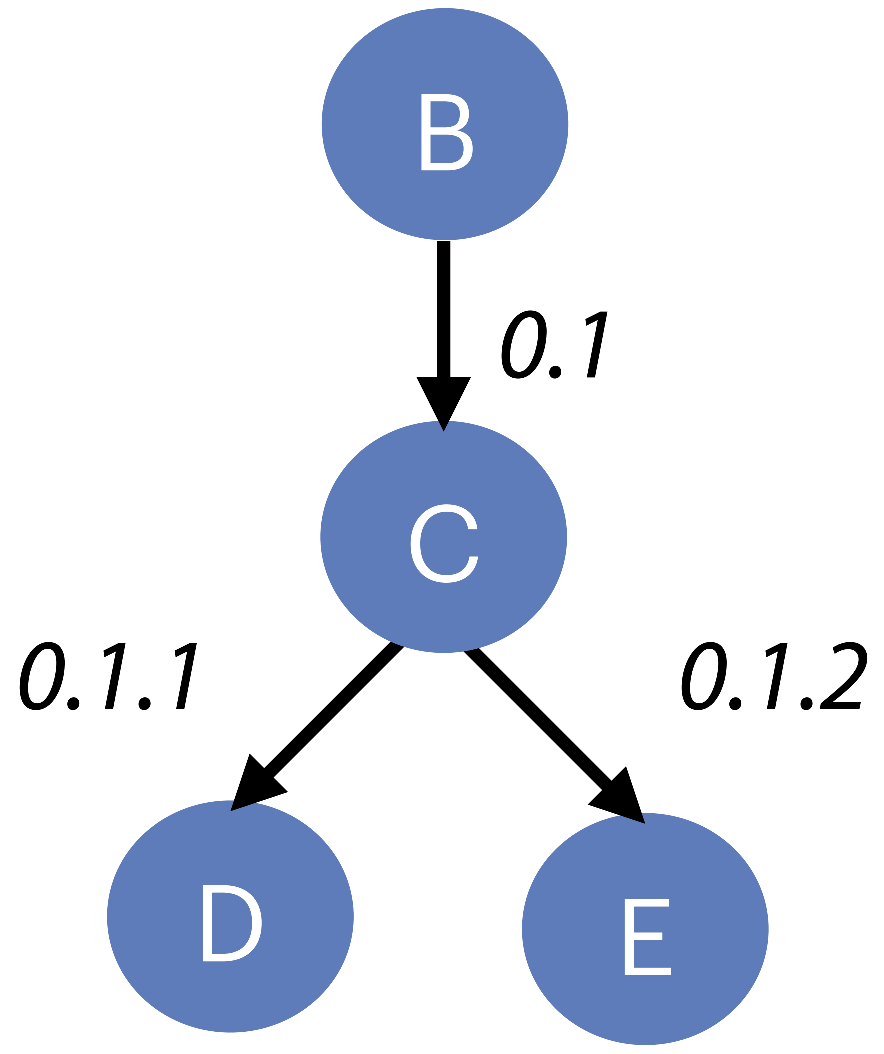


Example call graph

*https://github.com/alibaba/clusterdata/

Errors in the call graph datasets

- Many missing edges resulting in disconnected trace trees
- Incorrect edge identifiers (*rpcids)* prevent constructing the tree accurately



Example call graph

Key insight: hidden redundancies in their trace model that give us a starting point for fixing errors

Key Contributions

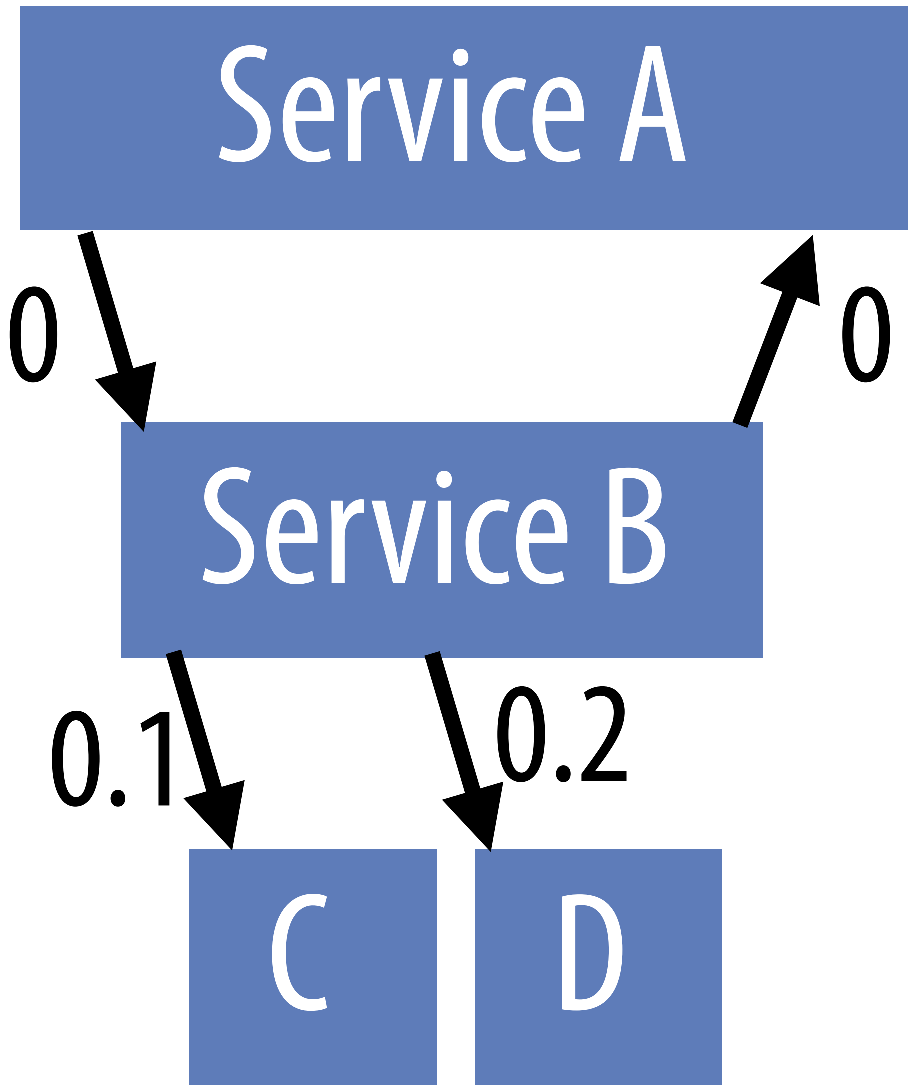
1. Identify categories of errors in the datasets
2. Method to use hidden redundancies in the dataset to recover from errors
3. Analysis of changes in the topological characteristics as a result of fixing errors
4. Released corrected trace data and code

Outline

- Introduction
- Using Alibaba's datasets
- Casper: Remedying errors using redundancies in trace model
- Evaluation
- Implications of errors in trace data

Alibaba's (assumed) data collection

Example trace



Upstream microservice *Downstream microservice* *Response time*

rpcid	UM	DM	rpctype	rt
0	A	B	http	8
0.1	B	C	db	0
0.2	B	D	db	0

call id given an ancestry

#.#.#

Delimeter between sequential calls

Type of call:
- Two-way: http, rpc
- One-way: db, mq, mc

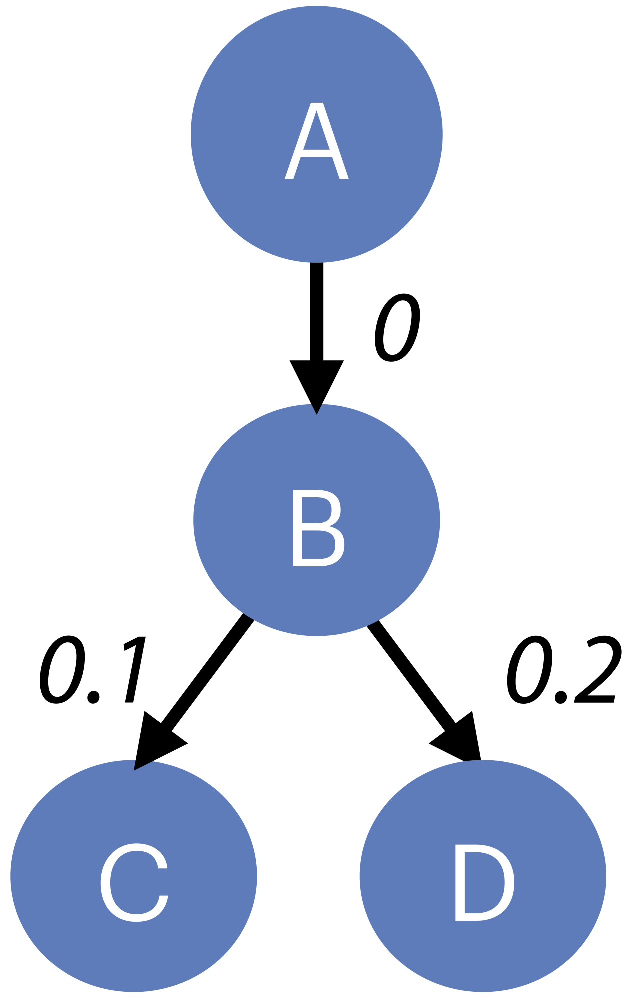
* focusing on 2021 dataset during this talk

Build a call graph for a trace

* in an ideal world

rpcid	UM	DM	rpctype	rt
0	A	B	http	8
0	A	B	http	-7
0.1	B	C	db	0
0.2	B	D	db	0

Example trace

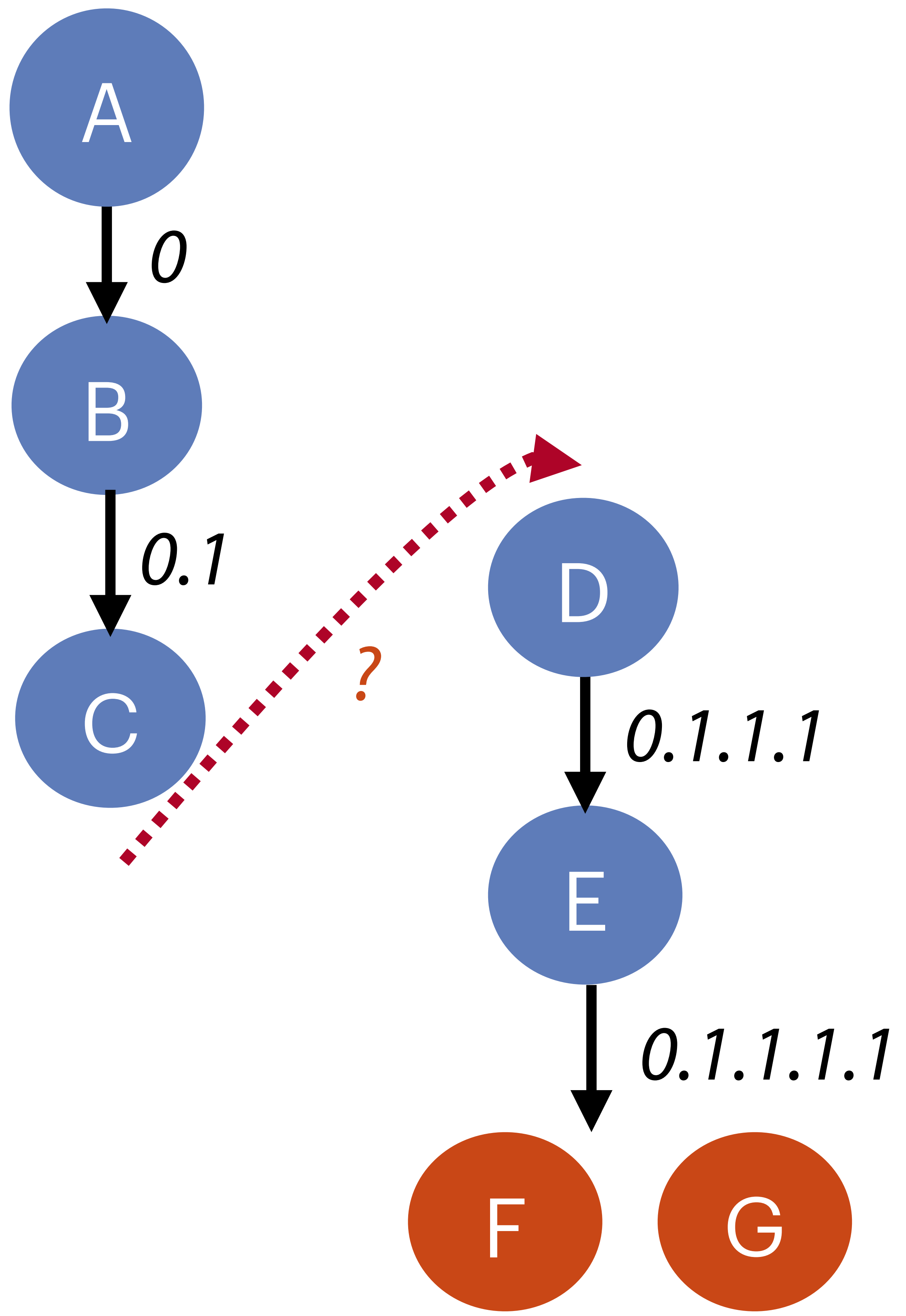


Example call graph

Build a call graph for a trace
* in the real world

rpcid	UM	DM	rpctype	rt
0	A	B	http	8
0	A	B	http	-7
0.1	B	C	mq	0
0.1.1.1	D	E	http	2
0.1.1.1.1	E	F	db	0
0.1.1.1.1	E	G	db	0

Example trace



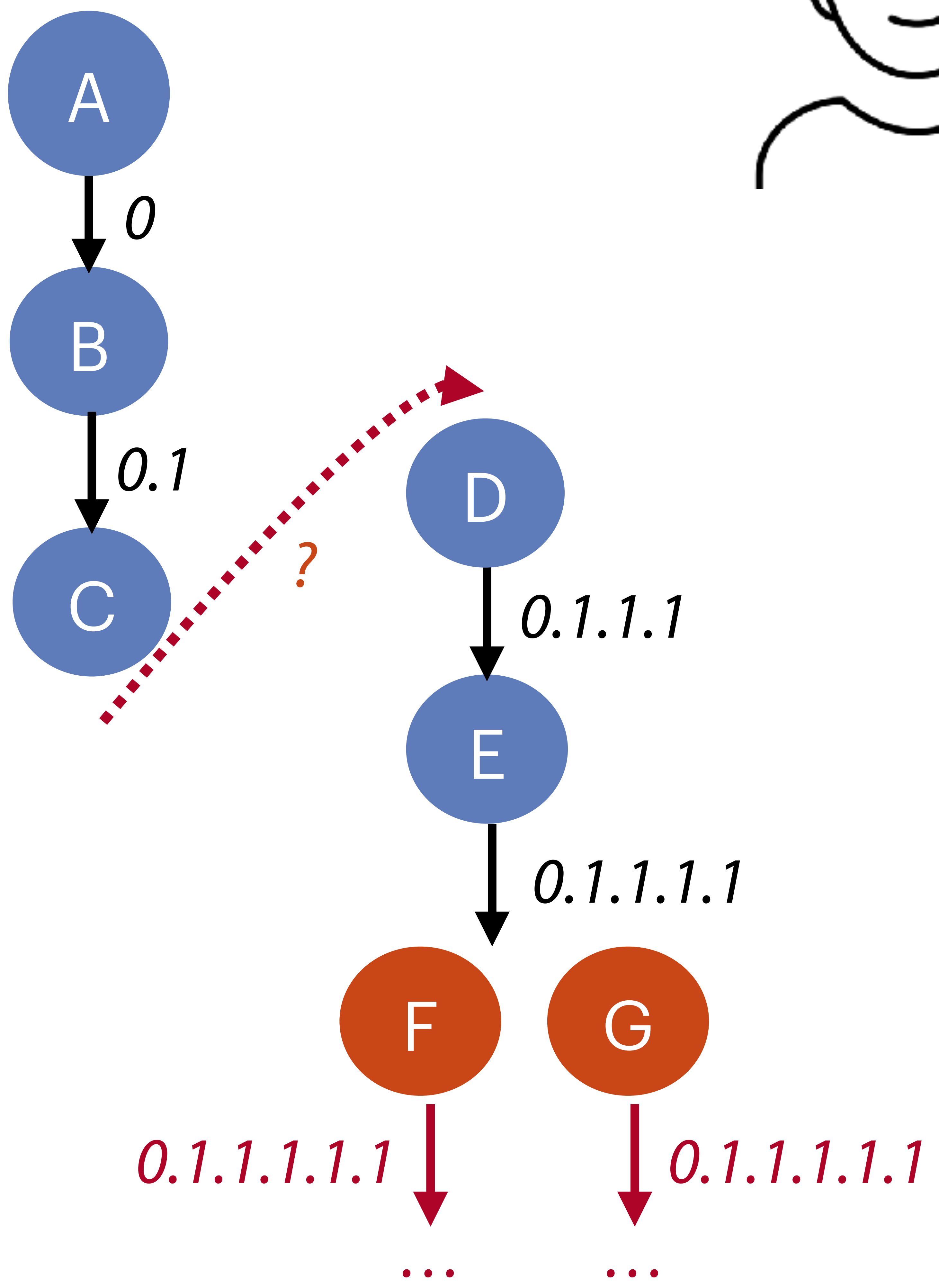
Inconsistencies in call graph data

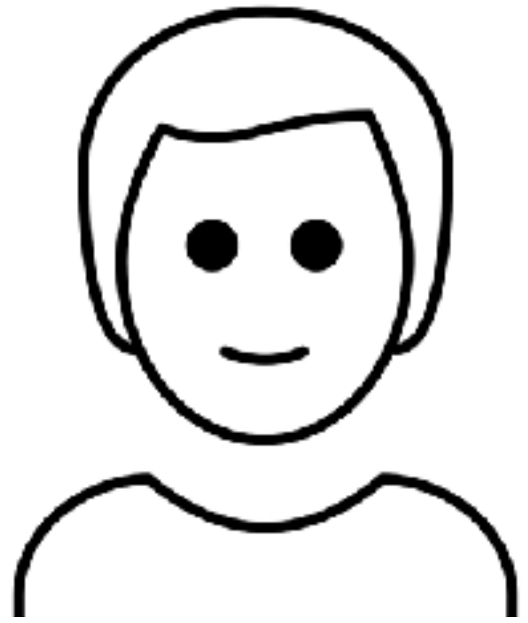
- 99.5% are missing a duplicate row for two-way communication
- 35.2% are missing all rows for a call

Disconnected graphs likely caused by ***data loss***

- 30.2% of traces have non-unique rpcids
- Non-unique rpcids are propagated to all calls downstream (rendering them non-unique)

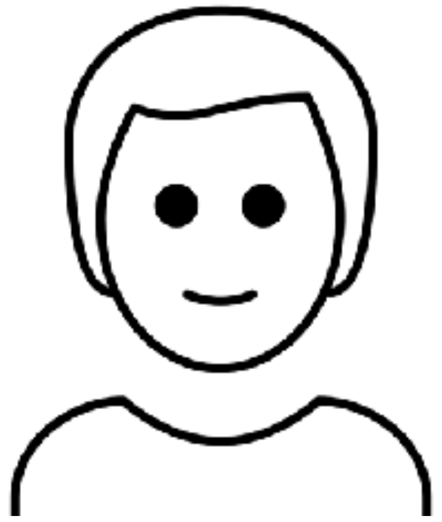
Ambiguous edges likely caused by ***context propagation errors*** (where the rpcid is not updated)





Outline

- Introduction
- Using Alibaba's datasets
- Casper: Remedying errors using redundancies in trace model
- Evaluation
- Implications of errors in trace data



Casper: high-level approach

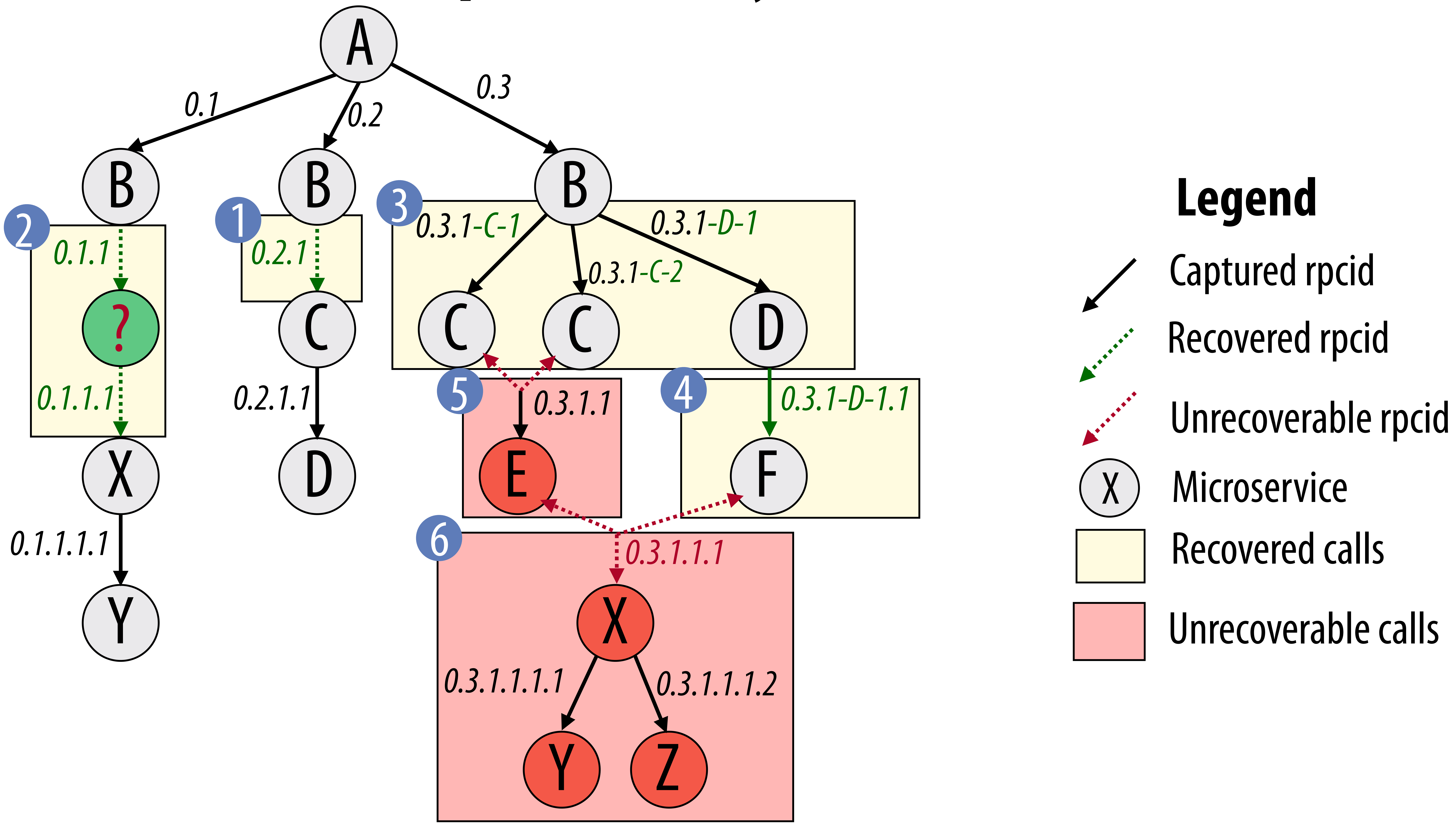
Goal: build the largest accurate trace topologies

- Omit ambiguous edges that remain after error correction

Approach: Breadth-first reconstruction, level-by level from the table

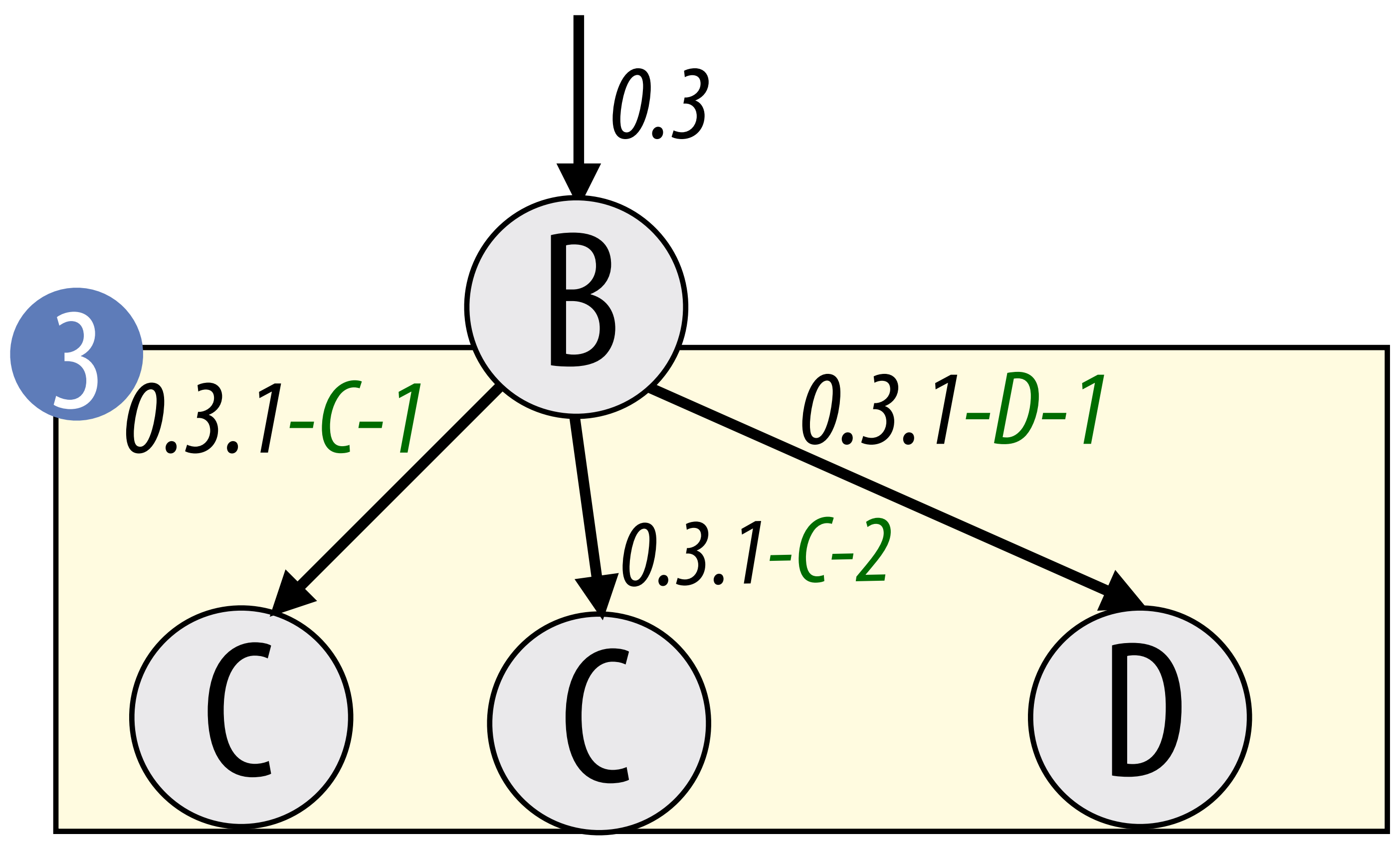
- At each level look for *data loss* or *CPE*s & correct inconsistencies
  - Data loss: easy, add missing upstream rpcids until connects to existing call graph
  - CPE: complicated, use redundancies in the table to differentiate calls

Casper recovery modes



Fixed trace graph (showing recovered and omitted nodes)

Recovery at source of CPE: # of unique calls



Calculating the number of calls to a DM

- True number of calls is unknown (due to data loss), but can find the minimum
- rt values are rounded down, anything below a threshold is rounded to 0
- -rt rows cannot be paired with 0 rt rows

rpcid	UM	DM	rpctype	rt
0.3.1	B	C	http	2
0.3.1	B	C	http	0
0.3.1	B	C	http	0
0.3.1	B	C	http	0
0.3.1	B	D	mq	0

3
2
1

1) Number of fast calls (0 rt): $\lfloor \frac{(rt==0)}{2} \rfloor$

2) Extra fast row: $(rt==0)\%2$

3) Number of slow calls:

$$\max(+\mathrm{rt}, abs(-\mathrm{rt}+extra\_fast\_row))$$

Total number of calls: fast + slow calls

Outline

- Introduction
- Using Alibaba's datasets
- Casper: Remedying errors using redundancies in trace model
- Evaluation
- Implications of errors in trace data

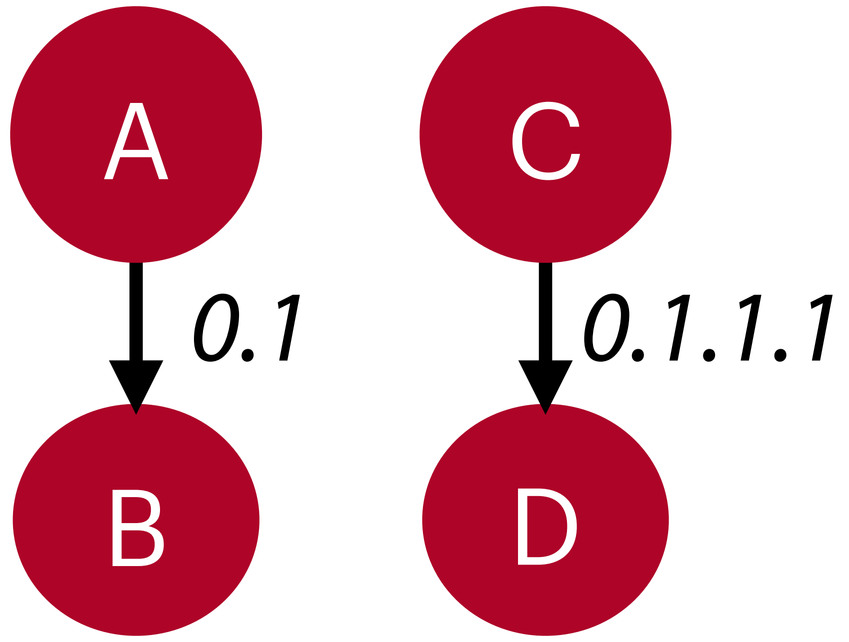
Additional complete traces are different from original complete traces

Casper increases the number of *complete* traces from 58% to 84%

Additional 26% of traces are larger, wider, and deeper than original set of complete traces

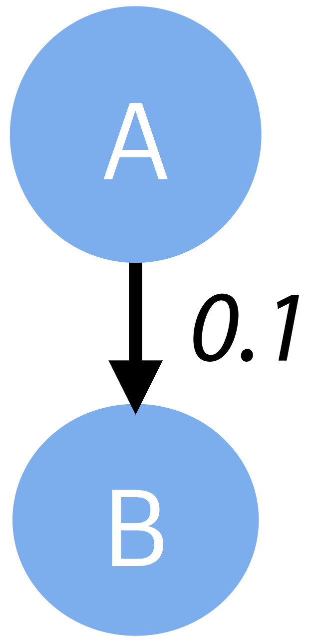
Methods of building traces

rpcid	UM	DM	rpctype	rt
0.1	A	B	http	+/-
0.1.1.1	C	D	http	+/-
0.1.1.1	C	E	db	+

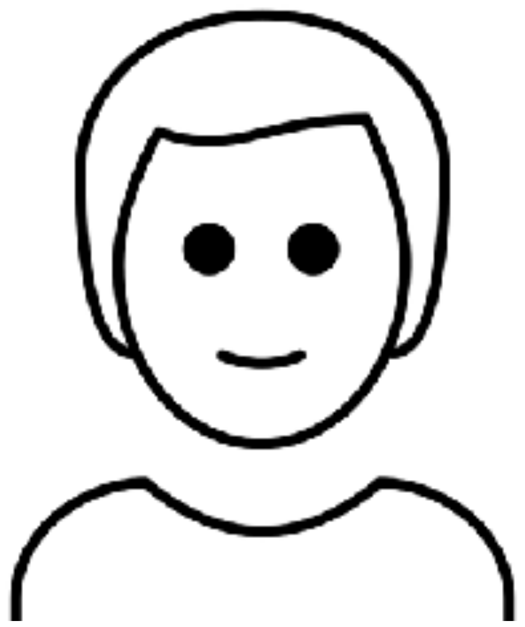


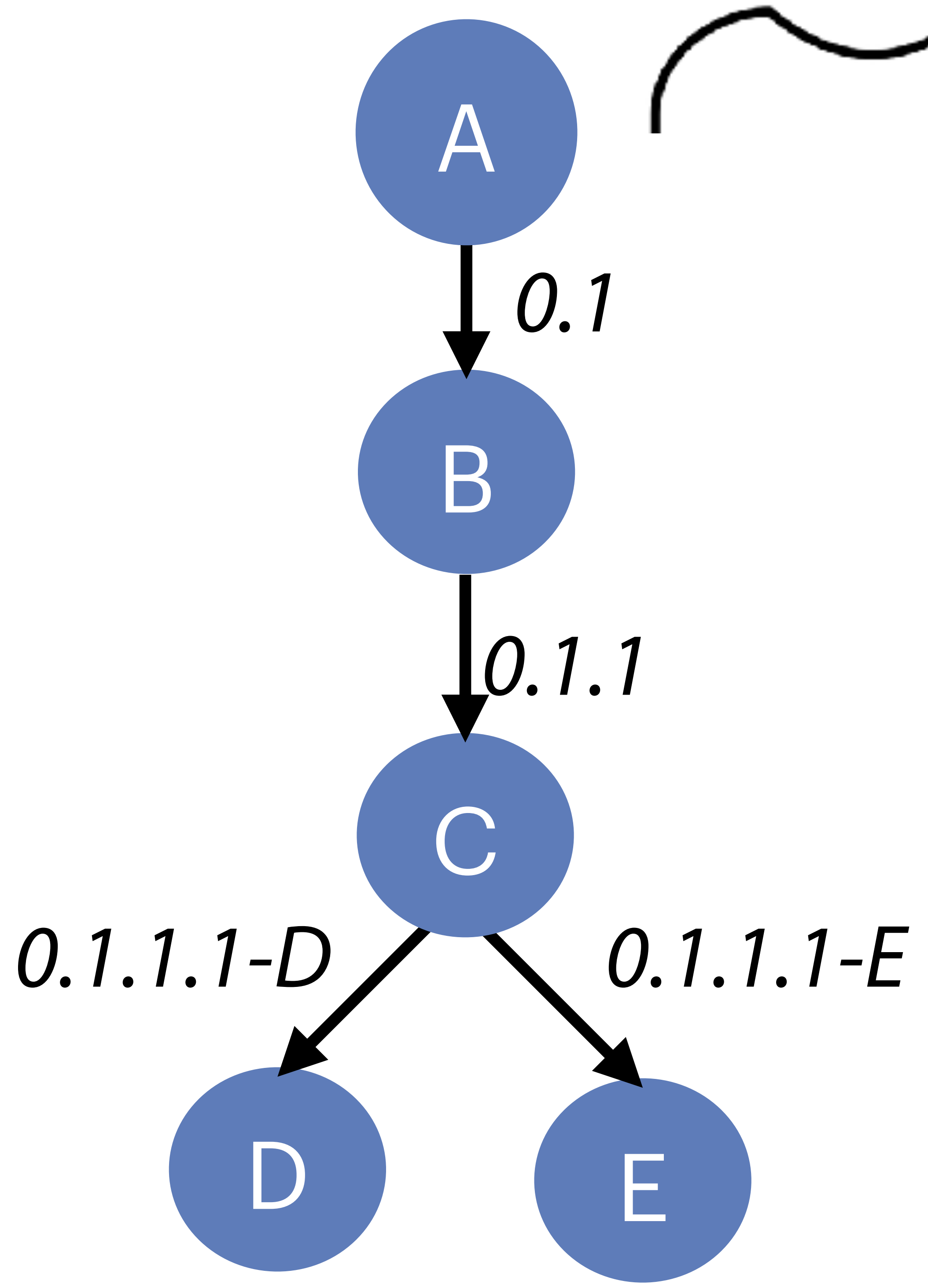
Naive-rpcid[Alibaba]:
using assumptions provided in paper

Naive-accurate:
only preserve traces that meet all assumptions



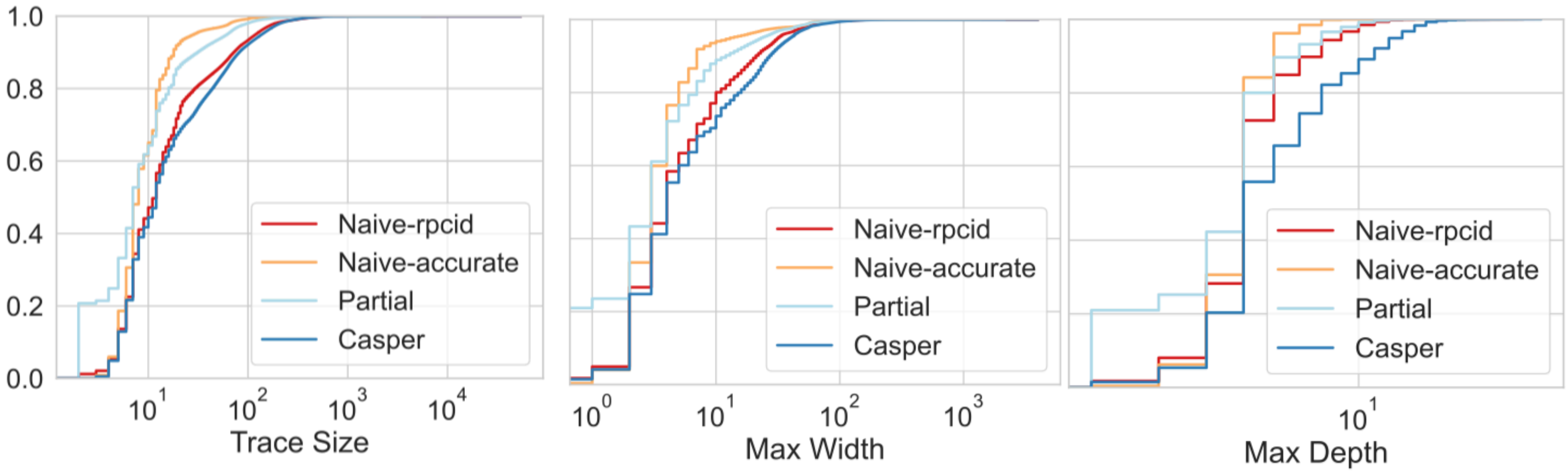
Partial[LatenSeer]:
keep portions of trace that meet assumptions, remove anything downstream from an inconsistency





Casper:
recover from errors in data

Casper traces are larger, deeper, and wider than other methods



Outline

- Introduction
- Using Alibaba's datasets
- Casper: Remedying errors using redundancies in trace model
- Evaluation
- Implications of errors in trace data

Implications & future work

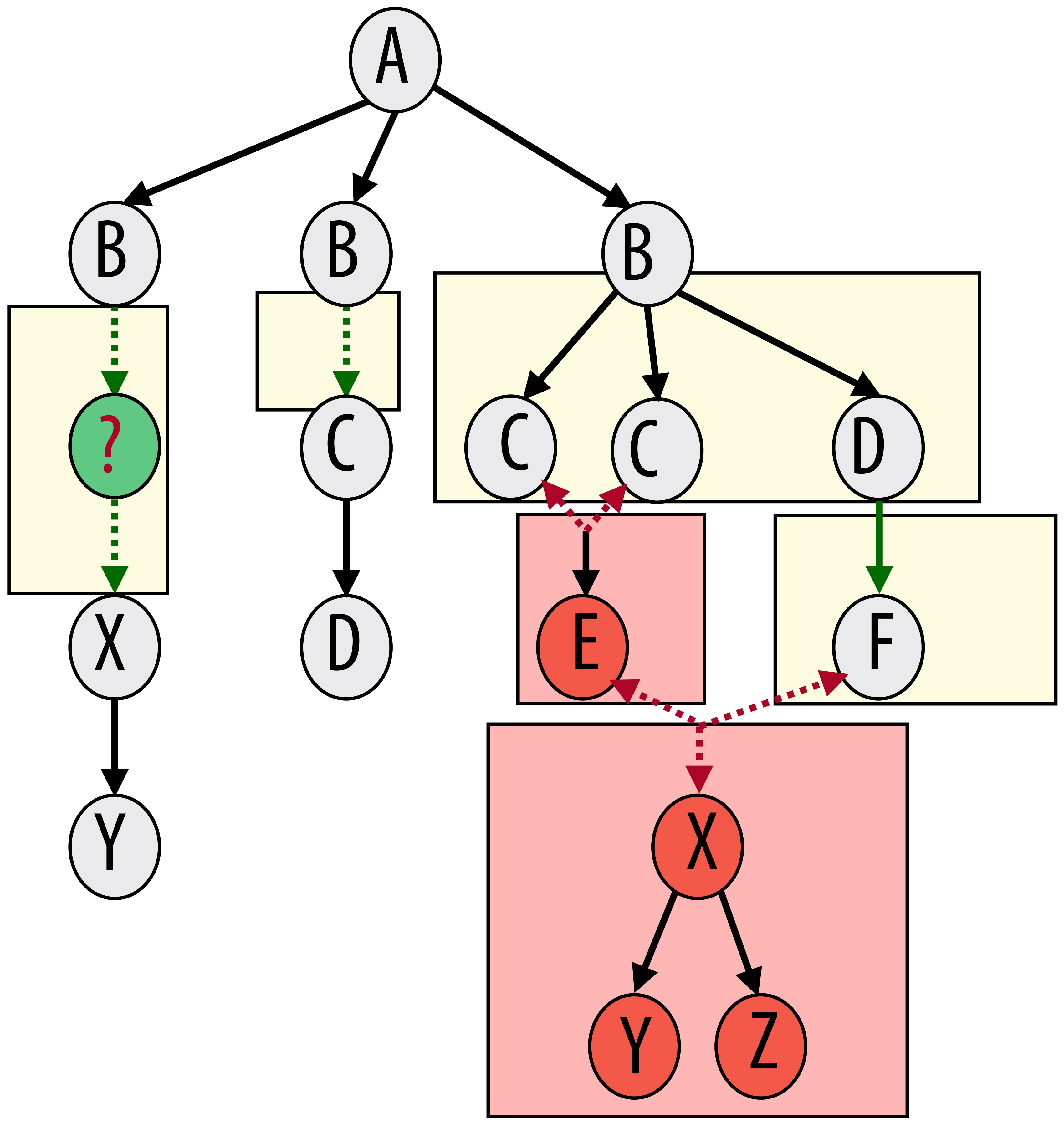
Users of Alibaba's call graph data should be aware of the data quality issues and the impact they can have on research

Redundancies in trace models can be powerful for recovering from errors

Proper context propagation is essential for observability in distributed systems

Casper Summary

- Classify inconsistencies in Alibaba's call graph dataset
- Present Casper: a tool that uses redundancies in Alibaba's trace model to fix errors
- Showed Casper traces are larger and wider than other rebuild methods
- Released corrected traces and Casper's code



HARVARD
Dataverse

https://github.com/docc-lab/casper

https://doi.org/10.7910/DVN/SS9SIY

Citations

- **[Google SOSP '23]:** Korakit Seemakhupt, Brent E. Stephens, Samira Khan, Sihang Liu, Hassan Wassel, Soheil Hassas Yeganeh, Alex C. Snoeren, Arvind Krishnamurthy, David E. Culler, and Henry M. Levy. 2023. A Cloud-Scale Characterization of Remote Procedure Calls. In Proceedings of the 29th Symposium on Operating Systems Principles (SOSP '23). Association for Computing Machinery, New York, NY, USA, 498–514. https://doi.org/10.1145/3600006.3613156
- **[Erms ASPLOS'23]**: Shutian Luo, Huanle Xu, Kejiang Ye, Guoyao Xu, Liping Zhang, Jian He, Guodong Yang, and Chengzhong Xu. 2022. Erms: Efficient Resource Management for Shared Microservices with SLA Guarantees. In Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS 2023). Association for Computing Machinery, New York, NY, USA, 62–77. https://doi.org/10.1145/3567955.3567964
- **[Nodens ATC'23]**: Jiuchen Shi, Hang Zhang, Zhixin Tong, Quan Chen, Kaihua Fu, and Minyi Guo. 2023. Nodens: Enabling Resource Efficient and Fast QoS Recovery of Dynamic Microservice Applications in Datacenters. 2023 USENIX Annual Technical Conference (USENIX ATC 23). https://www.usenix.org/conference/atc23/presentation/shi
- **[LatenSeer]**: Yazhuo Zhang, Rebecca Isaacs, Yao Yue, Juncheng Yang, Lei Zhang, and Ymir Vigfusson. 2023. LatenSeer: Causal Modeling of End-to-End Latency Distributions by Harnessing Distributed Tracing. In Proceedings of the 2023 ACM Symposium on Cloud Computing (SoCC '23). Association for Computing Machinery, New York, NY, USA, 502–519. https://doi.org/10.1145/3620678.3624787
- **[Mbench]**: A. Detti, L. Funari and L. Petrucci, "µBench: An Open-Source Factory of Benchmark Microservice Applications," in IEEE Transactions on Parallel and Distributed Systems, vol. 34, no. 3, pp. 968-980, 1 March 2023, doi: 10.1109/TPDS.2023.323644