Identification of Device Dependencies Using Link Prediction

Lukáš Sadlek, **Martin Husák**, Pavel Čeleda
husakm@ics.muni.cz

Masaryk University

May 7, 2024 @ IEEE/IFIP Network Operations and Management Symposium, South Korea

Introduction

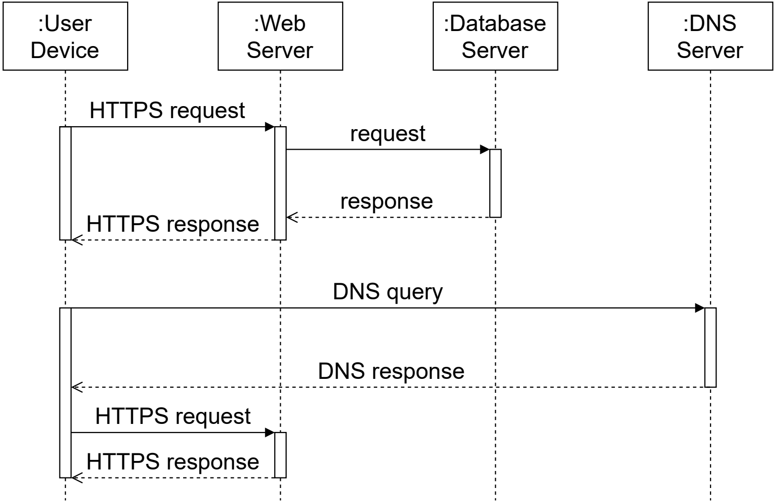
Motivation

- Some devices provide **essential network services**, e.g., domain controller, other devices **depend** on them
- How to determine such dependencies in **large and dynamic** networks?
- Provide **uniform method** instead of parsing **various** data sources

Definitions

- **Local-remote (LR) dependency** – a server needs a remote server to answer requests from user devices, e.g., a web server depends on a database server
- **Remote-remote (RR) dependency** – server is indirectly dependent on another one, e.g., a user first has to query a DNS server

Local-Remote and Remote-Remote Dependencies



Research Questions

RQ1 *Can we identify device dependencies using **graph-based** machine learning for the **link prediction** problem?*

RQ2 *What **correctness, time aspects, dependency types,** and **amount of processed data** of the link prediction approach for device dependency identification can we obtain?*

Fundamentals: Latent Graph Representation Learning

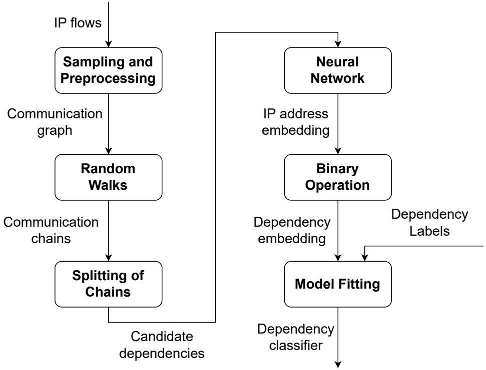
Description

- Reveals the **hidden structure** of the graph
- **Node embedding** – low-dimensional space where **nearby vertices** are **close** in the original graph

Common Approach

- Inspired by the **Natural Language Processing (NLP)** approaches
- **Random walks** represent sentences, **vertices** words, and **neighboring vertices** context
- Node embedding maps vertices to **vectors of values** – their embeddings

Steps of the Method



- **Sampling and preprocessing** – reservoir sampling of IP flows that represent edges
- **Random walks** – conditions imposed on two subsequent edges in random walks
- **Splitting of chains** – create candidate dependencies
- **Neural network** – one hidden layer with number of features equal to embedding dimension
- **Binary operation** – combines node embedding into dependency embedding, e.g., scalar product
- **Model fitting** – classifier is learned for known positive and negative dependencies

Method – Sampling and Preprocessing

Input

- **IP flows** – IPFIX in our case, unidirectional
- Sampling required due to the complexity of neural networks

Sampling

- Obtaining a **representative sample** with n internal and m external IPv4 addresses (removes network scanning, for example)
- Approx. 100 of the most important IP addresses in the experiment
- **Reservoir sampling** selects each edge with equal probability

Preprocessing

- Removing communication that does not use TCP or UDP protocols
- **Communication graph** is constructed from the filtered and sampled flows

Method – Conditions for Random Walks

Four Conditions

1. **Opening** of **LR** dependency

   $$t_1(v_i, v_{i+1}) \leq t_1(v_{i+1}, v_{i+2}) \leq t_2(v_{i+1}, v_{i+2}) \leq t_2(v_i, v_{i+1}), i \in \{1, \ldots, n-2\}$$

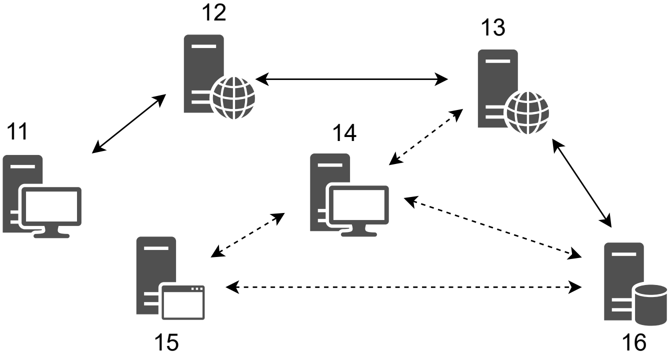
   where t_1 and t_2 denote the timestamps of beginning and end of the flow
2. **Return** from **LR** dependency (flow sequence in the opposite direction than Opening)
3. **Opening** of **RR** dependency
4. **Return** to the **previous** IP address

Notes

- **Mathematical expressions** for conditions 2 – 4 are listed **in the paper**
- Edges **fulfilling** conditions represent building blocks of **communication chains**

Method – Communication Chains and Candidate Dependencies

- **Communication chain:** $(11, 12, 13, 16)$
- **Context size:** 3
- **Contexts:** $(11, 12, 13), (12, 13, 16)$
- **Candidate dependencies:** $(11, 12), (11, 13),$ $(12, 13), (12, 16)$



- The longer the distance in the chain, the lower the chance that dependency exists
- **Candidate dependencies** are pairs of addresses within the context

Method – Final Steps

Neural Network (NN)

- **NN** uses the candidate dependencies to estimate features of the hidden layer
- IP addresses from candidate dependencies would be close together in the IP address embedding space

Dependency Embedding

- Scalar product of two vectors (node embeddings)
- This extends the space into which the candidate dependencies are mapped
- Contains a vector of values for each candidate pair
- Each possible dependency in the embedding is given its label by a trained **dependency classifier**

Evaluation

Python Implementation

- Reuses **neural network functionality** of Node2Vec from PyTorch Geometric
- We provide sampling of **random walks** and processing of **IP flows**

Datasets

- **T1 – T6** – datasets from cyber defense exercise with **six teams** and **topologies**
- **U10m, U1h** – **ten-minute** long and **one-hour** long datasets from **university campus network**

Ground Truth

- Determined by **exhaustive brute force** search of all possibilities

Ground Truth – Statistics

	T1	T2	T3	T4	T5	T6	U10m	U1h
DD	300	444	330	427	321	143	38,372	2,866
RR	248	131	177	310	98	35	1,731	164
RR3	1,875	113	697	2,067	161	26	23,905	2,347
TD	17	13	22	24	9	14	854	117
TD3	0	0	2	1	0	0	359	81

Table 1: **Number of dependencies** for datasets. U1h contains average values from **twelve time windows**. DD denotes **direct**, RR **remote-remote**, and TD **transitive** dependencies.

Data Size and Execution Time

	T2	T6	U10m	U1h
IP flows	54,941	28,506	8,259,584	78,270,416
IP addresses	1,421	247	451,365	1,235,300
Vertices	96	103	129	93
Edges	21,720	16,080	15,076	18,411
Preprocessing	14.9 s	6.2 s	27.2 s	27.3 s
Creating embedding	14.0 min	14.0 min	6.2 min	6.9 min
Computation	<1 s	<3 s	<1 s	<1 s

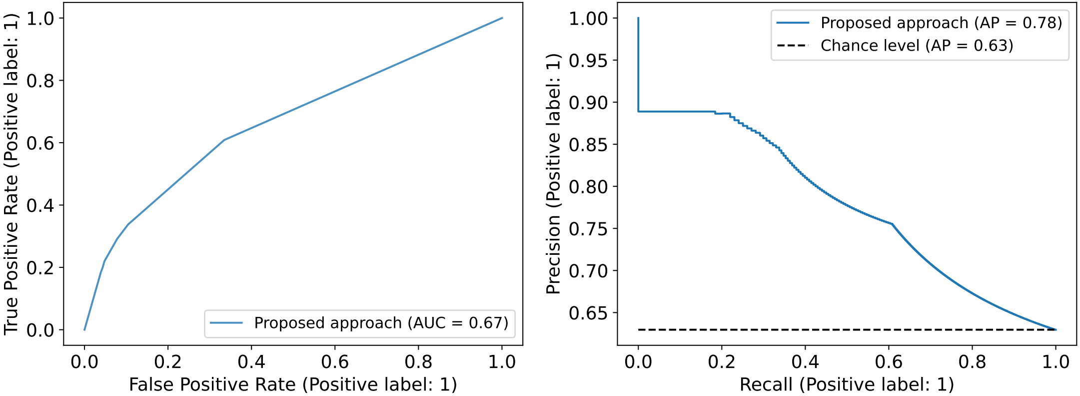
Table 2: **Amount of data** and measured **time** for selected datasets. U1h contains averages from **twelve time windows** except for IP flows and addresses.

Evaluation Metrics

Test size		T1	T2	T3	T4	T5	T6	U10m	U1h
0.25	Accuracy	0.514	0.417	0.493	0.503	0.429	0.337	0.479	0.515
	Precision	0.612	0.521	0.597	0.605	0.530	0.444	0.604	0.615
	F1 score	0.666	0.566	0.644	0.656	0.572	0.462	0.625	0.664
0.50	Accuracy	0.535	0.453	0.529	0.532	0.485	0.403	0.515	0.545
	Precision	0.628	0.544	0.621	0.630	0.580	0.485	0.612	0.638
	F1 score	0.677	0.584	0.669	0.672	0.617	0.510	0.645	0.681
–	AUC	0.68	0.64	0.67	0.68	0.67	0.61	0.71	0.69
	AP	**0.81**	**0.74**	**0.80**	**0.81**	**0.78**	**0.68**	**0.84**	**0.81**

Table 3: **Evaluation metrics** for datasets averaged from **fifteen train-test splits** except for U1h split also into twelve consequent windows.

Receiver Operating Characteristic and Precision-Recall Curves (T5)



- **Average Precision (AP)** is claimed to be a suitable metric for **imbalanced datasets**

Lessons Learned

Method

- Approach for **all dependency types** simultaneously
- Some types of dependencies do **not** provide **enough labels**

Evaluation

- Ground truth **corresponds** to **LR dependencies** obtained from NSDMiner
- AUC for **directed** graphs comparable with **related work** and **undirected** graphs
- **Local similarity indices** – approach reveals **not directly visible** dependencies
- **Large** amounts of data must be **split into batches**

Summary

Contribution

- Graph representation learning for a **new use case** – dependency embedding
- Core and **most complex** part – **custom exploration** of communication chains
- **Conditions** for timestamps of IP flows and adjustments for **IP flow data**
- **Measurements** of method's properties

Supplementary Materials

- A proof-of-concept **implementation**, **ground truth** labels, and **results**
- Available at: https://doi.org/10.5281/zenodo.10548433

MASARYK UNIVERSITY