Topology Discovery in controlled environments

Maximilian Pudelko
Betreuer: Florian Wohlfart, Sebastian Gallenmüller
Seminar Future Internet WS2015
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: maximilian.pudelko@tum.de

ABSTRACT
This paper describes a method to collect data about the topology of networks. Further on the data is used to generate images representing the links between the hosts giving a broader overview over the network. The single steps of the process are fully automated to reduce manual interaction with the system.

Keywords
topology discovery, test environment, lldp

1. INTRODUCTION AND SCENARIO
A research networking testbed can be a rapidly changing environment as some experiments require different setups. This leads to a constantly changing wiring and positioning of the single testbeds. These changes done must be documented manually and merged into the existing documentation. As this is seen as a cumbersome process it is often skipped leading to an outdated information base. It is the goal of this paper to develop a program which aids this documentation process by utilizing existing technologies and software to detect the physical topology of the testbed. With this goal set the program has to meet the following requirements:

- The detected topology should be a correct representation of the hosts and their interconnection at the link layer of the network. Detailed information about speeds and type, in case of links, and hardware configuration in case of hosts should be collected.
- Usage of the tool should result in reduced human involvement compared to collecting the data manually.
- The output must be graphical in a way that presents the topology suitably.
- The program should integrate into the existing testbed setup. Specifically it must not interfere with other running experiments.

The reminder of this paper is organized as follows: Section 2 describes existing technologies and how they apply here, Section 3 illustrates the developed solution of which the results on the Baltikum testbed are presented in Section 4.

2. RELATED WORK
As the idea of developing a method to discover present devices and their physical topology on a Local area network, IEEE 802 (LAN) is not new, several solutions have been already developed by different parties and organisations. The following chapter will give a short overview over some of these protocols, compares them and explains why Link Layer Discovery Protocol (LLDP) was chosen as the base to solve the problem at hand. Additionally the basic operation of LLDP will be clarified as needed for the understanding of the solution as it builds on it.

2.1 Proprietary Protocols
Introduced in 1994 the Cisco Discovery Protocol (CDP) aims to provide a mechanism to discover devices connected to a network at the Link Layer (L2) by broadcasting information about itself. This enables any device wanting to collect information to simply listen for this broadcasts without any prior configuration [3]. As the usefulness of such a tool for network administration became evident, it was included in the OS of Cisco's networking hardware and over the time developed further to e.g. set-up VoIP telephones.
Similar the Link Layer Topology Discovery Protocol (LLTD) was developed by Microsoft which too operates on any IEEE 802 network [4]. Contrary to the CDP it was not included in networking hardware, but in the network stack of Microsoft's OS starting with Windows Vista to display a graphical Network Map of home networks and to detect connectivity problems of wireless networks.
While for both protocols free Linux implementations exist, neither of them are a viable solution as they are either limited to vendor specific hardware, which would result in incomplete topologies in case of mixed setups, or require the signing of a license agreement to use them.

2.2 Link Layer Discovery Protocol
In 2005 a IEEE task force created a vendor neutral protocol called LLDP to unify the up to then incompatible vendor specific protocols. In the following the basic operation will be explained.

Like its predecessors LLDP operates on the Link-Layer of LANs, which gives it the benefit of low configuration prerequisites. In particular no IP addresses are needed as communication happens via Ethernet with vendor-set or manually administered MAC addresses. A participant, referred to as chassis, may be active and/or passive depending on configuration. An active chassis will broadcast LLDP frames on all of its ports in regular intervals to inform potential listeners of its presence. If a passive chassis retrieves this frame it will store this information in a local Management

doi: 10.2313/NET-2016-07-1_08

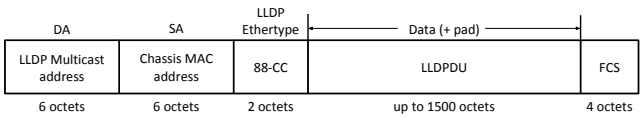

Figure 1: IEEE 802.3 LLDP frame format[2]

Information Base (MIB). While two hosts may be able to reach each other, no direct communication happens between them. Particularly it's not possible to make requests to other chassis asking for the content of their databases, as LLDP is designed as a one way protocol [2] using only broadcasted and not directly addresses frames frames. As LLDP operates on L2 every frame has to be wrapped in a Ethernet frame as shown in figure 1. Since these frames are not directed to one specific host a broadcast address has to be used in the destination address field. As normal broadcasted frames with target FF-FF-FF-FF-FF would get forwarded by bridges and switches, which would lead to an inclusion of equipment beyond the physical link of a chassis, the standard suggests a set of three reserved addresses [1] to limit frames to one link on conforming switches. The source address is filled out with the MAC of the interface (port) on which the frame goes out. To distinguish LLDP frames from other protocols like IP a separate Ethertype of 88-CC is used. Next follows the actual payload in form of the LLDP Data Unit (LLDPDU). Each LLDPDU consists of a concatenation of TLVs of which some are mandatory and some optional. The Chassis ID TLV contains the identifier of the chassis that send out the frame and has to remain constant for the time of operation of the LLDP service. This property is important as it enables us to compile a list of the available hardware in the network as explained in 3.2. The likewise mandatory Port ID TLV uniquely identifies the sending port of a chassis. With this value it becomes possible to distinguish between multiple links between two chassis or to determine the exact port number on a connected switch. Among the optional ones the organizationally specific [2] System Capabilities TLV and the MAC/PHY Configuration/Status TLV are of special interest, as they indicate the type of networking device (router, switch or simple station) and the bandwidth/type of the physical link respectively.

As this standard has been implemented by all major networking hardware manufacturer and is usable without restrictions, it is chosen as the protocol to use in the Baltikum testbed. To enable this functionality on the Linux hosts the FOSS program lldpd is employed, which additionally also implements a client for most vendor specific protocols [1] and thus ensures maximum coverage.

2.3 Other, higher layer protocols

While several other protocols like Open Shortest Path First, a IP link-state routing protocol (OSPF) and Neighbor Discover Protocol, part of IPv6 (NDP), which operate in a similar domain, exist, they are not applicable to the scenario of a testbed because of the layer they operate on. The requirement of getting a link layer topology can not be satisfied with protocols operating at network layer, as they, by design, abstract single links away. Also is the required knowledge and organization to set them up often too much of an overhead or not even possible at all.

[1] 01-80-C2-00-00-0E, 01-80-C2-00-00-03 and 01-80-C2-00-00-00. Each with different meanings to provide further control

[2] Vendors and organizations can define custom TLVs and apply for inclusion into the standard

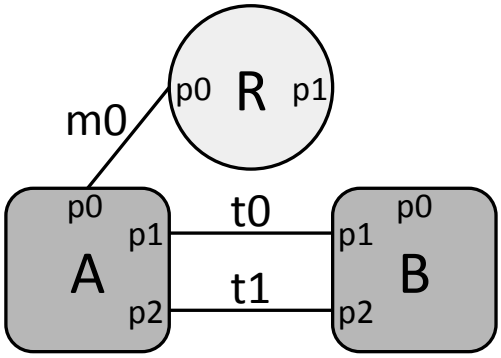

Figure 2: Example topology consisting of the management router R and testbeds A, B

3. PROPOSED ARCHITECTURE

Despite being favorable over the other protocols, LLDP comes with several shortcomings by not providing any means to solicit information from other chassis and operating on the link-layer (L2) only. On this layer communication is mostly limited to direct, neighbor-to-neighbor messages without any routing over other machines involved. The example in Figure 2 demonstrates this limitation. With R being the management router and the only host which MIB we consult, we would get correct information about the testbeds A and B and their management connections $m0_{R,A}$, $m1_{R,B}$ respectively, but the, usually more interesting, test links $t0$ and $t1$ would stay hidden. This leads to the conclusion that for a complete view of the network the content of all MIBs of the relevant testbeds, e.g. A and B too, has to be collected. This in turn complicates the setup process a bit as shown in figure 3, as it has to be ensured that every host has completed the setup step and is ready to receive/send messages or parts of the network will remain uncharted. Additionally there will be a high ratio of duplicate entries which will be dealt with in chapter 3.2.

3.1 Setup and Data gathering

The first step in the discovery process is to enable LLDP or vendor specific equivalents on the deployed hardware like switches or router and the testbeds. Depending on firmware or configuration these services may normally be disabled since they may influence experiments negatively or are generally not needed. It should be noted that they are only required during the discovery process and can safely be disabled again afterwards. On the Baltikum testbed this can be launched over the existing Command and Control (C&C) interface on the management host. Since the setup process may take a different amount of time on the hardware a synchronization barrier is employed as show in figure 3. In case of the Baltikum testbed of the TUM this is done over the management network, to which all test devices are connected, but different local viable solutions are thinkable. This barrier may only be passed once every participating host is

doi: 10.2313/NET-2016-07-1_08

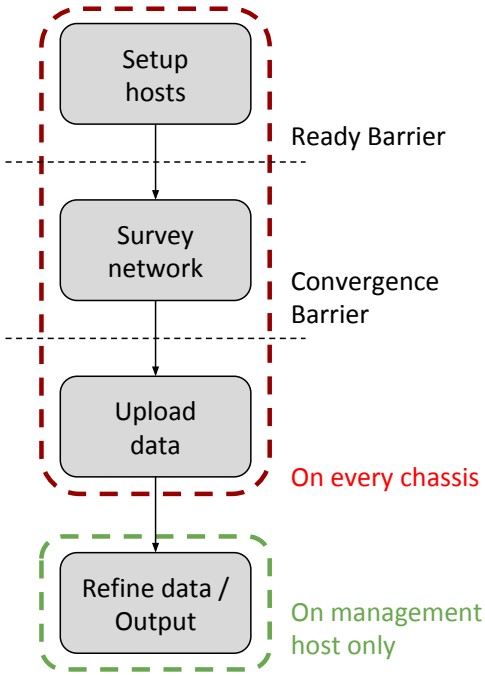

Figure 3: Proposed flow

ready, which marks the beginning of the data gathering phase.
Now every host sends the LLDP frames over its interfaces and collects information over its adjacent neighbor testbeds or hardware. The physical network should be stable at this point, so no more replugging should happen. Experiments on the Baltikum testbed have shown that this discovery process only takes a few milliseconds, but non-ideal conditions in networks spanning large distances or including wireless links may lead to dropped frames and delays. So a convergence timer of 10 sec to minimize this risk is employed with the second barrier. After this every host shuts down the discovery service and saves the gathered data locally to be collected in the next step. Additionally every chassis collects information about its hardware such as number and type of CPUs, amount of RAM and built in mainboard to enrich the dataset. As this kind of information is not part of the LLDP specification, it is done at last and simply added to the dataset.

3.2 Data aggregation and refinement

Now the collected data has to be aggregated and refined to make it usable. On the Baltikum testbed this has been realized over the already existing infrastructure for the upload of normal experiment results, but this may be solved as needed depending on the actual setup. Once the collection is completed the filtering phase begins on the management host. The need for this step can be explained with the example topology in figure 2 as we look at the theoretically collected data:

$$links = \{(A_{p1}, B_{p1}), (B_{p1}, A_{p1}), (A_{p2}, B_{p2}), (B_{p2}, A_{p2})\}^3$$

[3]The management links m0, m1 have been excluded for clarity. lldpd includes filter to ignore certain interfaces.

Where a tuple (X_i, Y_j) stands for a link from X on port i to Y on port j. And

$$chassis = \{A_A, A_B, A_B, B_A, B_A, B_B\}^4$$

where a value X_Y represents a chassis X as seen by Y.

The seemingly duplicate values come from the drawbacks of using a link layer protocol. As e.g. A is connected to B over multiple links, B will receive multiple frames from A only distinguished by the port IDs. The program will therefore detect multiple entries about the same chassis, as identified by identical chassis IDs, and merge them. The same principle applies to the set of links as each side reports the existence of a link from its Point of View (PoV), the program will try to find links with common endpoints and join them together to bidirectional ones. In the example above this would lead to the aggregation of e.g. the links (A_{p1}, B_{p1}) and (B_{p1}, A_{p1}) as they have matching chassis and ports to one link $\{A_{p1}, B_{p1}\}$.

3.3 Data presentation

To keep the output as flexible as possible the JavaScript Object Notation (JSON) was chosen to store the results for further use such as generating topology graphs later. The dataset is organized in two list. The first one contains all connections with associated metrics like link speed as detected by the LLDP:

```
[{"endpointA": {
    "interface_name": "p1",
    "device_name": "A",
    "device_id": {
      "type": "mac",
      "value": "ab:cd:ef:00:00:01"
    }
  },
  "endpointB": {
    "interface_name": "p1",
    "device_name": "B",
    "device_id": {
      "type": "mac",
      "value": "ab:cd:ef:00:00:02"
    }
  },
  "speed": "10GigBaseX"
}]
```

While the seconds list contains all discovered chassis identified by an ID and further described by a description string and fields containing their hardware info obtained by the local data collection:

```
[{"A": {
    "id": {
      "type": "mac",
      "value": "ab:cd:ef:00:00:01"
      },
      "descr": "Debian GNU/Linux 3.16.0-1-grml-amd64",
      "cpu": "E3-1230 V2 @ 3.30GHz",
      "cpu_count": 8,
      "ram": 17179869184
    }
}]
```

The plotting process then becomes a matter of drawing all chassis and connecting them.

[4]A chassis always "discovers" itself, as it can supply the most information about it.

doi: 10.2313/NET-2016-07-1_08

4. RESULTS ON THE BALTIKUM TESTBED

In this chapter the results of a prototype implementation of the strategy proposed in 3 will be presented. The program is divided in a bash script for the set-up phase which relies heavily on the existing infrastructure to start and control experiments, as the whole topology discovery process is defined as a regular experiment. For the refinement process and the image generation a Python script incorporating the graph drawing library pyGraphViz was developed. The Baltikum testbed consist of 10 Linux hosts, one switch, one router and the management network. Figure 4 shows a generated topology of the partial testbed as at the time of the discovery not every host was available. While the correctness of all the interconnections between the hosts could be confirmed via manual inspection of the interfaces, it became evident that LLDP did not detect links which start and end on the same chassis. In particular the testbeds Cesis and Nida are missing each two of these "short circuits" as seen in Figure 4. Inspection of the source code of lldpd revealed that frames that come from the same host are discarded silently to deal with faulty NIC drivers which relay broadcasted frames back. About the overall performance can be said that it's largely dependent on the boot time of the machines which can be as long as 5 minutes, while the experiment itself only takes a few seconds.

5. CONCLUSION & FUTURE WORK

While a the process of collecting the required information about a topology could be nearly automated, the goal of a complete automation could not be archived, as the process is not yet completely error free and the generated images often need manual adjustments to be prevent overlapping labels. In total this program still can provide a value help in the documentation process as a base to extent from. While the program it its prototype state is usable, certain areas need further research. In its current state the topology discovery still has to be explicitly run by the researcher and then inserted into the documentation, which can be forgotten. A possible way to solve this would be to let the LLDP service run continuously instead of running it on-demand. While it has to be determined that this does not interfere with the experiments it would provide some benefits, as now tracking changes over longer time intervals and the utilization of a central database would be possible. This would shift the responsibility to the network administrator and away from the single researcher, who could then, even retroactively, query this database to get information about the systems state at the time of his experiment. This interface again could be provided via the existing wiki to keep information central and generally available.

6. REFERENCES

[1] lldpd Development Homepage https://vincentbernat.github.io/lldpd/features.html Accessed: 2015-12-03

[2] IEEE standard for local and metropolitan area networks: Station and media access control connectivity discovery, ieee std 802.1ab, 2009.

[3] Cisco Systems, Inc. LLDP-MED and Cisco Discovery Protocol, Jun 2006.

[4] Microsoft Corporation. Link Layer Topology Discovery (LLTD) Protocol Specification, Aug 2010.

Glossary

C&C Command and Control

CDP Cisco Discovery Protocol

chassis A physical component incorporating one or more IEEE 802 LAN stations and their associated application functionality.

JSON JavaScript Object Notation

L2 Link Layer

LAN Local area network, IEEE 802

LLDP Link Layer Discovery Protocol

LLDPDU LLDP Data Unit

LLTD Link Layer Topology Discovery Protocol

MIB Management Information Base

NDP Neighbor Discover Protocol, part of IPv6

OSPF Open Shortest Path First, a IP link-state routing protocol

physical topology Physical topology represents the topology model for layer 1 of the OSI stack - the physical layer. Physical topology consists of identifying the devices on the network and how they are physically interconnected. Note that physical topology is independent of logical topology, which associates ports based on higher layer attributes, such as network layer address.

PoV Point of View

TLV type, length, value. A short, variable length encoding of an information element consisting of sequential type, length, and value fields where the type field identifies the type of information, the length field indicates the length of the information field in octets, and the value field contains the information, itself.

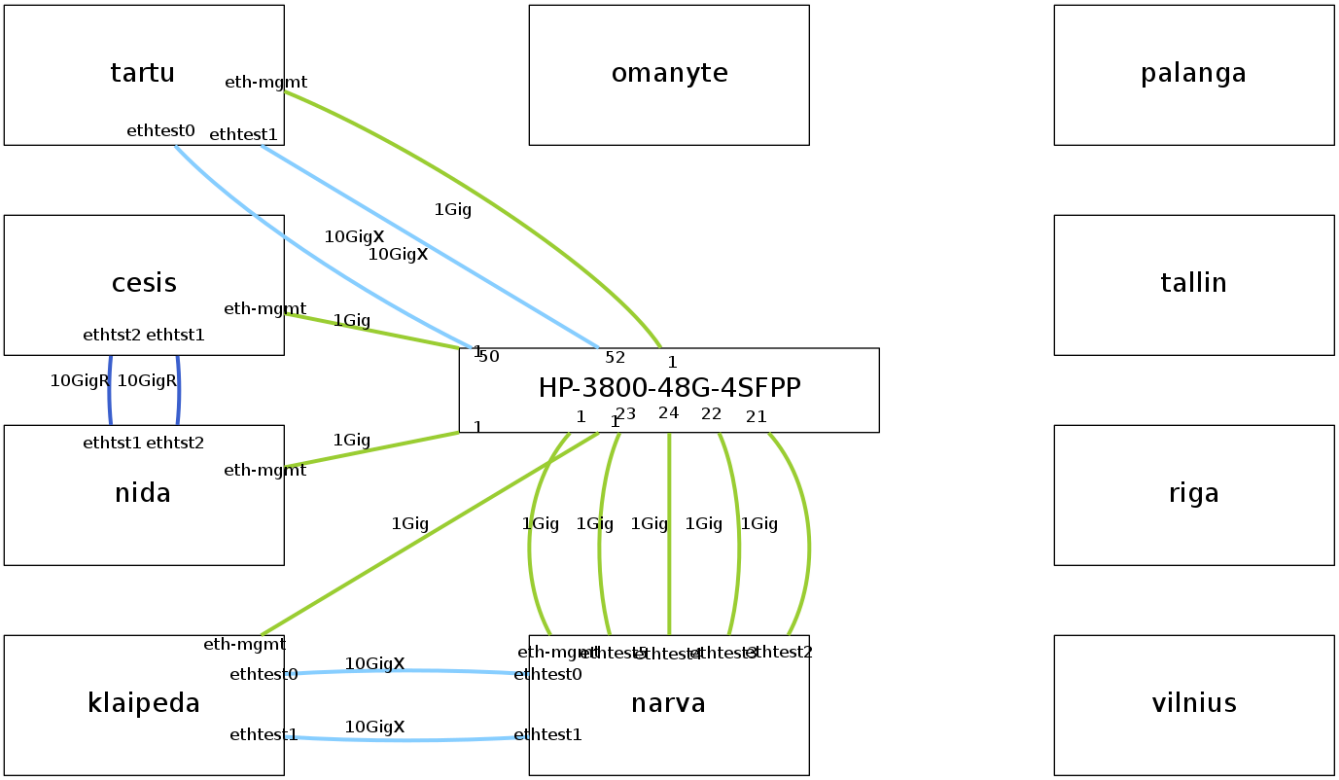

Figure 4: Generated partial topology map of the Baltikum testbed, 20.12.2015

doi: 10.2313/NET-2016-07-1_08