A Generic Software Architecture for PoE Power Sourcing Equipment

Andreas Mäkilä[1,2], Anna Friebe[2], Leif Enblom[1], Per Erik Strandberg[1], T. Seceleanu[2]
[1] Westermo Network Technologies, Västerås, Sweden
[2] Mälardalen University, Västerås, Sweden
{andreas.makila, leif.enblom, per.strandberg}@westermo.se
{anna.friebe, tiberiu.seceleanu}@mdu.se

***Abstract*—Many hardware solutions for Power over Ethernet (PoE) Power Sourcing Equipment (PSE) exist, with slightly varying feature sets. A software solution is needed for interaction with the PSEs, and for managing a power budget across several PSEs. A generic interface is desirable, as well as generic software components that can be used in support of several PSE solutions. In this paper we present a union of features and real-time requirements for three hardware solutions, and the development of a generic software architecture.**

***Index Terms*—power over Ethernet, software architecture, real-time requirements.**

I. Introduction

Power over Ethernet (PoE), a part of the IEEE 802.3 (Ethernet) standard, is a technique which allows powered devices with appropriate technology, called Power Sourcing Equipment (PSE), to deliver power to other devices which are otherwise unpowered, called Powered Devices (PD). This technique provides the advantage of not needing a separate power supply for every network device, cutting down on the need for cables and power outlets. This makes the technique especially important in rugged setups, such as outdoor surveillance cameras, where reducing the amount of cables helps remove points of failure and makes general setup and maintenance much easier.

There exist many hardware solutions for PoE PSE from many different vendors. Most of them have simple PoE circuits, able to handle individual ports. Some, though, contain so called “companion chips”, which can control the PoE circuits on a higher level. For instance, they can supervise the total amount of power consumption for all ports and take appropriate actions when the total power consumption exceeds the maximum that the device can provide. If a system does not feature these extra functions in hardware, then it falls to software to handle it, instead. This creates a unique opportunity. By creating a software API which can present a generic user interface and adapt to any combination of PSE circuits, it would allow the hardware of a system to be based on the functionality required of that system, without being restricted by the limitations of the circuits or the ability of any single-target software suite. This is the goal which this project, done in collaboration with *Westermo*, is aiming for.

In this paper, we outline work on developing the API architecture for PSE, such as can be seen in figure 1, enabling a generic interface with support for several PSE circuits. A union of features for the evaluated circuits is derived, and real-time requirements are analyzed. Four different architecture proposals are evaluated through the Architecture Tradeoff Analysis Method (ATAM).

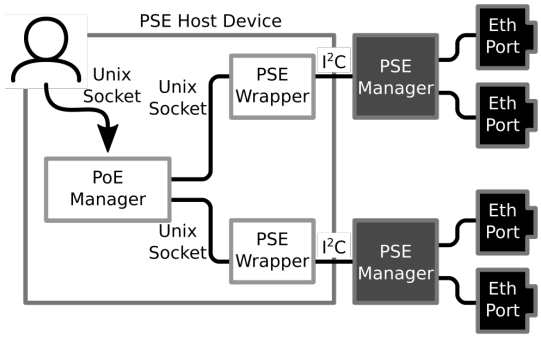


Fig. 1: An illustration of the overall structure of the final architecture.

II. Background

We briefly review here the technologies used by PoE.

Power over Ethernet. PoE is a group of standards defining methods of transmitting power over twisted-pair cables (often referred to as ”Ethernet cables”), which contain four pairs of twisted cables (of which only two are used in sub-gigabit connections). IEEE maintains three PoE standards, which govern how much power can be sent through the cables, as well as what transmission modes can be used [2].

The latter can be identified as: 802.3af - PoE or Type 1, allowing around 12.95 W to be received by the PD; 802.3at - PoE+ or Type 2, allowing 25.50 W at the PD; 802.3bt - PoE++/4PPoE, which allows either 51 W in Type 3 or 71 W in Type 4. All of these also include some ability to choose a level of power through classification and, optionally, the Link Local Discovery Protocol (LLDP), allowing the amount received to be tailored.

IEEE also maintains three standards for how the power is transmitted through the cable [2]: common-mode data pair power (*alternative/mode A*): the power is provided over the same two wire pairs used to carry the data signal on the 10 and 100 Mbit/s Ethernet variants; spare-pair (*alternative/mode*

B): the power is provided through the two wire pairs which go unused on these variants; 4-pair transmission (*4PPoE*): power is provided over all four wire pairs.

PoE and PoE+ support mode A and mode B, PoE++ Type 3 supports both modes as well as 4PPoE, while PoE++ Type 4 only supports 4PPoE.

Power Sourcing Equipment. The PSE can be installed either as an *endspan* or as a *midspan*. An endspan PSE is installed directly into the device the PD connects to (such as a switch), and is common in newly installed equipment, whereas midspans exist between the PD and the device it connects to, adding power to a previously unpowered conection, and being more common in situations where the equipment used is older and does not support PoE.

When a PD is connected to a PSE it goes through an initial power up phase, consisting of the following steps.

1. The PSE performs signature detection, using a low voltage to detect a 25 kΩ resistance present in PoE PD.

2. The PSE classifies the maximum power input of the PD by applying a higher voltage (15-20 V) and measuring the current.

3. The PSE begins supplying power.

A second power up phase may be performed, with the two devices communicating over LLDP to negotiate how much power the PD may use. The PD initiates the communication by sending its maximum and requested power to the PSE, and the PSE responds with the max power allocated to the PD. After this, the PD may use as much power as it has been allocated, with the PSE being able to request reduced power draw if needed.

PSEs contain micro controllers which perform their functions. There are options to configure them in different ways (e.g. establishing how many physical Ethernet ports there are, which standard to use for each, any restrictions on power draw, etc.) or read information about events (e.g. error interrupts, current power draw, etc.), typically using I^2C or UART. PSEs typically govern only a handful of ports, providing around 8 channels for PoE, with mode A and mode B connections using one channel while 4PPoE connections use two, resulting in between 4-8 physical ports per chip.

These micro controllers are quite basic and are unable to perform more advanced functions such as intelligent power balancing, and feature no method of inter-chip communication. Certain manufacturers (notably MicroSemi with their PD692X0 series) produce so-called companion chips, which connect the PSEs to enable these higher functions.

Link Local Discovery Protocol. LLDP is a vendor-neutral layer 2 protocol used by devices to announce their names, capabilities, and neighbours [1]. In systems using PoE it is optionally used for communication between the PSE and PD, allowing the two to negotiate the power to be delivered. LLDP provides the opportunity to send many different kinds of data with the optional *Type, Length, Value* (TLV) field. It begins with a 7 bit type ID, followed by a 9 bit length, and a data value which can be between 0-511 octets long.

Inter-Integrated Circuit (I^2C). Inter-Integrated Circuits (I^2C) is a serial communication bus standard primarily used for connecting lower speed integrated circuits to processors or micro controllers on a circuit board for short-range intra-board communication. Devices on the bus are divided into two categories: controllers, which generate a clock signal and initiate communications, and targets, which receive the clock signal and respond to messages sent from a controller. I^2C busses can be used to allow a single micro controller or processor to access register information from multiple other circuits while only requiring the use of two pins: Serial Data Line and Serial Clock Line.

User space vs. Kernel space. On most modern operation systems, including Linux-based ones, programs can live in two different parts of memory: the user space or the kernel space. The kernel space is reserved for programs which need elevated privileges and direct memory access. It includes the OS kernel and extensions, but also most of the device drivers and features such as process scheduling and memory management. In contrast to this, the user space operates on top of the kernel and houses programs which use the features provided by it. This includes the C standard library, init daemons, window managers, and all user applications.

Inter-Process Communication (IPC). Inter-Process Communication (IPC) is the broad term used to describe any mechanism which allows data to be transferred between two or more separate processes running on a single or multiple computers. IPC allow communication between user-space processes, kernel-space processes, or a combination of the two. In the development of processes that are not entirely self-contained, IPC is an imporant component. Examples of such processes are microkernels, that reduce functions in the kernel and instead provide them through other processes, and distributed computing, where different computers work together to compute a single result. IPC protocols exist in many shapes and forms, including files, TCP/IP sockets, message queues, etc. Some popular ones include: *Netlink*, which allows user space apps to communicate with kernel modules using standard socket APIs, such as those used in TCP/IP communication; *sysfs*, which creates a virtual file system that processes can read from and write to; the *Unix domain socket*, which provides an interface similar to an internet socket, but tailored for IPC on a single host; *message queues*, often used bu GUIs to receive input events from the host system.

Two IPCs are of particular note: Netlink and sysfs. Netlink is used primarily for networking functions, such as user space routing, but it also contains a generic interface for transmitting data to and from the kernel, although it requires its own protocol definition. sysfs allows communication with kernel modules through a pseudo-file system available under the */sys/* directory. It is used to access information and properties about different hardware connected to the host, such as USB, PCI, and ACPI devices, with kernel modules also able to directly create files and directories.

III. Related Work

Many modern devices featuring PoE (especially those aimed at the consumer market) feature only a couple PoE out ports with minimal configuration available, making high-level management software mostly irrelevant, which may contribute to the lack of research. Therefore, alongside existing solutions in the PoE field, solutions for adjacent technologies will also be analysed and considered.

Yokohata et al. have proposed an extension to LLDP which would allow PoE to supply power on demand [8]. Their work also provides the ability for devices to hold different priorities, with higher-priority devices causing lower-priority ones to shut down if there is insufficient power available for both. Their work seems to focus mostly on the design of the protocol used by the PSE and PD to communicate and negotiate, as opposed to the design of the API used within the PSE, and their conclusion explicitly states that the design and implementation of the PSE is a future work. They have also published another work [9], discussing and implementing an algorithm for handling power distribution.

Cumulus Networks (now owned by Nvidia) have created and maintain a Linux distribution implementing PoE with prioritized ports. However, this implementation currently supports only a handful platforms, and does not provide a generic interface. The system is also closed source, limiting the information available to make comparisons.

Sharma et al. [6] present an architecture to manage and control network energy consumption, called NEEM (Network Energy Efficiency Manager). This architecture utilized PoE for the purpose of allowing policies to be placed on the power consumption of PDs, such as turning off unnecessary devices or reducing the allocated power amounts for ports which are allocated more power than they use. The PoE policies were implemented using HP's Network Automation (HPNA) tool.

Another API aiming to improve consistency and flexibility, but in the field of network firewalls, is introduced by Singh and Singh [7]. It's proposed that the system resided in the kernel, with user space applications able to make calls to a "rule acceptor" module; the rules are stored in a "rule repository", to be used by a "rule matcher" module. The authors suggest using either input/output control (IOCTL) system calls or shared memory for communication between user space and kernel space. They also suggest that, between power downs, any configuration information should be recorded and stored by the user applications rather than the kernel module.

The development of an Internet Traffic Manager (ITM), using a unified framework of functions, and allowing operation in both user- and kernel-space, is detailed in [4]. This system aimed to be extensible and easy-to-deploy, and implemented and event driven system where, for each event generated by a packet, a set of functions is called to classify, log, process, update classes, and update control parameters. This type of unified event-driven system may be of interest in the design of the PoE API, where one could imagine events being generated by, for example, a PD being classified, disconnected, or exceeding its power limit.

IV. Method

The PSE managers considered for this work are the *Microsemi PD69200*, the *Analog Devices LTC4291*, and the *Texas Instruments TPS23880*. For testing purposes, these are all available on evaluation boards. The work of finding the feature-union of PoE solutions and determining any real-time constraints was done mostly through studies of technical documentation and discussions with company stakeholders, with practical tests of hardware (with the purpose of, for example, determining response times) being performed when necessary. The research of PoE features and real-time requirements was separate from the research and design of the API, and was largely a prerequisite for the API.

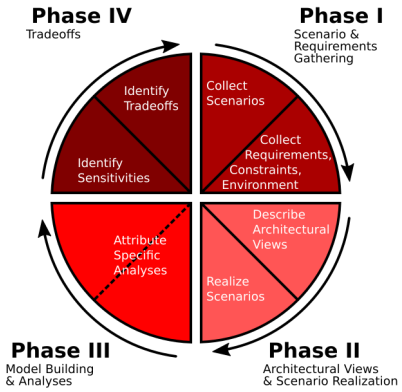


Fig. 2: An illustration of the ATAM model, inspired from [5]

Designing the API involved studies of previous similar systems, but was mostly done through iterative designs and implementations. The Architecture Tradeoff Analysis Method (ATAM)[5], [3] was used to ensure the resulting architecture fulfills the requirements placed on scalability, flexibility, and real-time deadlines. The ATAM model is divided into four phases, all but one of which are further divided into two parts [5]. Fig. 2 illustrates the model along the four phases described below.

Phase one: deals with the collection of the scenarios and requirements. Scenarios are events expected of the system, including both intentional ones (such as a server receiving a request), and unintentional ones (such as a server experiencing a reboot). Requirements are often derived from the scenarios and collected from stakeholders, and present a manner in which the scenarios are expected to be performed (e.g. the server should be able to process an event within x ms).

Phase two: describes architectural candidates for the system, based on the requirements previously identified. These views may be original or, more commonly, based on existing architectures, or on "improved" architectures - which still need to be compared with older ones. The architectures should be framed based on what they bring to the system, in terms of availability, security, modifiability, or other qualities which are

important to the system. Distinct architectural views are often used to analyse these qualities, such as module views, process views, dataflow views, etc.

Phase three: analysis of the quality attributes in isolation for each architecture, where the performance of the different scenarios is roughly determined. The exact technique to use is not important, as this step is primarily focused on finding how different architectures trade off qualities. However, the analysis should be scrutinized, and other techniques should be used on particularly sensitive areas where more detail is required, yielding a series of statements on system's behaviour.

Phase four: involves finding sensitivities and tradeoffs. Sensitivities are any modeled values which are significantly affected by a change to the architecture. Tradeoff points are elements of the architectures which affect the sensitivities when altered (e.g. the number of servers will affect both a systems availability and security, and is thus a tradeoff point).

V. Union of Features & Real-time Requirements

The creation of a union of features was primarily accomplished by reviewing relevant technical documentation for three PSE circuits: the *PD69204*, the *LTC4291*, and the *TPS23880*, along with that of the companion chip *PD69200*. The discovery of real-time requirements was primarily accomplished by performing a basic analysis and also by investigating current issues faced in the software development.

Union of Features. The union of features includes features present on the common PSEs and features present in the necessary add-on chip. These features may be originally implemented as either hardware or software solutions.

Some of the most important features we look for are *port priorities*, optional *forced PoE* (always transmit power, skip classification), and *power balancing*. These features are deemed to be of particular importance due to their current usage in PoE software[1]. They are especially important for implementing real-time requirements, as for communicating with the host computer.

PSE manager features. Table I shows features built into the given PSE managers which are considered for the purpose of this study. The *PD69200* is considered here as well, since the PSE managers which connect to it (*PD69208* and *PD69204*) cannot operate without it.

The investigated PSEs present many common features. Of these, the interrupt pin and the ability to generate interrupts are among the most important, since interrupts allow the handling of critical events without polling. The semi-auto mode most commonly describes an operation mode where classification is done automatically, but power is not supplied until accepted by the PSE host.

Notably, the LTC4291's priority system consists of the ability to tag certain ports as low-priority and, by pulling a pin up, being able to quickly shut them down. The activation of the pin must be done by software procedures, illustrating a need for a low-level, low-latency interrupt handling routine.

[1]in particular the *weos* operating system from Westermo https://www.westermo.se/solutions/weos

Feature	PD69200	LTC4291	TPS23880
Supports all 4 PoE-types	Yes	Yes	Only 3 and 4
Per-port status	Yes	Yes	Yes
Per-port priority	4 levels	2 levels	8 levels
Per-port power limit	Yes	Yes	Yes
LLDP Power Negotiation	Yes	Yes	No
Semi-auto mode	No	Yes	Yes
Forced power mode	Yes	Yes	Yes
Overcurrent protection	Yes	Yes	Yes
Overtemperature shutdown	Yes	Yes	Yes
Temperature measurement	Yes	No	Yes
Interrupt pin	Yes	Yes	Yes
I^2C	Yes	Yes	Yes
UART	Yes	No	No

TABLE I: Union of listed features for PSEs

Real-time Requirements. One of the major real-time requirements of a PSE is handling its own distribution of power. The circuits can only track their own power draw, not the draw of the entire system, and they will by default allow any PD to draw power as long as it won't put them above their own power limit. As such, in a system with multiple PSE circuits, it is possible for a device to become automatically connected despite there not existing enough free power for it. In these instances, the host device must in some way intervene before the additional power draw causes electrical faults. An example of how this can occur is illustrated in figure 3.

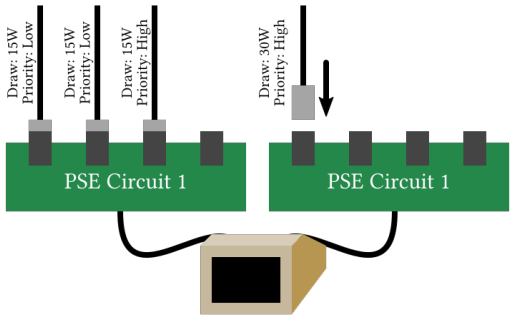
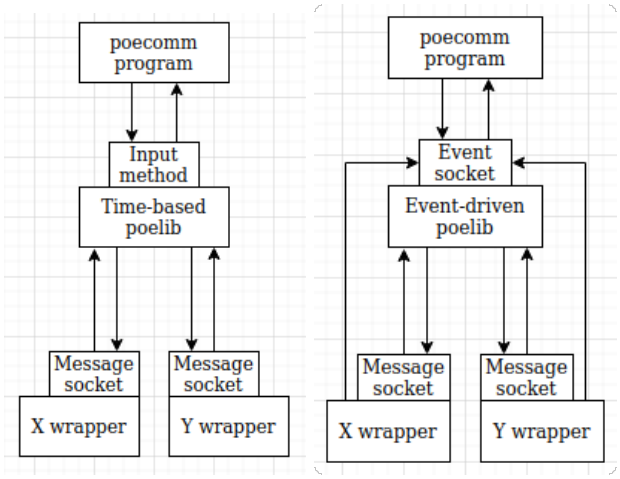


Fig. 3: The power distribution issue: when a new, high-priority, device is connected (the right one), the system's power budget will be exceeded. The two low-priority ports should shut down, thus providing enough power for the newly connected device.

In the case of rugged devices unable to use active cooling, these have an additional real-time requirement in the form of overheating. Since their devices are completely enclosed and thus don't have any fans or air intakes, cooling is only done passively through heat sinks on the outside of the devices. This means that PSEs may need to reduce their provided power or even disable some PDs in order to cool off. The real-time requirements of this are not as strict as those of excessive power draw mentioned above, but extended periods of overheating can still result in damage to the devices.

VI. Developing the API Architecture

After collecting the current PoE solution features, and completing the PSE manager functions and the real-time



(a) Polling architectures

(b) Event-driven architectures

Fig. 4: Module structures of the event-driven and polling architectures

requirements, the PoE API construction started.

Four architectures were designed, all residing user space. Two of the architectures are *Direct*, this means that they support only the functionality available in the circuits in use at a certain time. The two *Generic* architectures on the other hand provide software emulation for some functionality when it is not supported in hardware. Most notably this includes functions for a semi-auto mode even if no such mode is officially supported for the device. The Direct and Generic architectures are further divided into *Polling* and *Event-driven* versions.

All architectures provide wrappers for each type of PSE at the lowest level, as can be seen in Fig. 4. These host a socket server which would receive generic commands and translate them to PSE-specific API calls. Above the wrappers is the PoE library (poelib), which creates a generic C library front-end for sending and receiving to the socket servers. This library also implements commonly requested functionality such as priority shutdown handling and automatic restarting of ports. At the highest level is the PoE command utility (poecomm), which provides a terminal interface for modifying parameters such as priorities and calling poelib functions.

The generic architectures allow more general feature implementation at the cost of a slightly reduced ability to use circuit functions directly.

The Event-driven architectures provide fast response to user commands and less idle resource usage, by only performing functions when prompted by either the user sending a command or a wrapper reporting an event. The polling architectures feature a similar overall structure to their event-driven counterparts, but operate in timed loops through polling for updates and commands instead.

Attribute Models. The systems were primarily measured on four different criteria, described here as follows.

Recipient nr.	Arch.	Modify.	Univers.	Flexib.	Scalab.	Sum
1	Gen Poll	5	3	3	4	15
	Dir Poll	3	2	4	2	11
	Gen Event	4	3	3	4	14
	Dir Event	4	2	3	2	11
2	Gen Poll	4	3	3	4	14
	Dir Poll	4	3	2	2	11
	Gen Event	4	3	3	4	14
	Dir Event	4	3	3	2	12
3	Gen Poll	4	4	3	3	14
	Dir Poll	3	3	3	3	12
	Gen Event	4	4	3	4	15
	Dir Event	3	3	2	3	11
4	Gen Poll	4	3	3	3	13
	Dir Poll	5	4	4	5	18
	Gen Event	4	3	3	3	13
	Dir Event	5	4	3	5	17

Fig. 5: Results of the expert questionnaire.

Modifiability is the ability to modify the code to fit with the needs of the user and the system (such as unique prioritization schemes or scheduling of port power). The code should preferably be as easy to modify as possible to allow it to adapt to new or unique cases.

Universality is the ability of similar code to be run on different systems with the same effect (for example, port prioritization may always work the same way for *poelib*). More universal code is better able to handle different setups without needing to be modified, reducing the effort required to adapt it.

Flexibility is the ability of the code to adapt to new PSEs and their unique function set. More flexible code is better able to adapt to new circuits and make use of their unique functions.

Scalability is the ability of the code to continue working efficiently as more PSEs, incuding ones of different models, are added (Such as two different models with different limits on class). More scalable code allows the device to continue operating efficiently as more PSEs are added, allowing the requirements of the device to be unrestricted by the capacity of the code.

Analysis of Architectures. Here we present an analysis of the different architecture options in regard to the different attributes. This was done primarily through a questionnaire, with questions based on plausible scenarios the system may experience which test the required attributes (so-called ”realization of scenarios” in the ATAM model). It was also done through subjective analysis of the architectures and, later on, partial implementations.

Analysis of Questionnaire Results. The questionnaire contains descriptions of four candidate architectures (Generic Polling, Direct Polling, Generic Event-driven, and Direct Event-driven), with questions intended to cover each of the four attributes and a text field for giving any further thoughts. The responses received for the attributes are shown in Fig. 5, along with the sum of the values given to each attribute, and visualized in Fig. 6. The generic architectures were preferred over the direct architectures and there was no preference towards either polling or event-driven, when addressing the generic architectures.

In-depth Analysis. It is hard to determine a superior architec-

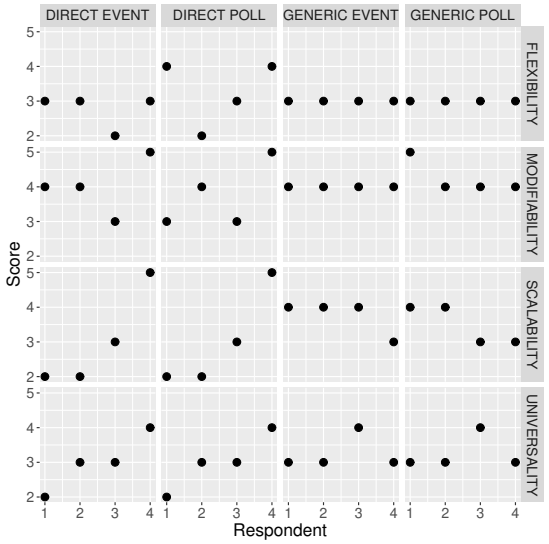


Fig. 6: Visualization of the questionnaire scores per architecture, quality attribute and respondent.

ture between the direct and generic. The generic architectures appear better overall, with better modifiability (thanks to only having one set of functions), scalability (since the manager doesn't need to be rewritten for new PSEs or combinations of PSEs) and universality (since much code can be reused, owing to the universal featureset). However, it also has lower flexibilty (since it can't always utilize unique features of PSEs), it relies somewhat on emulation, and it may experience problems with less than fully featured PSEs.

The event-based architectures also present a possible trade between better scalability (since only reacting to events reduces idle traffic) and response time, but possibly lower modifiability (since event-based code will have more considerations than the simpler polling).

Tradeoffs. One of the initially assumed tradeoffs in the system was that between a universal API and a flexible API, since a more universal API would need to hide some parts of the PSEs' functionality in order to implement features, which would make it less flexible to unique functions. The opposite applies to a flexible API, which cannot be as universal since its functions are exposed more directly, and these functions may not be the same between PSEs.

In the questionnaire two participants rated universality as being higher on the generic variants and lower on the direct ones, while one rated it as equal on all and one as being higher on the direct variant. Participants also rated flexibility seemingly unrelated from their rating on universality.

In general the participants favored the generic variants over the direct variants in every way, and had a slight preference towards either the polling or event-driven one, with one participant rating polling with better modifiability and another rating event-driven with better scalability.

VII. Conclusion and Future Work

The goal of this study was to create an API architecture for Power over Ethernet (PoE) which would allow any combination of PSEs to work together, for the purpose of allowing more open PSE hardware selection by removing the limitations of the PSEs or their associated software suites.

From the work performed on discovering features and analysing the different architectures, four primary candidates have been identified, which allow excellent compatibility with PSEs of almost any feature set. These architectures can also be easily expanded to handle new PSEs, while reducing the amount of real-time requirements handled by the host computer by delegating these to the PSEs themselves.

Although disregarded for this work due to time constraints and a seeming lack of benefits, APIs residing partly or fully in kernel space may still be of interest for future research, if cases are found where this is required to meet real time requirements.

Although a best effort was made in discovering current PSE feature sets, only a small number of circuits were analyzed in depth. There exist many which were not thoroughly analyzed, which may work in ways which contradict the findings on common feature sets described in this work, and would thus require a different API architecture.

References

[1] IEEE Standard for local and metropolitan area networks– station and media access control connectivity discovery corrigendum 2: Technical and editorial corrections. *IEEE Std 802.1AB-2009/Cor 2-2015 (Corrigendum to IEEE Std 802.1AB-2009)*, pages 1–68, 2015.

[2] IEEE Standard for Ethernet Amendment 2: Physical Layer and Management Parameters for Power over Ethernet over 4 pairs. *IEEE Std 802.3bt-2018 (Amendment to IEEE Std 802.3-2018 as amended by IEEE Std 802.3cb-2018)*, pages 1–291, 2019.

[3] Mario Barbacci, S. Carriere, Peter Feiler, Rick Kazman, Mark Klein, Howard Lipson, Thomas Longstaff, and Charles Weinstock. Steps in an architecture tradeoff analysis method: Quality attribute models and analysis. 06 1998.

[4] G. Diamant, L. Veytser, I. Malta, A. Bestavros, M. Guirguis, Liang Guo, Yuting Zhang, and Scan Chen. itmbench: generalized api for internet traffic managers. In *IEEE Global Telecommunications Conference Workshops, 2004. GlobeCom Workshops 2004.*, pages 306–311, 2004.

[5] Rick Kazman, Mark Klein, Mario Barbacci, Thomas Longstaff, Howard Lipson, and S. Carrière. The architecture tradeoff analysis method. pages 68–78, 01 1998.

[6] Puneet Sharma, Sujata Banerjee, Deniz Demir, Srikanth Natarajan, and Swamy Mandavilli. Neem: Network energy efficiency manager. In *2012 IEEE Network Operations and Management Symposium*, pages 623–626, 2012.

[7] Maninder Singh and Maninder Singh. A powerful easy-to-use packet control api for linux. In *2009 International Conference on Advances in Recent Technologies in Communication and Computing*, pages 146–148, 2009.

[8] Masaya Yokohata, Tomotaka Maeda, and Yasuo Okabe. An extension of the link layer discovery protocol for on-demand power supply network by poe. In *2013 27th International Conference on Advanced Information Networking and Applications Workshops*, pages 1612–1616, 2013.

[9] Masaya Yokohata, Tomotaka Maeda, and Yasuo Okabe. Power allocation algorithms of poe for on-demand power supply. In *2013 IEEE 37th Annual Computer Software and Applications Conference Workshops*, pages 517–522, 2013.