X-Leep: Leveraging Cross-Layer Pacing for Energy-Efficient Edge Systems

Stefan Reif
Friedrich-Alexander University Erlangen-Nürnberg
reif@cs.fau.de

Benedict Herzog
Friedrich-Alexander University Erlangen-Nürnberg
benedict.herzog@cs.fau.de

Pablo Gil Pereira
Saarland Informatics Campus
gilpereira@cs.uni-saarland.de

Andreas Schmidt
Saarland Informatics Campus
andreas.schmidt@cs.uni-saarland.de

Tobias Büttner
Friedrich-Alexander University Erlangen-Nürnberg
tobias.buettner@fau.de

Timo Hönig
Friedrich-Alexander University Erlangen-Nürnberg
thoenig@cs.fau.de

Wolfgang Schröder-Preikschat
Friedrich-Alexander University Erlangen-Nürnberg
wosch@cs.fau.de

Thorsten Herfet
Saarland Informatics Campus
herfet@cs.uni-saarland.de

ABSTRACT

Edge systems enable large numbers of embedded nodes to communicate in order to cooperate towards achieving a shared goal. However, such systems operate under both timeliness and energy-efficiency constraints. This paper proposes X-Leep, a run-time system that detects the pace of the system, supporting Internet-of-Things and Edge scenarios. X-Leep adapts the local processing speed accordingly, considering time-related and energy-related constraints. Our evaluation shows that X-Leep increases energy efficiency compared to state of the art with only a minor effect on the quality of service.

CCS CONCEPTS

• **Computer systems organization** → **Embedded and cyber-physical systems**; • **Networks** → *Network protocols*; • **Hardware** → **Power estimation and optimization**.

KEYWORDS

Energy Efficiency, DVFS, Real-Time Communication, Cross-Layer Pacing, Edge Computing, Internet of Things

ACM Reference Format:
Stefan Reif, Benedict Herzog, Pablo Gil Pereira, Andreas Schmidt, Tobias Büttner, Timo Hönig, Wolfgang Schröder-Preikschat, and Thorsten Herfet. 2020. X-Leep: Leveraging Cross-Layer Pacing for Energy-Efficient Edge Systems. In *The Eleventh ACM International Conference on Future Energy Systems (e-Energy'20), June 22–26, 2020, Virtual Event, Australia.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3396851.3402924

e-Energy'20, June 22–26, 2020, Virtual Event, Australia

ACM ISBN 978-1-4503-8009-6/20/06…$15.00
https://doi.org/10.1145/3396851.3402924

1 INTRODUCTION

Edge computing [6, 7] is key to ensure low latency and high reliability for application domains in the Internet of Things (IoT) [26]. With the increasing amount of data acquisition (i.e., sensor data) and pre-processing of interconnected nodes at the edge (i.e., data computation of IoT devices), it becomes of greatest importance to consider the energy demand of individual applications and usage scenarios too. Activities at the software level, which originate from various layers of the software stack, lead to energy demand of the underlying hardware components.

In the spirit of work from operations research [17], parts of this energy demand can be considered as different forms of *waste*, as a careful look reveals that these can be avoided by design-time as well as run-time adaptations. To achieve this in Edge scenarios, it must be ensured that operations at the application level (i.e., user space) and at the operating-system level (i.e., kernel space) adapt to the run-time behaviour of the network stack [1, 13]. Cross-layer optimisations have been subject to research [12, 22] and in-depth analysis in previous work. Packet pacing [9, 11, 20, 25] and, in particular, cross-layer pacing have been proven to be an effective operating scheme to reduce jitter and improve latencies in large-scale, networked systems.

In this paper we explore ways to leverage cross-layer pacing to improve energy efficiency in edge computing scenarios. Our approach exploits the information gathered by cross-layer pacing at different layers of the system (i.e., application, network, and operating system) . The collected run-time data is used to improve the operations within the overall system dynamically at run-time. For example, X-Leep adjusts the voltage and frequency of the CPU to eliminate slack time while ensuring that certain quality of service requirements, for example response time of the edge system, are not violated. X-Leep achieves this without requiring application code annotations or changes, but utilising already available information.

The contributions of this paper are threefold. Firstly, we discuss and present the X-Leep approach. Our approach addresses current

shortcomings in edge scenarios, where energy-efficiency improvements are untapped as cross-layer concerns are not considered. Secondly, we contribute a prototype implementation of X-Leep and outline design considerations. Thirdly, we evaluate X-Leep and discuss the results of our analysis.

The structure of the paper is as follows. Section 2 presents background information and the concepts underlying X-Leep. The implementation of X-Leep is discussed in Section 3. Section 4 evaluates the current X-Leep implementation. Related work is discussed in Section 5 and Section 6 concludes the paper.

2 BACKGROUND

Energy efficiency plays an important role in the IoT domain, so that a correct operation of battery-powered devices is ensured and a sustainable deployment can be achieved. Sustainability is particularly relevant in this case, given the mere amount of devices expected to be deployed[1], which makes desirable, if not mandatory, to follow the *Green IT* principles [16]. Data samples captured by sensors at the edge have strict latency constraints when monitoring physical processes to keep them under control [2], thus requiring timely delivery in order to guarantee the stability of such processes. Therefore, since the value of information is reduced as time goes by, the data loses value while it is buffered. This may result in the waste of precious resources in case the data does not reach its destination in time. For example, the energy used to forward a packet is wasted if the packet can no longer arrive in time at the receiver.

Buffers can be found in end nodes and in-network nodes to allow for asynchronous process communication and to cope with unavoidable bursts. However, they have the side effect that queues are created when the communication and processing steps do not run at the same pace[2], which we define as the time a given step needs to process a certain data unit. This may result in *bufferbloat* [8], i.e. persistent queues that increase the end-to-end latency, and therefore should be avoided.

In any system there is always at least one step which takes the longest to process a data unit (e.g., communication channel, receiver application, etc.). We denote the pace of such a step as the *bottleneck pace*. In order to keep buffers empty, we have implemented *cross-layer pacing*, which measures the pace of every step and communicates this information to all the other steps, so that they can be aware of the bottleneck pace and adapt accordingly to minimise the resource footprint. Figure 1 shows the relevance of the cross-layer pacing approach for the control scenario. The unpaced sensor samples a signal at a certain pace and then transmits the samples over a channel, which in this case is the bottleneck with a pace twice the sensor pace. Not only is the unpaced controller not able to recover the original signal, but packets are eventually dropped due to buffer overflows. In contrast, the paced sensor can use the bottleneck pace information to adapt its sampling rate, which results in the controller reconstructing the original signal in time. In order to keep buffers empty, all the steps preceding the bottleneck *must* run at the bottleneck pace, otherwise queues

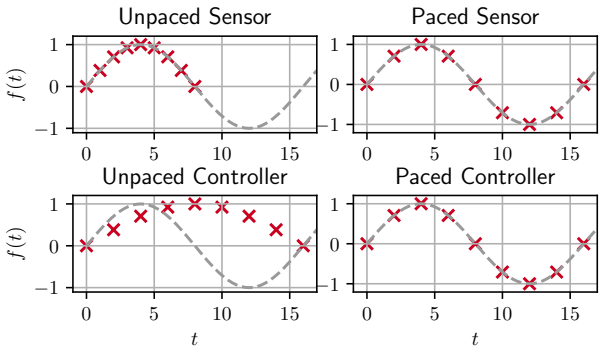

Figure 1: When the sensor uses cross-layer pacing, the sampling rate is adapted to match the bottleneck—achieving high fidelity and avoiding data loss due to buffer overflows.

appear as a result of the mismatch between input and output rate at the buffer. The steps behind the bottleneck *may* decide to adapt to it, for instance to optimise the resource usage (e.g., slow down the CPU frequency to save energy) or to minimise the end-to-end delay (e.g., run preparatory tasks in advance).

Cross-layer pacing has been implemented in the *Predictably Reliable Real-time Transport* (PRRT) protocol [10, 21], which provides a latency-aware, partially reliable, in-order datagram delivery service. The PRRT layer[3] can measure the application and network pace, and communicate this information end-to-end, so that the sender and receiver are aware of the bottleneck pace, which makes it ideal for the implementation of cross-layer pacing.

PRRT provides several means for the application to use cross-layer pacing. First, the application can query the socket for the bottleneck pace and in turn adapt its internal parameters to meet it, thus becoming *pacing-aware*. Second, the transport protocol can measure the application's pace and artificially delay the send calls when it detects the application runs too fast. This approach is transparent to the application, as legacy applications need no modifications. Finally, the application can use *synchronous send calls*, which block until the next data packet can be created and transmitted. This approach requires the application to have periodic behaviour, so that its pace can be measured and accordingly adapt the blocking behaviour to produce just-in-time processing.

Besides cross-layer pacing, PRRT also implements error control to cope with the packet losses introduced by the communication channel. It implements an interface for the application to state its latency requirement and packet error rate tolerance, so the error control function, which is implemented using a *Hybrid ARQ* (HARQ) scheme, can be optimised to reduce the amount of redundancy and meet the application's latency and error rate requirements. The HARQ scheme also leverages information about the communication channel to fine-tune its parameters, such as the propagation delay, data rate and packet loss rate. The propagation delay is estimated following a similar approach to that in NTP [15], while the IETF

[1] "Gartner Says 5.8 Billion Enterprise and Automotive IoT Endpoints Will Be in Use in 2020", https://www.gartner.com/en/newsroom/press-releases/2019-08-29-gartner-says-5-8-billion-enterprise-and-automotive-io

[2] The pace behaves similarly to energy demand, i.e. lower is better (fast pace = low value, slow pace = high value).

[3] Typically, the PRRT layer is the transport layer, but it does not require specific services from lower layers except datagram-delivery and process-multiplexing, so technically it can run directly on top of, e.g. Ethernet or Wi-Fi.

draft on delivery rate estimation [4] has been followed for the data rate estimation. Finally, the receiver keeps track of received packets to estimate the packet loss rate. The application constraints and channel information are used to decide the number of redundancy packets to be generated, which can be transmitted either proactively, using forward error coding (FEC), or as a reaction to receiver's feedback, using automatic repeat request (ARQ). Therefore, given the target latency specified by the application, the HARQ scheme balances the amount of time budget that is used for ARQ and FEC.

Finally, congestion control is implemented to avoid network congestion and ensure a fair share of network resources. The implemented algorithm is based on BBR [3], which tries to operate at the *Bandwidth-Delay Product* (BDP) to maximise the data rate while minimising the delay. The network *round-trip time* and *bottleneck data rate* are estimated as described above to calculate this operating point. Besides maintaining the *congestion window*, BBR also controls the sending rate by pacing the packet transmission to meet the available data rate and keep in-network buffers empty.

3 IMPLEMENTATION

In this section we present the implementation of `X-Leep`, a runtime system to leverage network-level knowledge in order to select a power-efficient CPU frequency. `X-Leep` operates between the transport protocol and the operating system and adjusts the execution speed to the most power-efficient value. The optimal execution speed in our case is the lowest possible execution speed without deadline violations, and hence the lowest possible power demand. Therefore, `X-Leep` configures the dynamic voltage and frequency scaling (DVFS) algorithm of the operating system to set the CPU frequency accordingly.

In order to collect application knowledge, `X-Leep` provides an application interface. The layers above `X-Leep` are expected to report the current period length (t_{period}) and the actual required execution time ($t_{execution}$) at the end of each period to `X-Leep`. If the application does not report this data, the network protocol collects the required information and provides it to `X-Leep`. From these two values, `X-Leep` can determine whether changes to the execution speed are necessary. If $t_{execution}$ is considerably lower than t_{period} the application runs too fast and can be slowed down. In contrast, if $t_{execution}$ is equal or higher than t_{period} the application violates deadlines and needs to speed up its execution.

`X-Leep` does not only consider the last application report, but a configurable number of past reports is kept in a sliding-window average filter. This allows to track changes over time and reduces the influence of outliers. Furthermore, we enforce that a frequency change can only happen after a minimal amount of time has passed since the last change, thereby avoiding flip-flopping between CPU frequencies when the sensing rate is higher than the actuation rate, which also allows to amortise the overhead caused by CPU frequency changes. Non-controllable factors, for example operating-system noise, may increase the number of narrow deadline misses, as this approach tries to make the execution speed converge as close as possible to the period length. Consequently, small increases in the execution time would lead to deadline violations. This is the reason why `X-Leep` includes a configurable safety margin to optimise for a slightly reduced period length and to reduce the probability for narrow deadline violations.

The application reports are used to determine the currently optimal execution speed and thus frequency. The optimal frequency is calculated by the following equation, where t_{period} and $t_{execution}$ are the values as reported by the application and f_{cur} is the currently applied CPU frequency.

$$f_{opt} := f_{cur} \times \frac{t_{execution}}{t_{period}}$$

Assuming that the execution speed scales linearly with the CPU frequency, the equation calculates the optimal CPU frequency in one step. This assumption is generally true for CPU-bound applications, where the execution speed depends only on the CPU frequency. However, for memory-bound applications and applications including I/O, the execution speed scales sub-linearly with the CPU frequency. In these cases, the algorithm approaches the optimal CPU frequency in several steps and eventually reaches the optimal frequency.

Furthermore, this algorithm automatically copes with deadline violations. If for one period the deadline is violated, that is $t_{execution}$ is greater than t_{period}, the fraction solves to a value greater than 1 and the CPU frequency is increased again. This allows continuous adaptations at run-time even for changing environments that require different epoch lengths and execution speeds. This control algorithm, together with suppressing of flip-flopping between two CPU frequencies, allows `X-Leep` to be robust—assuming the underlying cross-layer pacing implementation is robust as well.

The integration of `X-Leep` in PRRT can be achieved either by adding calls into `X-Leep` directly from an application or by extending the PRRT implementation itself. The advantage of a PRRT integration is the utilisation of application-specific knowledge without changes to the application itself. The only prerequisite is the use of PRRT in the application.

In our implementation, `X-Leep` is integrated into the pacing mechanism of the sender's side of PRRT. Specifically, after each transmission PRRT calculates whether a sender node has to be slowed down and hence has knowledge about the actual execution time and period length. Consequently, PRRT reports these values on behalf of the application to `X-Leep`.

Application reports are decoupled from the actual CPU frequency changes. This allows for different backends that are responsible for changing the CPU frequency, which are tailored to the available hardware features, respectively. For our system we use the Linux `userspace` governor, which allows DVFS configurations from user space utilising the `sysfs` interface. Initially, the highest available CPU frequency is used. Once the first application reports are collected, the frequency is dynamically adapted as described above.

4 EVALUATION

The evaluation setup considers a typical edge-computing use case where an embedded node periodically samples a sensor, pre-processes the data, and transmits it to an edge-located node. In this scenario, the embedded node is strictly power-constrained.

4.1 Evaluation setup

The setup consists of three network nodes— sender, receiver, and controller—connected via Ethernet. The controller has administrative tasks, and the sender and receiver transmit a fixed number of packets for each experiment. We only measure the energy demand of the sender node to evaluate its ability to adapt to the system pace[4]. The energy measurements are conducted with a LTC2991 power monitor platform connected to a microcontroller for periodic sampling. Both the sender and the receiver nodes are Raspberry Pi 3B+, running Raspbian 10, with a patched Linux kernel to enable fine-grained DVFS settings.

We synchronise clocks before the experiment using NTP, but disable clock synchronisation during the experiment to avoid erratic clock jumps. Both the sender node and the receiver node run dedicated benchmarking applications. On the receiver node, the application issues sleep calls for a configurable delay, which resembles load-induced delays. On the sender node, a dummy function of configurable length is executed, which resembles data pre-processing computations. The reason behind the two different load types is that the experiment evaluates the energy efficiency of the sender node—the receiver-side load is independent of the processor frequency and thus repeatable, whereas the sender-side load scales realistically with the processor frequency. We specifically measure the execution time of the sender-side workload at all available processor frequencies, which allows us to choose scenario parameters where we know the bottleneck. This knowledge, however, is not supplied explicitly to the system—instead, PRRT has to detect the timing at run-time and provide it to `X-Leep`.

Several DVFS strategies are compared in the evaluation. First, the Linux `ondemand` governor adapts the processor frequency to its utilisation. This strategy does slow down the sender in the case where it runs too fast. The reason is cross-layer pacing that forces the sender to wait if it produces data items too quickly. However, this strategy is unaware of application-level and network-level timing requirements. Second, each `fixed` frequency in the range of 0.6 GHz to 1.4 GHz is evaluated. Third, `X-Leep` adapts the processor frequency based on information provided by the network protocol, as described in Section 3.

For each scenario, the sender transmits 10 000 packets, which is enough to detect network and receiver paces, and to adapt accordingly. In particular during the start-up phase, packet loss is expected if the sender operates too fast, as it is not aware of receiver-side timings yet. Therefore, this relatively large number of packets reduces the unavoidable noise caused by the experiment warm-up phase. After completing an evaluation, the controller node collects all node-local measurement data for further processing.

Several different scenarios are evaluated. These scenarios represent a variety of possible system bottleneck types. The detailed configurations are outlined in Table 1.

`s*` Sender node is the bottleneck. This is achieved by a large sender-side delay and a small receiver-side delay. The sender should run at full speed to achieve optimal system throughput and minimal inter-packet times.

[4]The receiver node is implicitly paced as it cannot process data faster than provided by the sender node.

Table 1: Summary of evaluation scenario configurations with sender-side (`s*`), receiver-side (`r*`), mixed (`m*`), and varying (`v*`) system bottlenecks.

Name	Sender cpu load	Receiver delay
s1	400	1000
s2	700	2000
r1	700	7000
r{2–6}	{500,600,700,800,900}	10 000
m1	300	1000
m{2–4}	{400,500,600}	2000
m{5–7}	{700,800,900}	5000
m{8–9}	{800,900}	7000
v{1–3}	{700,800,900}	5000 → 10 000
v{4–6}	{700,800,900}	10 000 → 5000

`r*` Receiver node is the bottleneck. This is achieved by a large receiver-side delay and a small sender-sided delay. The sender should run at minimum speed because the receiver cannot process packets fast enough.

`m*` Receiver node is the bottleneck. To achieve this, the sender delay is slightly lower than the receiver-side delay, which means that the two paces match if the sender operates slightly slower. The sender should run at neither minimum nor maximum speed, but somewhere in-between, so that both nodes operate with the same pace.

`v*` Receiver timing varies in the middle of the experiment, causing the bottleneck to change. This is achieved by changing the receiver-side delay during the experiment. The sender should initially adapt to the receiver pace, then later re-adapt when the receiver delay changes. In these scenarios, none of the fixed frequency configurations is optimal.

4.2 Timeliness Evaluation

The general strategy of `X-Leep` is to slow down until the point where all available time of the packet processing period is utilised. Figure 2 summarises the processor-time utilisation for all evaluated scenarios. Since the scenarios cover a large variety of different timings, we normalise all measurement values. The results show that `X-Leep` slows down more aggressively than ondemand, but the sender-side processing finishes in time (i.e., the value is below 0) in most scenarios. Running at maximum speed (i.e., 1.4 GHz fixed) causes poor utilisation in most scenarios, except for `s*` where the sender is the bottleneck.

Figure 3 compares the deadline misses of all strategies. In general, running at the slowest processor frequency (i.e., 0.6 GHz fixed) results in many deadline misses, except for the `r*` scenarios where the receiver is the bottleneck despite slowing down the sender to its minimum speed. For `X-Leep`, the number of deadline misses is slightly lower, compared to the `ondemand` governor, which adapts the processor frequency to the utilisation but is unaware of application timing requirements.

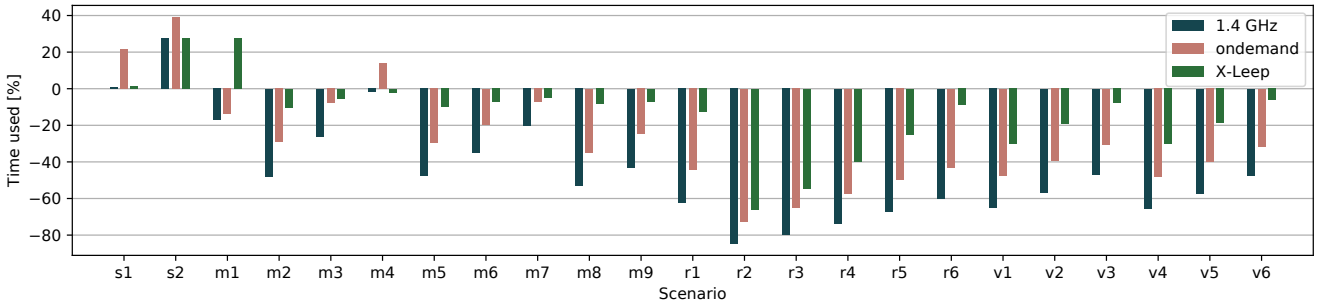

Figure 2: Amount of time used for sender-side computations, normalised to the period. Goal: stay close to but below 0.

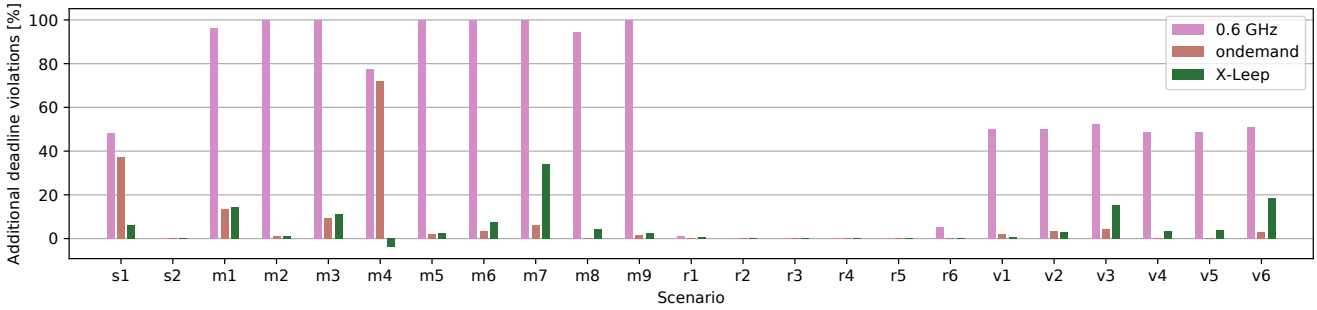

Figure 3: Increase in deadline violations, normalised to the execution at 1.4 GHz fixed.

4.3 Energy Efficiency Evaluation

A suitable metric to evaluate the power efficiency compares the utility of the system to its cost. We therefore define the power efficiency as the quotient of the packet rate and the power draw (i.e., higher is better).

$$\text{Efficiency} = \frac{\text{packet rate}}{\text{power draw}} = \frac{\#\,\text{Packets}}{\text{Energy}}$$

Figure 4 compares the energy efficiency of all evaluated strategies, normalised to the execution at 1.4 GHz fixed. First, the figure shows that 1.4 GHz is the most efficient fixed frequency in some scenarios, but also the least efficient one in some other scenarios. This finding strengthens our point that dynamic frequency adaption is crucial for energy efficiency in highly adaptive networking systems. Both the `ondemand` governor and `X-Leep` are often close to the optimal fixed frequency, indicating that both strategies tend to converge to near-optimal frequencies. In comparison, the `X-Leep` strategy is more efficient than `ondemand`. In the `v*` scenarios where the bottleneck changes during the experiment, `X-Leep` is also more power-efficient than the best fixed frequency.

4.4 Analysis

In summary, `X-Leep` is able to utilise network-related information to improve the energy efficiency of edge nodes. It is, in summary, more energy-efficient than existing solutions while also adhering to whole-system timing requirements. As expected, fixed-frequency strategies in general fail to adapt to varying network properties.

5 RELATED WORK

Dynamic voltage and frequency scaling (DVFS) techniques have been previously used to optimise the energy efficiency of soft real-time systems. Simunic et al. [23] adapt to varying execution times of a multimedia-processing embedded system and thereby optimise the energy demand. Their approach yields significant improvements, but requires an application-specific stochastic model. In contrast, `X-Leep` is not limited to a specific application, but poses a more generic approach for different applications.

More recently, Martins et al. [14] propose a system for multiple workloads and CPU cores. Their system exploits available slack time by changing the execution speed. However, they require applications to report available slack times to their system and thus require changes to applications. Furthermore, they rely on simulations instead of actual energy measurements.

Rausch et al. [19] propose an energy-aware cluster architecture for edge computing. They use an energy-aware cluster management software to balance between application responsiveness and energy demand of an edge-located computing cluster, depending on the client-induced load. PRRT with `X-Leep`, in contrast, uses the bottleneck information to identify the optimal processing speed.

There has also been research for hard real-time systems by Pillai et al. [18]. They rely on a previously determined worst-case execution time (WCET) and adapt the execution speed to meet this WCET. However, this approach requires a statically observable WCET and is not able to cope with dynamic changes to the WCET.

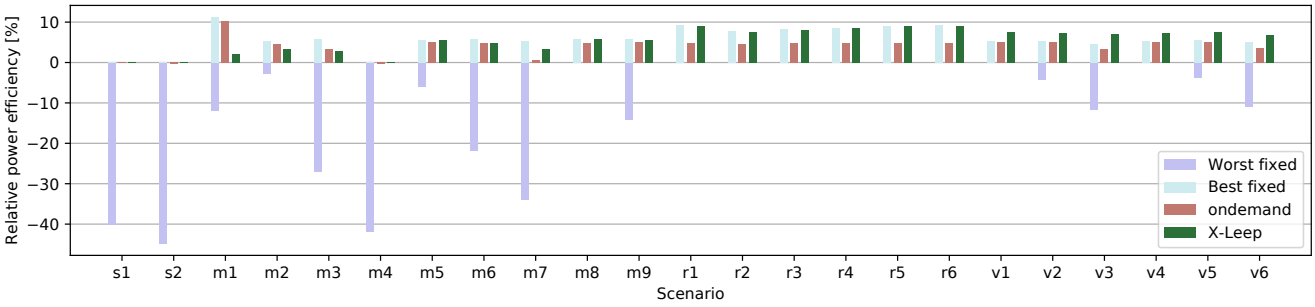

Figure 4: Power efficiency comparison, normalised to the execution at 1.4 GHz fixed.

In the networking domain, there are related approaches that aim for energy-efficient communication at the edge. Aves [5] considers this a deployment problem, i.e. answering *where* the different parts of a networked sense-process-act system should be executed—thereby optimising energy demand. Other solutions for data centre networks use power-saving features of network devices and apply traffic engineering to find appropriate transmission schedules and routes that minimise energy demand [24]. As `X-Leep` works at the end-hosts, both approaches represent orthogonal solutions to ours in a way that either potentially benefit from the other.

6 CONCLUSION

This paper has shown that `X-Leep` utilises network-provided information to slow down individual edge nodes, in order to improve energy efficiency. The evaluation demonstrates that PRRT adapts to all bottleneck scenarios, thereby enabling `X-Leep` to improve the energy efficiency. While the approach presented in this paper uses a relatively simple heuristics to select frequencies, future work will apply more elaborate energy models for additional energy savings.

ACKNOWLEDGMENTS

The work is supported by the German Research Foundation (DFG) as part of SPP 1914 "Cyber-Physical Networking" under grants HE 2584/4-2 and SCHR 603/15-2.

REFERENCES

[1] R. Bolla, R. Bruschi, F. Davoli, and F. Cucchietti. 2011. Energy Efficiency in the Future Internet: A Survey of Existing Approaches and Trends in Energy-Aware Fixed Network Infrastructures. *IEEE Communications Surveys Tutorials* 13, 2 (2011), 223–244.

[2] Michael S Branicky, Stephen M Phillips, and Wei Zhang. 2000. Stability of networked control systems: Explicit analysis of delay. In *Proc. of the 2000 American Control Conference.*, Vol. 4. IEEE, 2352–2357.

[3] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-based congestion control. *ACM Queue* 14, 5 (Dec. 2016), 50:20–50:53.

[4] Yuchung Cheng, Neal Cardwell, Soheil Hassas Yeganeh, and Van Jacobson. 2017. Delivery Rate Estimation - IETF DRAFT. (2017).

[5] Roshan Bharath Das, Marc X Makkes, Alexandru Uta, Lin Wang, and Henri Bal. 2019. Aves: A Framework for Energy-efficient Stream Analytics across Low-power Devices. In *IEEE Big Data 2019*. IEEE Computer Society.

[6] A. Davis, J. Parikh, and W. E. Weihl. 2004. Edgecomputing: Extending Enterprise Applications to the Edge of the Internet. In *Proceedings of the 13th International World Wide Web Conference (Alternate Track, Papers and Posters) (WWW Alt. '04)*. 180–187.

[7] Pedro Garcia Lopez, Alberto Montresor, Dick Epema, Anwitaman Datta, Teruo Higashino, Adriana Iamnitchi, Marinho Barcellos, Pascal Felber, and Etienne Riviere. 2015. Edge-Centric Computing: Vision and Challenges. *SIGCOMM Comput. Commun. Rev.* 45, 5 (Sept. 2015), 37–42.

[8] Jim Gettys and Kathleen Nichols. 2012. Bufferbloat: Dark Buffers in the Internet. *Commun. ACM* 55, 1 (2012), 57 – 65.

[9] Monia Ghobadi and Yashar Ganjali. 2013. TCP pacing in data center networks. In *Proceedings of the 21st IEEE Annual Symposium on High-Performance Interconnects, (HOTI)*. 25–32. https://doi.org/10.1109/HOTI.2013.18

[10] Manuel Gorius. 2012. *Adaptive Delay-constrained Internet Media Transport.* Ph.D. Dissertation. Saarland University.

[11] Keon Jang, Justine Sherry, Hitesh Ballani, and Toby Moncaster. 2015. Silo: Predictable Message Latency in the Cloud. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*. 435–448.

[12] S. Karthikeyini and S. Shankar. 2020. Cross Layer Aware Optimization of TCP Using Hybrid Omni and Directional Antenna Reliable for VANET. In *Inventive Computation Technologies*. Springer International Publishing, 530–546.

[13] Chengfa Li, Mao Ye, Guihai Chen, and Jie Wu. 2005. An energy-efficient unequal clustering mechanism for wireless sensor networks. In *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference*. 1–8.

[14] André Martins, Marcelo Ruaro, Anderson Santana, and Fernando G. Moraes. 2017. Runtime Energy Management under Real-time Constraints in MPSoCs. In *Proc. of the 2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. 4.

[15] D. Mills, J. Martin, J. Burbank, and W. Kasch. 2010. Network Time Protocol Version 4: Protocol and Algorithms Specification - IETF RFC 5905. (2010).

[16] S. Murugesan. 2008. Harnessing Green IT: Principles and Practices. *IT Professional* 10, 1 (2008), 24–33.

[17] Taiichi Ohno. 1988. *Toyota Production System: Beyond Large-scale Production.* CRC Press.

[18] Padmanabhan Pillai and Kang G. Shin. 2001. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *Proc. of the 18th ACM Symposium on Operating Systems Principles (SOSP)*. 89–102.

[19] Thomas Rausch, Cosmin Avasalcai, and Schahram Dustdar. 2018. Portable Energy-Aware Cluster-Based Edge Computers. In *Proceedings of the 2018 Symposium on Edge Computing (SEC'18)*. IEEE, 260–272.

[20] Ahmed Saeed, Nandita Dukkipati, Vytautas Valancius, Vinh The Lam, Carlo Contavalli, and Amin Vahdat. 2017. Carousel: Scalable Traffic Shaping at End Hosts. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 404–417. https://doi.org/10.1145/3098822.3098852

[21] Andreas Schmidt. 2020. *Cross-layer latency-aware and -predictable data communication.* Ph.D. Dissertation. Saarland University.

[22] Andreas Schmidt, Stefan Reif, Pablo Gil Pereira, Timo Hönig, Thorsten Herfet, and Wolfgang Schröder-Preikschat. 2019. Cross-Layer Pacing for Predictably Low Latency. In *Proc. of the 6th Intl. IEEE Workshop on Ultra-Low Latency in Wireless Networks (ULLWN)*. Paris, France, 6.

[23] Tajana Simunic, Luca Benini, Andrea Acquaviva, Peter Glynn, and Giovanni De Micheli. 2001. Dynamic Voltage Scaling and Power Management for Portable Systems. In *Proc. of the 38th Annual Design Automation Conference*. 524–529.

[24] Lin Wang, Fa Zhang, Kai Zheng, Athanasios V Vasilakos, Shaolei Ren, and Zhiyong Liu. 2014. Energy-efficient flow scheduling and routing with hard deadlines in data center networks. In *2014 IEEE 34th International Conference on Distributed Computing Systems*. IEEE, 248–257.

[25] David X Wei and Steven H Low. 2006. TCP Pacing Revisited. In *Proceedings of IEEE INFOCOM*. 1–11.

[26] Ben Zhang, Nitesh Mor, John Kolb, Douglas S. Chan, Ken Lutz, Eric Allman, John Wawrzynek, Edward Lee, and John Kubiatowicz. 2015. The Cloud is Not Enough: Saving IoT from the Cloud. In *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*.