Hermes: Enabling Energy-efficient IoT Networks with Generalized Deduplication

Christian Göttel
Department of Computer Science
University of Neuchâtel, Switzerland
christian.goettel@unine.ch

Lars Nielsen
DIGIT, Department of Engineering
Aarhus University, Denmark
lani@eng.au.dk

Niloofar Yazdani
DIGIT, Department of Engineering
Aarhus University, Denmark
n.yazdani@eng.au.dk

Pascal Felber
Department of Computer Science
University of Neuchâtel, Switzerland
pascal.felber@unine.ch

Daniel E. Lucani
DIGIT, Department of Engineering
Aarhus University, Denmark
daniel.lucani@eng.au.dk

Valerio Schiavoni
Department of Computer Science
University of Neuchâtel, Switzerland
valerio.schiavoni@unine.ch

ABSTRACT

The Internet of Things (IoT) is connecting a massive number of devices that generate a growing amount of data to be transmitted over the network. This traffic growth is expected to continue. Generalized deduplication (GD) is a novel technique to effectively compress the data to (a) reduce the data storage cost by identifying similar data chunks, and (b) reduce the pressure on the network infrastructure. This paper presents Hermes, an application-level protocol for the data-plane that can operate using GD as well as classic deduplication. Hermes significantly reduces the data transmission traffic while effectively decreasing the energy footprint, a key goal in many IoT deployments. We fully implemented Hermes, evaluated its performance using consumer-grade IoT devices (*e.g.*, Raspberry Pi 4B), and highlighted key tradeoffs to be considered to manage real-world workloads. Several fold to several order of magnitude gains over standard compressors and deduplication are achievable.

CCS CONCEPTS

• **Networks** → *Application layer protocols*; • **Mathematics of computing** → *Coding theory*; • **Computer systems organization** → Sensor networks.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
DEBS '20, July 13–17, 2020, Virtual Event, QC, Canada
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8028-7/20/07…$15.00
https://doi.org/10.1145/3401025.3404098

KEYWORDS

IoT, generalized deduplication, energy efficiency

ACM Reference Format:
Christian Göttel, Lars Nielsen, Niloofar Yazdani, Pascal Felber, Daniel E. Lucani, and Valerio Schiavoni. 2020. Hermes: Enabling Energy-efficient IoT Networks with Generalized Deduplication. In *The 14th ACM International Conference on Distributed and Event-based Systems (DEBS '20), July 13–17, 2020, Virtual Event, QC, Canada.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3401025.3404098

1 INTRODUCTION

The increasing adoption and expansion of Internet of Things (IoT) technologies is leading to a growing number of connected, low-energy Internet-enabled devices. Despite the imminent introduction of wider-band wireless technologies (*e.g.*, 5G and beyond), it is clear that the pressure on the network will continue to increase. Data compression [5] is an interesting solution given the compression potential of IoT-generated data. Figure 1 shows the compression gain (high is good) on ambient water and energy [1, 2].

Efficient compressors are usually too computationally intensive [3] and memory-eager for IoT devices. On the other hand, lightweight, memory-efficient approaches tend to have poorer compression performance [12]. Moreover, many IoT applications rely on *small data packets* and *compress data on a per packet basis* due to memory limitations, which limits the compression potential of traditional algorithms [11]. The compression gain in Figure 1 for LZW [10] and DEFLATE [3] decreases dramatically as packets become smaller.

Data deduplication (DD) is a known scheme to eliminate redundant data. Generalized deduplication (GD) [9] is a recently introduced scheme to reduce the cost of storage not only by finding equal data chunks, but also by finding similar

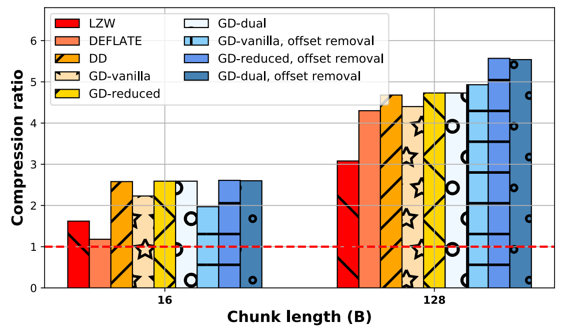

Figure 1: Compression ratio for real-world ambient water and energy data set (higher is better).

data chunks. The compression ratio of IoT data for DD and different schemes of GD [4] in Figure 1 shows that GD can not only outperform DD, but also LZW and DEFLATE for small packet sizes.

This paper introduces HERMES, a protocol and its implementation for data transmission reduction in IoT networks, especially suited for resource-limited nodes. HERMES' design is inspired by and expands on the schemes proposed in [11]. In HERMES, nodes share a common (and growing) data pool at the sink node, typically a Cloud- or Edge-based device. Nodes benefit from reduced traffic by contributing to the data pool without direct interactions between nodes. We implemented and experimentally evaluated HERMES' performance by micro- and macro-benchmarks on Raspberry Pi 4B. We show that GD can outperform DD, LZW and DEFLATE. A full version of this paper is available at [4].

2 BACKGROUND

Generalized deduplication (GD) [9] is a lossless data compression approach that eliminates equal and similar data chunks. This is achieved *without* comparing directly to previous chunks, but rather using a transformation function to systematically cluster similar data. GD splits data into a series of equal-sized smaller chunks and maps each chunk onto a basis-deviation pair by applying a transformation function. As transformation function, an error-correcting code (ECC) can be used. Each basis is saved exactly once and assigned a fingerprint, *e.g.*, using SHA-1 or CRC32. Rather than saving the chunk, GD stores a fingerprint-deviation pair. It should be noted that DD is a special case of GD with deviation zero.

GD for efficient data transmission. GD can also reduce data transmission in a lossless manner [11]. Source nodes apply GD by mapping chunks to a basis-deviation pair and send the basis' fingerprint and the deviation to the sink nodes. Sink nodes check for the fingerprint and send back an acknowledgement, if it is available. The basis and deviation can then be erased from source nodes' memory. If the basis is not available at a sink node, the sink node requests the missing basis from the source node. The source node sends the basis, and awaits an acknowledgement from the sink node to delete the basis and deviation.

Table 1: Message types for the HERMES protocol and relation with the node classes.

Message Type	basic	dedup.	gen. dedup.
Response	✓	✓	✓
Data	✓	⇥	⇥
Deduplication	✗	✓	✗
Deduplication data	✗	✓	✗
Gen. deduplication	✗	✗	✓
Gen. deduplication data	✗	✗	✓

Transformation function. Hamming codes [7] are linear ECCs that can be used for our mapping [8]. By applying the decoding function of the Hamming codes to the *codeword* (original chunk in our case) we obtain a *message* (a basis in our case). The deviation carries the information about the difference between the codeword and the *error-free codeword*. The latter is created by encoding the basis with the Hamming code. Hamming codes can correct one bit errors, which means that codewords are at most one bit away from the error-free codewords. The location of the bit is specified in a *syndrome vector* which represents the deviation in our case.

3 HERMES ARCHITECTURE

Assumptions: *(1)* all nodes use the same fingerprint length; *(2)* all GD nodes use the same transformation configuration; and *(3)* all deduplication nodes use the same chunk length.

A distributed deployment of HERMES uses *source*, *sink* and *intermediate* nodes. Sources inject data into the network, while sinks only ingest data without further retransmissions. An intermediate node is any node between a sink and source.

Nodes handle messages according to their class: *basic*, *DD* and *GD*. Basic nodes transmit messages without any processing, while DD and GD nodes perform DD and GD locally on the node, respectively. The node type determines the message type it handles as in Table 1. For example, a basic node acting as a source sends only raw data using the `Data` message type. Response messages for success, acknowledgement, or failures are sent to the previous node in the communication route.

A GD node can send two types of messages: *(1)* a `Gen. deduplication` message with the (basis fingerprint, deviation) pair, and *(2)* a `Gen. deduplication data` message, which sends the basis. DD nodes follow a similar structure but the `Deduplication` messages contain the chunk's fingerprint and the `Deduplication data` messages contain the original chunk. The `Data` message payload is processed using DD or GD according to the node's type. `Gen.`

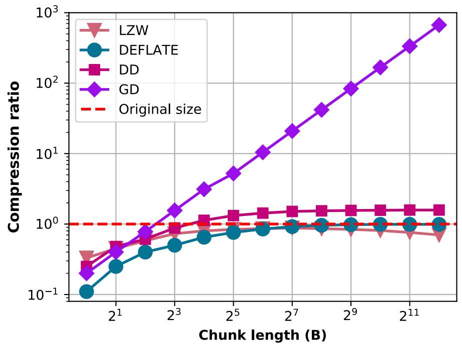

Figure 2: Compression ratio for different schemes

`deduplication data` or `Deduplication data` messages are transmitted when a receiving node responds with a `New fingerprint` message. This indicates the basis' fingerprint has not been seen before in the receiving node. Naturally, sink nodes maintain a growing record of previously seen fingerprints and their basis. Intermediate nodes can maintain a similar record, but limited to the fingerprints successfully communicated to a sink node.

4 EVALUATION

Testbed. Our experiments are deployed over a switched cluster of 16 Raspberry Pi 4B[1] featuring a Raspberry Pi PoE-HAT[2] to enable 802.3af Power-over-Ethernet [6] by an Ubiquiti Networks UniFi USW-48P-750 switch connected over Gigabit Ethernet. Raspberry Pis are used as source nodes and are connected to a Dell PowerEdge R330 server acting as sink node.

Each Raspberry Pi is running the beta bootloader 2019-12-03 and Raspian Buster Lite (2020-02-13) which are installed on a 32 GiB SanDisk Extreme microSDXC UHS-I card. The clocks of all machines are synchronized using NTP in order to relate the power consumption to the statistics of a benchmark run.

Power Measurements. The power measurements are gathered using two techniques: *(1) PowerSpy2.* The Alciom PowerSpy2[3] is a power analyzer connected between a power plug and a power adapter. We use the PowerSpy in real-time mode, where it streams periodic measurements with Bluetooth v2 at a frequency of 50 Hz. *(2) UniFi Switch.* The UniFi switch provides an access protected API reachable over a local `telnet` connection to query the PoE status of its ports. Using an ad-hoc `expect` script[4] we periodically gather PoE measurements at a frequency of about 8 Hz.

[1]https://www.raspberrypi.org/products/raspberry-pi-4-model-b/
[2]https://www.raspberrypi.org/products/poe-hat/
[3]https://www.alciom.com/en/our-trades/products/powerspy2/
[4]https://core.tcl-lang.org/expect/index

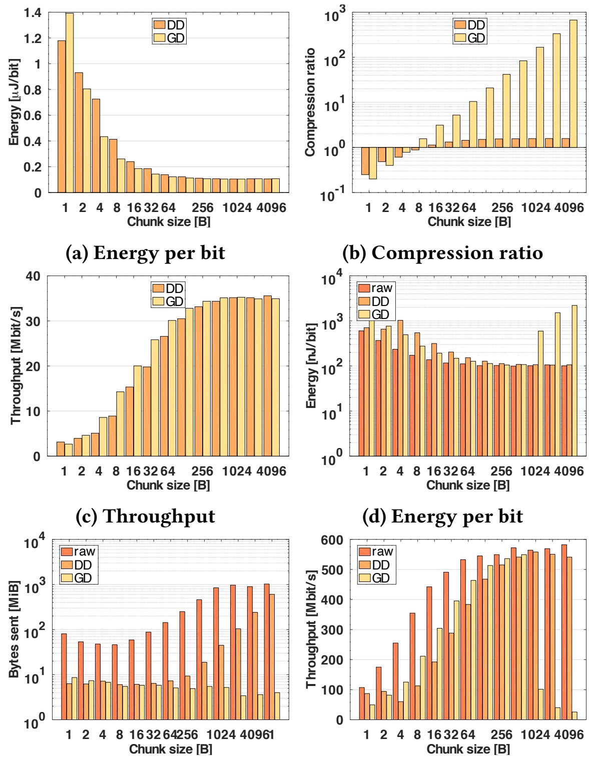

Figure 3: Micro- (a, b, c) and macro-benchmark (d, e, f) on Raspberry Pi 4B cluster.

Reduction of interferences. The Raspberry Pi is attached to the PowerSpy during the micro-benchmark. With an auxiliary machine we simultaneously monitor the Raspberry Pi over a serial channel (USB-to-UART) and record PowerSpy measurements. No other peripherals were attached to the Raspberry Pi except for Ethernet and the UART GPIO pins in order to keep the static power consumption of the Raspberry Pi as low as possible and to avoid power interference from other peripherals.

Synthetic dataset. We will show our best case scenario using synthetic data sets. We developed a parameterizable generator for a synthetic dataset to create a specific number of (unique) bases and to derive a specific number of chunks. A basis can be easily generated by selecting a random number considering the basis' length. The error-free codeword is obtained by encoding the basis. Chunks can then be derived by flipping a random bit.

Compression ratio. Figure 2 compares the compression ratio of our technique based on GD using CRC32 fingerprints with two standard compression algorithms, DEFLATE and LZW, considering the synthetic data sets. We apply all schemes on each chunk individually. As Figure 2 shows, neither DEFLATE nor LZW compress the data, only DD slightly compress data for larger chunk sizes. GD achieves significant compression ratios for chunk sizes beyond 8 B. Compression ratios of 334 and 668 for chunk lengths of 2048 and 4096 B can be achieved compared to 1.58 for DD.

Micro-benchmark. Our set of micro-benchmarks are shown in Figure 3 on the synthetic datasets. We show the different trade-offs in terms of chunk-size, energy per bit, compression ratio and throughput. Results show that energy per bit performance of GD is comparable to DD even considering the added transformation computation. For chunk sizes of 8 B and above, GD compresses the data by orders of magnitude better than DD due to the large number of chunks matched to each basis. Finally, the achieved throughput maxes out at 30 Mbit/s for a single thread. The throughput includes the following operations: reading the data from memory, applying the compression algorithm and looking for fingerprints in memory to check the availability of the fingerprint for DD and GD. In a real world scenario, a source node only needs to apply the compression algorithm and it is the sink node which looks for the fingerprint. Thus, we expect a higher throughput in real deployments.

Macro-benchmark. In the macro-benchmark, we evaluate raw data chunks, DD, and GD transmission methods as combined statistics for the Raspberry Pi cluster. Although our prototype implementation has multi-threading capability, we decided to use only two threads to be comparable to the micro-benchmark. We use one thread to handle the networking and a second thread to do the necessary transformations and look-ups. Figure 3d shows that for smaller chunks the energy per bit is higher. This is due to the inaccuracy of the measurement done with the UniFi switch. The UniFi switch has a much lower time resolution to update the PoE status statistics, which we observed to be around 4 to 5 seconds, thus lowering the accuracy and resolution of our measurement. With large chunk sizes (larger than typical IoT ones), the energy increases for GD. Most importantly, the network traffic generated for the different methods in Figure 3e is reduced significantly with GD. As seen in the micro-benchmark, the macro-benchmark also shows an optimal throughput between chunk sizes of 2 and 512 bytes. Approaching 1024 KiB, we recognize a significant drop in throughput for GD we assume is due to hardware limitations.

5 CONCLUSION

This work proposed and evaluated a new protocol (Hermes) for data compression across multiple sources using the emerging concept of generalized deduplication. For small data packets, our evaluations show that GD significantly outperforms standard compression approaches *e.g.*, LZW, and DEFLATE. Additionally, we demonstrated that Hermes under ideal conditions can provide orders of magnitude better compression than DD and even LZW/DEFLATE. We achieve this with a small added computational overhead as shown in deployments with Raspberry Pi model 4B. Future work will focus on developing transformations that are more data-aware and on carrying out large-scale deployments of Hermes with expanding its capability to packet loss correction.

ACKNOWLEDGEMENTS

This work was partially financed by the SCALE-IoT Project (Grant No. 7026-00042B) granted by the Independent Research Fund Denmark, by the Aarhus Universitets Forskningsfond (AUFF) Starting Grant Project AUFF- 2017-FLS-7-1, and Aarhus University's DIGIT Centre. The research leading to these results has also received funding from the European Union's Horizon 2020 research and innovation programme under the LEGaTO Project (legato-project.eu), grant agreement No 780681.

REFERENCES

[1] 2013. Indian Dataset for Ambient Water and Energy. http://iawe.github.io/ Accessed: 2020-02-03.
[2] Nipun Batra, Manoj Gulati, Amarjeet Singh, and Mani B Srivastava. 2013. It's Different: Insights into home energy consumption in India. In *ACM Workshop on Embedded Syst. for Energy-Efficient Build.* 1–8.
[3] L. Peter Deutsch. 1996. *DEFLATE Compressed Data Format Specification version 1.3.* RFC 1951. RFC Editor. http://www.rfc-editor.org/rfc/rfc1951.txt http://www.rfc-editor.org/rfc/rfc1951.txt.
[4] Christian Göttel, Lars Nielsen, Niloofar Yazdani, Pascal Felber, Daniel E. Lucani, and Valerio Schiavoni. 2020. Hermes: Enabling Energy-efficient IoT Networks with Generalized Deduplication. arXiv:2005.11158
[5] Abdallah Jarwan, Ayman Sabbah, and Mohamed Ibnkahla. 2019. Data transmission reduction schemes in WSNs for efficient IoT systems. *IEEE Journal on Selected Areas in Comm.* 37, 6 (2019), 1307–1324.
[6] Galit Mendelson. 2004. All you need to know about Power over Ethernet (PoE) and the IEEE 802.3 af Standard. *Internet Citation,[Online] Jun* (2004).
[7] Jorge Castiñeira Moreira and Patrick Guy Farrell. 2006. *Essentials of error-control coding.* John Wiley & Sons.
[8] Lars Nielsen, Rasmus Vestergaard, Niloofar Yazdani, Prasad Talasila, Daniel E Lucani, and Márton Sipos. 2019. Alexandria: A Proof-of-concept Implementation and Evaluation of Generalised Data Deduplication. In *IEEE Global Communications Conference (GLOBECOM).*
[9] Rasmus Vestergaard, Daniel E Lucani, and Qi Zhang. 2019. Generalized Deduplication: Lossless Compression for Large Amounts of Small IoT Data. In *European Wireless Conference.* IEEE.
[10] Terry A. Welch. 1984. A technique for high-performance data compression. *Computer* 6 (1984), 8–19.
[11] Niloofar Yazdani and Daniel E Lucani. 2019. Protocols to Reduce CPS Sensor Traffic using Smart Indexing and Edge Computing Support. In *IEEE GLOBECOM - Workshop on Edge Comp. for Cyber Phys. Sys.* IEEE.
[12] J. Ziv and A. Lempel. 1977. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory* 23, 3 (May 1977), 337–343. https://doi.org/10.1109/TIT.1977.1055714