THESIS FOR THE DEGREE OF LICENTIATE OF ENGINEERING

Parallel and Distributed Processing in the Context of Fog Computing
High Throughput Pattern Matching and Distributed Monitoring

CHARALAMPOS STYLIANOPoulos

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2018
Parallel and Distributed Processing in the Context of Fog Computing
High Throughput Pattern Matching and Distributed Monitoring
Charalampos Stylianopoulos

Copyright © Charalampos Stylianopoulos, 2018.

Technical report 179L
ISSN 1652-876X
Department of Computer Science and Engineering

Chalmers University of Technology
SE-412 96 GÖTEBORG, Sweden
Phone: +46 (0)31-772 10 00

Author e-mail: chasty@chalmers.se

Printed by Chalmers Reproservice
Göteborg, Sweden 2018
Parallel and Distributed Processing in the Context of Fog Computing
High Throughput Pattern Matching and Distributed Monitoring
Charalampos Stylianopoulos
Department of Computer Science and Engineering, Chalmers University of Technology

ABSTRACT

With the introduction of the Internet of Things (IoT), physical objects now have cyber counterparts that create and communicate data. Extracting valuable information from that data requires timely and accurate processing, which calls for more efficient, distributed approaches. In order to address this challenge, the fog computing approach has been suggested as an extension to cloud processing. Fog builds on the opportunity to distribute computation to a wider range of possible platforms: data processing can happen at high-end servers in the cloud, at intermediate nodes where the data is aggregated, as well as at the resource-constrained devices that produce the data in the first place.

In this work, we focus on efficient utilization of the diverse hardware resources found in the fog and identify and address challenges in computation and communication. To this end, we target two applications that are representative examples of the processing involved across a wide spectrum of computing platforms. First, we address the need for high throughput processing of the increasing network traffic produced by IoT networks. Specifically, we target the processing involved in security applications and develop a new, data parallel algorithm for pattern matching at high rates. We target the vectorization capabilities found in modern, high-end architectures and show how cache locality and data parallelism can achieve up to three times higher processing throughput than the state of the art. Second, we focus on the processing involved close to the sources of data. We target the problem of continuously monitoring sensor streams —a basic building block for many IoT applications. We show how distributed and communication-efficient monitoring algorithms can fit in real IoT devices and give insights of their behavior in conjunction with the underlying network stack.

Keywords: fog computing, resource-constrained devices, high throughput, pattern matching, vectorization, distributed processing, distributed monitoring
Preface

Parts of the contributions presented in this thesis have led to the following manuscripts.

▷ Charalampos Stylianopoulos, Magnus Almgren, Olaf Landsiedel, Marina Papatriantafilou,
  “Multiple Pattern Matching for Network Security Applications: Acceleration through Vectorization”,
  appeared in the 46th International Conference on Parallel Processing (ICPP), Bristol, United Kingdom, 14-17 August, 2017

▷ Charalampos Stylianopoulos, Magnus Almgren, Olaf Landsiedel, Marina Papatriantafilou,
  “Geometric Monitoring in Action: a Systems Perspective for the Internet of Things”,
  under submission, 2018
Acknowledgments

First, I would like to express my gratitude to my supervisors Magnus Almgren, Olaf Landsiedel and Marina Papatriantafilou. I feel privileged to have the continuous support and guidance of three excellent mentors. Without them, this thesis would not be possible.

I am also grateful to my past and current colleagues in the Division of Networks and Systems, for making these first years of my PhD studies a fun and rewarding experience. Many thanks to Adones, Ali, Aljoscha, Amir, Aras, Bapi, Bastian, Bei, Beshr, Boel, Carlo, Christos, Dimitris, Elad, Elena, Fazeleh, Georgia, Hannah, Yiannis N., Iosif, Ivan, Joris, Karl, Katerina, Nasser, Oliver, Paul, Philippas, Romaric, Thomas R., Thomas P., Tomas, Valentin T., Valentin P., Vincenzo, Vladimir and Wissam. Also, thanks to my collaborators Oskar, Linus and Simon.

Special thanks go to my friends in Göteborg that made it a lovely place to live in. Thank you Petros, Vaggelis, Stavros, Yiannis S., Angelos, Chloe, Maria, Vasiliki, Manos and Suvi for enduring my endless talking. Also, many thanks to old friends, especially Nikos, Kostas, Efi, Elias and Dinos, for all the great times we had and will keep having.

My thanks go to my family, my parents and my siblings for their unconditional love and trust in me. I owe it all to them. Finally, I want to thank Kelly for being there for me and for all the moments we share together.

Charalampos Stylianopoulos
Göteborg, June 2018
I INTRODUCTION

1 Introduction

1.1 Challenges in processing IoT data: the need for scalability on a range of new architectures

1.2 Background

1.2.1 Advances in processing hardware

1.2.2 Wireless sensors networks

1.3 Scope of the thesis

1.4 Representative problems in the context of fog computing

1.4.1 Data processing in the context of cyber-security

1.4.2 Distributed monitoring of sensor readings

1.5 Research questions and contributions

1.5.1 Research questions

1.5.2 (Paper I) Multiple Pattern Matching for Network Security Applications: Acceleration through Vectorization
## CONTENTS

1.5.3 (Paper II) Geometric Monitoring in Action: a Systems Perspective for the Internet of Things ........................................ 14

1.6 Conclusions and Future Directions ........................................ 16

Bibliography ........................................................................... 17

### II PAPERS

#### 2 PAPER I

2.1 Introduction ........................................................................ 28

2.2 Background ........................................................................ 31

2.2.1 Traditional Approach to Multiple-Pattern Matching .......... 31

2.2.2 Filtering Approaches and Cache Locality ......................... 31

2.2.3 Vectorization ................................................................. 32

2.3 System Model .................................................................... 33

2.4 Algorithmic Design ............................................................. 34

2.4.1 S-PATCH: a vectorizable version of DFC ......................... 34

2.4.2 V-PATCH: Vectorized algorithmic design ......................... 38

2.5 Evaluation ......................................................................... 42

2.5.1 Experimental setup ......................................................... 42

2.5.2 Overall Throughput ....................................................... 44

2.5.3 The effects of the number of patterns ............................... 46

2.5.4 Filtering Parallelism ....................................................... 48

2.5.5 Changing the vector length: Results from Xeon-Phi ........ 48

2.6 Related Work .................................................................... 50

2.6.1 Pattern matching algorithms ........................................... 50

2.6.2 SIMD approaches to pattern matching ............................. 51

2.6.3 Other architectures ....................................................... 52

2.7 Conclusion ......................................................................... 52

Bibliography ........................................................................... 53

#### 3 PAPER II

3.1 Introduction ........................................................................ 60
<table>
<thead>
<tr>
<th>Sections</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2 Background</td>
<td>63</td>
</tr>
<tr>
<td>3.2.1 The Geometric Monitoring Method (GM)</td>
<td>63</td>
</tr>
<tr>
<td>3.2.2 In the context of wireless sensor networks</td>
<td>65</td>
</tr>
<tr>
<td>3.3 Practical GM-based threshold-monitoring: design aspects and</td>
<td>66</td>
</tr>
<tr>
<td>algorithmic implementation</td>
<td></td>
</tr>
<tr>
<td>3.3.1 Addressing system challenges: processing, communication</td>
<td>66</td>
</tr>
<tr>
<td>3.3.2 Tunable system-parameters</td>
<td>69</td>
</tr>
<tr>
<td>3.4 Experimental methodology</td>
<td>69</td>
</tr>
<tr>
<td>3.5 Evaluation from a holistic system perspective</td>
<td>72</td>
</tr>
<tr>
<td>3.5.1 Full-system simulations</td>
<td>72</td>
</tr>
<tr>
<td>3.5.2 Validation through Testbed Experiments</td>
<td>75</td>
</tr>
<tr>
<td>3.5.3 Runtime insights: a closer look</td>
<td>76</td>
</tr>
<tr>
<td>3.5.4 Accuracy/Responsiveness: The effect of packet losses</td>
<td>79</td>
</tr>
<tr>
<td>3.6 Related Work</td>
<td>82</td>
</tr>
<tr>
<td>3.7 Conclusions and Future Work</td>
<td>83</td>
</tr>
<tr>
<td>Bibliography</td>
<td>84</td>
</tr>
</tbody>
</table>
List of Figures

1.1 An example fog computing architecture that includes a cloud layer with high-end servers, an intermediate layer close to the gateways and the IoT layer with resource-constrained devices. The work in this thesis targets representative problems on Area A (centralized, high throughput processing) and Area B (distributed processing and communication). ........................................... 9

2.1 Filter Design of S-PATCH. HT stands for the Hash Tables that contain the full patterns. ................................................................. 36

2.2 Input Transformation from consecutive characters to sliding windows of two characters. ................................................................. 40

2.3 Figure describing the filter merging optimization. In the upper half, lookups on two filters require two gather invocations. Once the filters are merged in memory in the lower half, one gather brings information from both filters to the registers. . . . . 41

2.4 Performance comparison between the different algorithms for public and random data sets on the Xeon platform. . . . . . . 45
2.5 a) comparison between the scalar and vectorized versions of our approach, as the number of patterns increases. b) filtering-to-verification ratio (left axis), as well as the average number of useful elements in the vector registers after filter 2 (right axis), as the number of patterns increases. c) comparison between the scalar and vectorized approach, as the fraction of matches in the input increases.  46

2.6 Measuring the performance of the filtering part only. “V-PATCH-filtering+stores” includes the cost of storing the results of the filtering phase to temporary arrays.  49

2.7 Performance comparison between the different algorithms for public and random data sets on the Xeon-Phi platform.  50

3.1 An example illustrating the GM method.  65

3.2 Full system simulations: The duty cycle, broken down to sending and listening, as well as the cost of idle listening.  74

3.3 Runtime insights from the execution of GM over a period of 20 hours.  77

3.4 Runtime insights: Percentage of epochs with concurrent updates (regular on the left part and triggered on the right part).  78

3.5 Accuracy/Responsiveness of the algorithm measured as we introduce packet losses.  80
Part I

INTRODUCTION
Introduction

The last decade has been marked with many disruptive technologies where the Internet of Things (IoT) is one of the most well known. An increasing number of objects are now becoming “smart” by getting sensing and networking capabilities. The number of devices that are expected to be connected to the internet in the coming years is impressive [1], including everyday objects, cars, as well as devices that are part of the electricity grid and the industry [2]. However, what is even more impressive is the amount of data that they will produce. The upcoming challenge associated with all that data is now this: how, when and where to process the high volumes of data, in order to extract value [3] [4]?
1.1 Challenges in processing IoT data: the need for scalability on a range of new architectures

The usefulness of IoT computing comes from the fact that small devices and sensors are able to continuously provide readings about their state and the state of their environment. That data then needs to be processed to provide valuable information about the system, and often to create control commands that are fed back to the devices. IoT devices are typically resource-constrained nodes, sometimes equipped with only enough computational power to acquire readings and send them onwards. In order to accommodate the heavy processing required for many IoT applications, IoT networks rely on a connection with the cloud, where the hardware resources are abundant. This computing model, however, has scalability problems and fails to meet the requirements of many IoT applications. Primarily, the vast number of connected devices, in conjunction with the increasing volume of data generated by each device, means that it is impossible to send all this data to the cloud without exhausting the available bandwidth. Moreover, IoT applications often have tight latency requirements, which means that aggregating data to the cloud, processing it and sending back the control commands adds an unacceptably high overhead.

As a way to address these limitations, fog computing$^1$ has been proposed as an alternative extension to cloud computing described above [5]. The core idea of fog computing is to move the processing to where it is most needed, closer to the data origin. In this paradigm, the processing and control logic that would typically be found on cloud servers is now pushed down to intermediate nodes, closer to the sources of data. Base stations and gateways will thus be enhanced with processing capabilities and storage, enough to quickly act on aggregated data coming from IoT networks. Moreover, the networks themselves will take over some of the processing and control logic, so that a significant portion of data do not have to be forwarded upwards. It is estimated that this way, more

---

$^1$The term fog computing is often used interchangeably with edge computing, with the latter focusing more on processing at the devices that produce the data. We will use the term fog computing from now on, to emphasize the need for distributed processing.
than 40 percent of IoT data [6] will be processed on devices physically close to
the data sources, by 2019. Hence, the up-link network bandwidth will no longer
a limiting factor and there is opportunity to significantly reduce the overall pro-
cessing latency.

The introduction of fog computing opens new, interesting research ques-
tions in several ways. On one hand, fog computing comes with a new set of
challenges, that mainly revolve around the problems of (i) how to distribute
computational tasks on the layers of fog [7], (ii) how to move them there, (iii) in what ways the different components of fog interact and connect with
each other [3] and (iv) how to maintain Quality of Service [8]. On the other
hand—and this is important in the context of this thesis—fog computing brings
together processing applications with different requirements that target different
platforms, under the same computing approach. Processing methods originally
designed for servers in the cloud now also become relevant at the intermediate
layer of fog and must adapt to the hardware found there. At the same time,
computational tasks designed to operate on aggregated data on a single node can
benefit if the processing logic is made distributed and handled close to, or even
by the nodes that produce that data. This increases the design space of existing
solutions and poses interesting research questions along the whole spectrum of
processing involved in fog computing.

1.2 Background

This section outlines background topics that are relevant in the context of this
thesis. We start by discussing relevant advances in hardware, followed by back-
ground information on wireless sensor communication.

1.2.1 Advances in processing hardware

Commodity hardware is constantly changing and evolving. Every new gener-
ation of platforms is enriched with new hardware features, designed to better
serve the needs of applications. The introduction of such features enables new
techniques that applications can use to gain performance.

A characteristic example of such important techniques is vectorization. It utilizes processing units that operate on a vector of elements simultaneously, instead of separate elements at a time, in a Single Instruction Multiple Data (SIMD) manner [9, 10]. SIMD vectorization is traditionally used in computationally intensive, number-crunching applications, where computation is performed on independent data, sequentially stored in memory.

Vectorization has been available on commodity hardware for many years. Recent advances in hardware platforms have made it relevant again for the following reasons: (i) new vector instruction sets have introduced the *gather instruction* [11] that allows accessing data from non-contiguous memory locations and (ii) modern processor designs are shifting towards new architectures with more emphasis on vectorization. As an example Intel’s Xeon Phi [12] supports 512 bit vector registers. On those platforms, vectorization is not just an option but a must, in order to achieve high performance [13].

Along with the introduction of new hardware techniques, existing features become more widespread and are adopted by a wider range of platforms. As an example, even embedded devices [14] are now massively parallel and support programmable Graphics Processor Units (GPUs), allowing some of the techniques found on high-end servers to be used on those devices as well.

### 1.2.2 Wireless sensors networks

Considering the lower layer of fog computing, the hardware found there has substantially different characteristics compared to the high-end servers described in the previous section. Typical devices at this layer are small, battery powered sensors. The main hardware components on these nodes are usually: (i) a set of simple sensors that periodically collect data from the environment (ii) a resource constrained microcontroller unit (MCU) for simple data and packet processing, and (iii) a radio transceiver for communication with other nodes that is used to form networks (either structured or mesh).

In wireless sensors networks (WSN), the battery lifetime is the most valu-
1.3. SCOPE OF THE THESIS

able resource. Thus the design of hardware and software for wireless sensors emphasizes on minimizing the energy footprint. As a consequence, applications on wireless sensors need to deal with the fact that processing components are very simple and resource constrained. In addition to computation, communication in WSN is also expensive. In fact, the radio is the most energy-hungry component, often consuming up to 10 times more energy than the MCU. For this reason, the goal of most communication protocols is radio duty cycling (RDC), where the radio is kept off as much as possible and is turned on for only a small fraction of the total time. A simple and commonly used RDC policy is to turn the radio on a fixed number of times per second, called the channel check rate (CCR) [15]. A node that wants to transmit, will keep transmitting for a duration of at least $1/\text{CCR}$ seconds to ensure that all neighbouring nodes had a chance to turn their radio on and receive. The channel check rate is a tunable parameter of the protocol that directly affects the battery lifetime of the nodes.

In addition to energy reservation, latency and reliability are other important considerations in WSNs. Sensors usually form multi-hop networks over unreliable and lossy links that are constantly subject to interference. For this reason, a large body of WSN research has focused on how to design reliable and low-latency network protocols, with interesting new advances over the last few years [16–18].

1.3 Scope of the thesis

This thesis covers aspects of processing and communication in fog, through representative applications that pose challenges stemming from two key problems: (i) hardware diversity and (ii) data diversity.

Hardware diversity is important since processing can happen on a wide and heterogeneous range of platforms, from resource-constrained embedded devices to high-end servers. On each of these platforms, understanding and making efficient use of the new architectures’ features enable new possibilities. Part of the work in this thesis focuses on how to make efficient use of new
architectures and features found on these platforms to enable new processing methods.

Data diversity ranges from high-rate data aggregated to a single point, to lower-rate data from many sources, where processing needs to happen in a distributed manner. Applications that operate on accumulated data usually require high throughput processing rates, while distributed applications require minimal, energy efficient communication and distributed control.

We use two applications that instantiate the problems just described and highlight challenges and solutions. Those applications span across the two ends of the spectrum of fog computing: processing for security applications close to the cloud and distributed monitoring of sensor readings inside IoT networks. These applications target different hardware and address different challenges and they become relevant in the context of fog computing, as part of the same processing pipeline. We return to these applications in the next section.

Figure 1.1 shows a conceptual representation of a fog computing architecture. The two problem domains that this thesis focuses on are shown in this figure.

1.4 Representative problems in the context of fog computing

This section introduces the two problem domains that are used as a basis for the work in this thesis. We summarize the challenges in this section and elaborate further on the contributions in each domain in Section 1.5.

1.4.1 Data processing in the context of cyber-security

In the first problem domain, we consider the problems and challenges involved in pattern matching, with a focus on its application for Network Intrusion Detection Systems (NIDS).

Motivation. NIDSs are typically found at the entry point of networks and their purpose is to analyze the incoming and outgoing network traffic to detect
1.4. REPRESENTATIVE PROBLEM IN FOG COMPUTING

Figure 1.1: An example fog computing architecture that includes a cloud layer with high-end servers, an intermediate layer close to the gateways and the IoT layer with resource-constrained devices. The work in this thesis targets representative problems on Area A (centralized, high throughput processing) and Area B (distributed processing and communication).

any malicious behavior, ranging from unauthorized access, malware that exploit software vulnerabilities, data exfiltration etc. They typically employ sophisticated analysis that considers not only the packet headers but also the contents of each packet (deep packet inspection [19]). There are many available NIDSs, with Snort [20] and Bro [21] being some of the most popular and mature in the open source community.

Network Intrusion Detection Systems gain new significance in the context of fog computing. IoT networks are connected to the Internet, sending sensor readings upwards towards the cloud and receiving back control traffic. The end-devices producing sensitive data are potential attack targets. However, since they are typically resource-constrained, traditional security mechanisms cannot be employed on them. Hence, it is important to add protection mechanisms, both at the entry point of the network and along the data path towards the cloud.

**Challenges.** An essential building block of many such systems is *pattern matching*, i.e., to discover if any of many predefined patterns exist in an input stream (*multiple* pattern matching), for whitelisting or blacklisting. ² Consider-

²Apart from its role in intrusion detection, pattern matching is also a core function in many other
erating the processing involved in NIDSs, pattern matching is the most computationally intensive part and represents a major performance bottleneck. More than 70% of the running time of a NIDS is spent on pattern matching [25, 26]. This fact, in conjunction with the ever increasing rates of traffic that needs to be processed, pushes the performance of NIDSs to their limits. Achieving high pattern matching throughput is challenging yet crucial for these systems: if the processing throughput cannot match the incoming traffic rate, the system will have to start dropping packets and maybe miss significant attacks.

**Related work.** Pattern matching has been an active field of research for many years and there are many proposed approaches. The algorithm proposed by Aho and Corasick [27] is one of the most well known and the one currently used by Snort. The first step of Aho-Corasick is to create a finite-state automaton from the malicious patterns. Then, the algorithm scans the input traffic byte-by-byte to traverse the automaton, until it arrives at a final state that indicates the detection of an attack. Even though Aho-Corasick performs only a small number of operations per byte, it fails to perform well in practice, due to poor cache locality.

State of the art approaches have been proposed to address the limitations of Aho-Corasick. A family of algorithms in the literature replace the state machine of Aho-Corasick with filters. Choi et al. [22] use a series of succinct filters, created using a small part of each malicious pattern. In this way, most of the benign input traffic is quickly filtered out, using cache-resident data structures. The part of the input that matches the information in the filters is further examined in a later verification phase that involves lookups in hash tables that contain the actual patterns. Similarly, Moraru et al. [28] use a modification of Bloom Filters [29] to scan both the input and the subset of patterns that are relevant.

**Open problems.** Even though the state of the art approaches have substantially increased the achieved throughput, they perform sub-optimally in modern architectures, because they fail to make use of the new characteristics and features available. As an example, most pattern matching algorithms do not make tasks, such as virus detection [22], text search [23] and genome analysis [24].
use of the vector execution units and leave them underutilized. In this thesis we present techniques that allow us to make the most out of the available hardware and achieve considerable speedups in throughput. A summary of this work follows in Section 1.5.2.

1.4.2 Distributed monitoring of sensor readings

In the second problem domain, considering the computation involved in resource-constrained IoT networks, we focus on the important problem of distributed monitoring of sensor readings.

Motivation. We address the issue of continuously monitoring a distributed set of sensor values and keeping track of a function of interest, defined over the network-wide aggregate of these values. Often, the goal is to always be able to detect whether the value of the monitored function has exceeded a pre-defined threshold. Keeping track of such a function is a basic building block for many IoT applications and control loops, e.g. for detecting outliers [30], hot-spots [31] or denial-of-service attacks [32].

Monitoring sensor values is a prime example of the applications that fog computing is designed for. The need for timely monitoring and low latency detection of a threshold violation calls for local processing, close to the sources of data. Ideally, the monitoring logic can even be placed inside the IoT network and distributed to the sensor nodes themselves.

Challenges. Keeping track of a function defined over a network-wide aggregate is a challenging task in practice. A simple solution is to aggregate every reading from every node in the network to a central entity and compute the aggregate there. Such an approach is impractical in networks with battery-constrained devices: using the radio for transmission or reception is the single most expensive operation in terms of energy [33]. Thus, the challenge associated with this problem is to reduce the number of sensor readings that need to be transmitted, by letting all nodes locally determine whether a reading should be transmitted. However, finding such local criteria is challenging when the function to monitor is non-linear (e.g. the variance of the readings), yet it is
non-linear functions that are particularly interesting for many real-world applications (e.g. detecting a denial-of-service attack).

**Related work.** Sharfman et al. [34] proposed a general method called *geometric monitoring (GM)* that can monitor any function (linear or not) defined over the average of network readings and keep track of its value with respect to a threshold. Every node in this algorithm is capable of deriving constraints on its local values and avoid communication as long as those constraints are not violated. The GM method has been extended with sketches [35] and prediction models [36] and has been applied to outlier detection [30] and data stream queries [37].

**Open problems.** Apart from the existing general analysis of algorithms, such as the one described above, the applicability to a real IoT deployment is still unclear, from a practical perspective. Up until now, there are no insights in how the system aspects of IoT networks interact with such algorithms. Specifically, the underlying network stack can have a significant impact on the efficiency of the algorithms, in terms of energy consumption on the nodes, as well as latency and reliability of communication. Moreover, the resource constrained nature of the sensor nodes makes the processing required by the algorithm challenging in practice.

In this thesis we take a step beyond the existing analysis and consider, not just the algorithm in isolation, but also the interplay with system aspects, such as the network stack. A summary of this works follows in Section 1.5.3.

1.5 **Research questions and contributions**

Based on the challenges discussed above, in this section, we introduce general research questions that have driven the work in this thesis, as well as a summary of the papers included in this thesis that contribute to these questions.
1.5. RESEARCH QUESTIONS AND CONTRIBUTIONS

1.5.1 Research questions

This thesis addresses the following research questions that emerge in the context of fog computing:

- **RQ1**: How can new hardware support be used to improve the processing throughput of data, across a wide spectrum of platforms.

- **RQ2**: How can distributed algorithms be used to push computation closer to the sources of data, in order to utilize data locality and make efficient use of the limited resources found there?

- **RQ3**: How do system aspects of the different layers of fog computing influence the design of efficient algorithmic approaches?

*RQ1* and *RQ3* become particularly relevant when considering applications that require high throughput processing of large volumes of data, such as pattern matching. The variety of features found in modern platforms (e.g. advanced vector instructions) offers new possibilities for faster processing, but requires novel, hardware-aware algorithmic designs that make efficient use of those features.

*RQ2* and *RQ3* have an important role in the context of fog computing, where distributing computation is an important way to ensure scalability, in terms of bandwidth and latency. However, turning a centralized algorithm into a distributed one is challenging, especially when it is applied in a resource constrained setting (e.g. in wireless sensor networks).

We relate back to these research questions and how we address them in this thesis, in the context of the following research contributions.

1.5.2 (Paper I) Multiple Pattern Matching for Network Security Applications: Acceleration through Vectorization

In this paper, we introduce V-PATCH, a data-parallel algorithm for pattern matching, that uses vector instructions to process multiple bytes of input, in
parallel. This work builds on the observation that recent, state of the art approaches for pattern matching that rely on quick filtering of the input, have brought the problem close to the processor and achieve good cache locality. As a result, long memory latencies are no longer the dominant bottleneck and the computational part of pattern matching becomes significant. With that in mind, we target that computational part and show how to improve it further, through vectorization (see Section 1.2.1). In this way, we contribute towards \( RQ1 \) and show how to make efficient use of the available hardware features.

We follow a two-step approach. First, we propose a refined and extended filtering strategy that: (i) performs filtering based on cache-resident data structures and is effective for the types of patterns found in real traffic, and (ii) is simple enough to allow efficient vectorization. As an example, we deal separately with small, but frequently found patterns and perform more targeted filtering for longer patterns. Second, we design a vectorized version that uses specialized instructions to parallelize the computation performed on the filters, together with optimizations (e.g. filter merging) that allow us to make the most of the filtering design.

We evaluate the effectiveness of V-PATCH using real malicious patterns from Snort [38], against both real and synthetic traffic mixes. The results on two platforms, an Intel Haswell processor and an Intel Xeon Phi co-processor, show up to 1.8x and 3.6x times faster processing throughput respectively, against the state of the art. Furthermore, we find that the vectorized approach retains a stable speedup of 1.4x over the scalar one, as the number of malicious patterns increases.

1.5.3 (Paper II) Geometric Monitoring in Action: a Systems Perspective for the Internet of Things

In this paper, we study geometric monitoring from a full-system perspective, when applied on real IoT networks. We propose a system design for geometric monitoring on top of a wireless sensor network stack. Then, we thoroughly evaluate the performance benefits achieved in practice, the run-time behavior of
the algorithm and the effects of packet losses. Through this work, we contribute towards \( RQ2 \) and \( RQ3 \), by showing that distributing computation to resource-constrained devices can have significant benefits, as well as by evaluating the effect of the system aspects in the performance of the algorithm.

We design the system on top of multi-hop mesh networks, without the need for maintaining a topology. When a node detects a violation, it will trigger a network-wide broadcast and inform every other node of its new value. This is done by network flooding, where a node that receives “new” information will broadcast it further to its neighbors. In the event that a node fails to receive an update by any of its neighbors, that node will be \textit{out of sync} until a subsequent broadcast from the same originator arrives.

Important considerations that are taken into account are the following. First, upon every new measurement or update, nodes need to check their local violation criteria, which typically involves finding whether two curves intersect. This can be computationally challenging for resource-constrained devices and in some cases, it might be impractical to compute it accurately and in time. To this end, we propose a simple relaxation of the violation check that introduces a trade-off between computational efficiency and communication reduction. Second, we investigate the important parameters of the network stack that affect the effectiveness of the algorithm in practice. As we show in the paper, the rate at which nodes wake-up to receive traffic (Channel Check Rate, CCR) greatly affects the energy savings of the GM method.

We evaluate our design using both full-system simulations and real IoT testbeds. Overall, we find that GM brings significant benefits to monitoring tasks, in terms of communication reduction. Specifically, when monitoring the variance and the average of real temperature data, GM achieves 3x and 11x reduction in duty-cycle, respectively. However, those benefits are limited compared to the communication reduction of the algorithm in isolation (4.3x and 44x respectively), due to baseline energy overhead of the network stack. Closer looks into the run-time behavior of the algorithm show that (i) the communication pattern varies greatly, and (ii) packet losses greatly impact the amount of time a node is \textit{out of sync} and reduce the ability of the algorithm to detect
1.6 Conclusions and Future Directions

Motivated by the emergence of IoT and fog computing, this thesis targets the challenges of processing involved across a range of platforms, by focusing on two representative problems, namely pattern matching and continuous monitoring of sensor readings.

On the problem of pattern matching for Network Intrusion Detection, we focus on how to make efficient use of newly introduced hardware features to improve the processing throughput, thus contributing towards RQ1 (cf. Section 1.5.1). The work in this thesis shows that, using advanced vector instructions, it is possible to improve the pattern matching throughput, across a range of data sets and different platforms.

On the problem of continuously monitoring distributed sensor readings, the work in thesis contributes insights towards RQ2. We demonstrate how geometric monitoring can be used to share the processing logic across the resource-constrained devices of IoT networks and give insights from real deployments. Through this work, important aspects of RQ3 are also made clear and we find that a) algorithmic engineering is required to adapt to the needs of battery and CPU-constrained devices and we suggest appropriate approximations, and b) the underlying communication stack greatly influences the performance of the algorithm, in terms of energy, latency and accuracy.

The results presented above target challenges and techniques on two separate sides of the fog computing spectrum. The next challenge is how to bring them closer together and extend them across the different layers of fog.

In the context of pattern matching, it is interesting to consider approaches that would better fit in the intermediate layer, for the following reasons. First, the gateways found at the intermediate layer are points of entry into the IoT network. For this reason, fast intrusion detection is required to secure both the gateways themselves as well as the network to which they provide access. Second, performing intrusion detection closer to the sources of data and not
on remote servers, provides opportunities for bandwidth reservation as well as better response times. Finally, the hardware found at the intermediate layer is becoming increasingly powerful and offers new capabilities. As an example, embedded RaspberryPi-like devices come with multiple cores and even programmable GPUs [14], so they can take the role of gateways with support for processing. Hence, it is interesting to see how to make efficient use of new platforms and the hardware resources they have to offer.

Based on insights from the work in distributed monitoring, described above, there are two interesting aspects that are worth further investigation. First, distributed monitoring algorithms can be extended to take into account the reality of IoT communication. As an example, would it be possible to rely less on a network-wide broadcast mechanism and, instead, resolve violations in a local neighborhood? This would bring overall benefits for both the energy savings and the accuracy of the algorithm. In this direction, we are currently looking at ways to extend the existing literature [31] with a distributed algorithm that is less demanding in terms of communication and is not oblivious to the underlying network stack. Second, it is interesting to see how to design a tailored protocol that serves the communication needs of such distributed applications. Recently, advances in wireless sensor protocols (e.g. synchronous transmissions [16] and channel hopping [39]) have made it possible to achieve fast and highly reliable flooding in mesh networks. One can thus investigate to what extent these protocols can serve the needs of distributed monitoring and how they can be extended further.

**Bibliography**


