Pigasus 2.0: Making the Pigasus IDS Robust to Attacks and Different Workloads

Zhipeng Zhao,* Nirav Atre,‡ Hugo Sadok,‡ Siddharth Sahay,‡
Shashank Obla,‡ James C. Hoe,‡ Justine Sherry‡
‡ Carnegie Mellon University * Microsoft

ACM Reference Format:
Zhipeng Zhao, Nirav Atre, Hugo Sadok, Siddharth Sahay, Shashank Obla, James C. Hoe, Justine Sherry. 2022. Pigasus 2.0: Making the Pigasus IDS Robust to Attacks and Different Workloads. In *ACM SIGCOMM 2022 Conference (SIGCOMM '22 Demos and Posters), August 22–26, 2022, Amsterdam, Netherlands.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3546037.3546065

1 Introduction

Intrusion Detection and Prevention Systems (IDS/IPSes) are critical components of the service chain for many network deployments. Ever-increasing network line rates and security threats have imposed substantial performance and correctness requirements on these systems: 100Gbps+ throughput with 100K+ concurrent connections, while scanning for 10K+ attack signatures in every packet.

Unlike traditional software-based IDS/IPSes (*e.g.*, Snort [2]) which required hundreds of CPU cores to scale to 100 Gbps, the Pigasus IDS/IPS [6] was the first to demonstrate that it is possible to achieve these goals within the footprint of a single, FPGA SmartNIC-equipped server. Pigasus performs this feat using a novel, "FPGA-first" approach to IDS/IPS, where the vast majority of packet processing (*e.g.*, packet parsing, TCP reassembly, multi-string pattern matching) is done using a highly-parallel datapath aboard the FPGA. The FPGA pipeline is able to efficiently filter out input traffic that is provably innocent, only forwarding 'suspicious' packets – a small fraction of the overall input – to the CPU for more expensive, final-stage processing. Overall, this saves significant compute, power, and cost compared to a software-based IDS.

However, despite its impressive performance, the original version of Pigasus ('Pigasus 1.0') suffers from three shortcomings that hinder its deployment potential. First, Pigasus 1.0 espouses a one-size-fits-all processing pipeline, an approach that may not be sufficiently general for real-world deployments. Second, owing to its static datapath, any divergence from common-case behavior during run-time (*e.g.*, due to bursty traffic patterns, or transient oversubscription of one or more modules) may result in significant performance drops. Finally, the same design decisions that optimize Pigasus 1.0 for the common-case also leave the system vulnerable to Algorithmic Complexity Attacks (ACAs) [1, 3], a potent class of Denial-of-Service (DoS) attacks targeting compute-intensive network functions like Pigasus.

In this work, we present Pigasus 2.0, a complete re-architecture of the Pigasus framework that enables *compile-time* configuration of the processing pipeline to cater to different deployment environments, *run-time* provisioning of CPU cores to handle FPGA spillover during periods of oversubscription, and *robustness* against ACAs targeting the Pigasus datapath without sacrificing common-case performance. Compared to Pigasus 1.0, Pigasus 2.0 achieves, on average, 38% higher throughput during normal operation, and suffers 90-99% lower loss in goodput due to ACAs.

Permission to make digital or hard copies of part or all 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 third-party components of this work must be honored. For all other uses, contact the owner/author(s).
SIGCOMM '22 Demos and Posters, August 22–26, 2022, Amsterdam, Netherlands
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9434-5/22/08.
https://doi.org/10.1145/3546037.3546065

2 The Pigasus Datapath

To put the design of Pigasus 2.0 into context, we give a brief overview of Pigasus's functionality and key modules. Pigasus is a signature-based IDS/IPS whose goal is to identify network flows which match one or more attack signatures ('rules'). The datapath consists of the following modules:

TCP Reassembler: Reconstructs an in-order bytestream from a sequence of out-of-order packets. For each flow, packets are stored in a linked list ordered by sequence number.

Multi-String Pattern Matcher (MSPM): A set of fast, lightweight filters that sift out innocent traffic entirely on the FPGA. Only 'suspicious' packets – typically <5% of the input packet rate – are forwarded to the CPU-side Full Matcher. The MSPM is further divided into the *Fast Pattern String Matcher (FPSM)*, *Header Matcher (HM)*, and *Non-Fast Pattern String Matcher (NFPSM)*.

CPU-side Full Matcher: A comprehensive rule checking engine that processes packets (*e.g.*, performing regular expression search) deemed to be suspicious by the MSPM.

3 Pigasus 2.0 Design

We now briefly describe the key features of Pigasus 2.0 that enable workload adaptation and robustness against ACAs.

3.1 Disaggregated Architecture

One of the major changes in Pigasus 2.0 is the use of a disaggregated architecture. In this architecture, all modules are independent services [5] connected via a standard streaming interface [4]. Disaggregation allows Pigasus' modules to be reused more easily—even outside Pigasus—and lets us scale them up or down independently to accommodate different workloads. Here, we describe some features of Pigasus 2.0 resulting from its disaggregated architecture.

Module scalability: Figure 1 shows the Pigasus pipeline with its five main modules: TCP Reassembler, FPSM, HM, NFPSM, and the DMA Engine. In this example only the NFPSM module is overloaded, while all other modules have spare capacity. Yet, Pigasus 1.0's monolithic architecture requires that we duplicate the *entire* pipeline in order to scale the design. This results in wasted FPGA resources, since most of the modules are overprovisioned. Because Pigasus 2.0

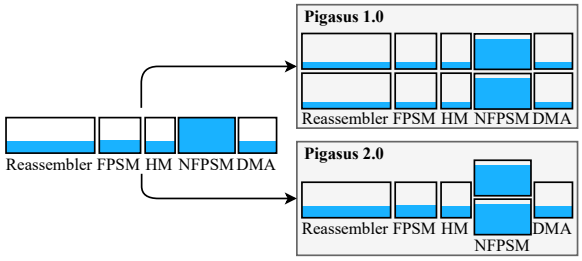

Figure 1: Scalability in Pigasus 1.0 (top) *vs.* Pigasus 2.0 (bottom). The fill level represents each module's load. Pigasus 1.0's monolithic design requires duplicating the entire pipeline, even though only the NFPSM is overloaded. Pigasus 2.0 allows per-module scaling.

employs a disaggregated architecture, users can choose to replicate only the overloaded module, achieving the same performance as Pigasus 1.0 while utilizing significantly fewer FPGA resources.

Parameterized modules: Besides scaling up individual modules by replicating them, Pigasus 2.0 exposes modules' internal parameters that let users grow or shrink the module without replication. For instance, some modules' bandwidth are dictated by a configurable 'width.' Users can increase the width to increase bandwidth or decrease it to save resources or make the design fit in a smaller FPGA. Parameterized modules also improves portability, allowing some of the modules to be used in applications besides Pigasus.

Pipeline recomposability: In Pigasus 2.0, users can freely recompose modules according to their needs. Users define a graph in a python-based DSL with all the modules that they desire. Users can even choose to spread modules across multiple FPGAs, connected using Ethernet, to let Pigasus scale to even higher packet rates. The tool then automatically generates the interconnect logic to implement the specified graph. The ability to specify different graphs also means that users can choose which modules to include in the pipeline. For instance, users with smaller FPGAs might choose to not include the NFPSM module, while users concerned with ACAs can choose to include SurgeProtector [1]. We also hope that Pigasus 2.0's recomposability will make it easier for other researchers to develop and include their own modules in Pigasus.

3.2 Dynamic Spillover Mechanism

While the disaggregated architecture gives users many knobs to tailor Pigasus to their own requirements and workloads, all the adjustments are made at compile time. To let the system adapt to changes in workloads at runtime, Pigasus 2.0 also provides a *dynamic* spillover mechanism. Since it is not possible to change the modules on the FPGA fast enough to respond to changes to incoming traffic, Pigasus instead relies on the CPU to absorb traffic when one of its modules is overloaded.

3.3 SurgeProtector

In order to maximize throughput while still meeting FPGA resource constraints, Pigasus heavily prioritizes common-case performance and memory efficiency. Unfortunately, some of these design decisions come at the cost of *poor resiliency against ACAs*, a potent class of DoS attacks. In an ACA, an adversary targets algorithms with high worst-case runtime complexity, inducing large amounts of work in the system using a small amount of attack bandwidth, displacing a significant fraction of innocent traffic in the process.

In the context of Pigasus, both the TCP Reassembler and Full Matcher are vulnerable to two different kinds of ACAs, allowing an attacker to displace upwards of 100 *bps* of innocent traffic for each *bps* of attack bandwidth they invest into the attack. Motivated by this very use-case, recent work proposed SurgeProtector [1], an adversarial scheduling framework that provides resilience against ACAs *without* sacrificing common-case performance. This framework was recently integrated into the datapath, and represents a key component of the Pigasus 2.0 artifact.

4 Demonstration Overview

Pigasus 2.0 is publicly available[1] and implements all the features described in this abstract. We exemplify the three key features described in §3 in the following demonstrations.

D1 – Different workloads and configurations: We will show how small changes to the configuration graph can trigger large changes in throughput for different workloads by shifting the bottlenecked module.

D2 – Dynamic Spillover: We will show how the dynamic spillover mechanism makes the system more robust to sudden bursts of packets that are bottlenecked by the same module on the FPGA.

D3 – SurgeProtector: We will show how attack workloads targeting Pigasus' TCP Reassembler and Full Matcher can drastically reduce throughput, and how adding the SurgeProtector module to the configuration graph can make the system robust to such attacks.

Acknowledgments

We thank the anonymous reviewers for their great comments and feedback. This work was supported in part by funding from a VMware Systems Research Award, NSF grant #1700521, and by Intel and VMware through the Intel/VMware Crossroads 3D-FPGA Academic Research Center.

References

[1] Nirav Atre, Hugo Sadok, Erica Chiang, Weina Wang, and Justine Sherry. 2022. SurgeProtector: Mitigating Temporal Algorithmic Complexity Attacks using Adversarial Scheduling. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA.

[2] Cisco. 2020. Snort 3. https://www.snort.org/snort3. (2020).

[3] Scott A. Crosby and Dan S. Wallach. 2003. Denial of Service via Algorithmic Complexity Attacks. In *12th USENIX Security Symposium (USENIX Security 03)*. USENIX Association, Washington, D.C. https://www.usenix.org/conference/12th-usenix-security-symposium/denial-service-algorithmic-complexity-attacks

[4] Intel. 2022. *Avalon Interface Specifications*. Technical Report 683091. Intel. https://www.intel.com/content/www/us/en/docs/programmable/683091/20-1/introduction-to-the-interface-specifications.html.

[5] Joseph Melber and James C. Hoe. 2020. A Service-Oriented Memory Architecture for FPGA Computing. In *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE Computer Society, Los Alamitos, CA, USA, 91–97. https://doi.org/10.1109/FPL50879.2020.00025

[6] Zhipeng Zhao, Hugo Sadok, Nirav Atre, James C. Hoe, Vyas Sekar, and Justine Sherry. 2020. Achieving 100Gbps Intrusion Prevention on a Single Server. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*. USENIX Association, 1083–1100.

[1] https://github.com/crossroadsfpga/pigasus