Fast Publish/Subscribe Using Linux eBPF

Michael Tatarski, Helge Parzyjegla, Peter Danielis, and Gero Mühl
Institute of Computer Science
University of Rostock, 18051 Rostock, Germany
michaeltatarski@yahoo.de, {helge.parzyjegla, peter.danielis, gero.muehl}@uni-rostock.de

***Abstract*—In recent years, the requirements for publish/subscribe systems have changed enormously. Mainly, this is due to the vastly increasing number of smart objects, services, and apps being connected to the cloud and their growing demand for bandwidth and realtime message processing. Unfortunately, many publish/subscribe systems are hardly able to handle both necessary and available data rates when being built on conventional network stacks that suffer from significant system overhead.**

In this paper, we present a novel architecture for fast publish/subscribe that consists of an edge broker for preprocessing notifications and a cloud broker that leverages eBPF to efficiently process network packages and speed up notification delivery. We address limitations of eBPF, discuss different filter options, and quantify the achievable performance. The evaluation confirms a significant performance improvement of the eBPF-enabled cloud broker when compared to a conventional implementation.

I. Introduction

Publish/subscribe is a versatile communication pattern [1] gaining a lot of interest in the Internet of Things (IoT) where it connects reams of smart objects over the cloud to managing services and (mobile) user applications [7]. In publish/subscribe, a data producer, also called a *publisher*, generates *notifications* about events of interest, e. g., a new sensor reading. A data consumer, also called a *subscriber*, uses a *subscription* to specify notifications in which it is interested to receive. An intermediary *notification service* is then responsible to deliver a published notification to all interested subscribers that have a matching subscription. The resulting many-to-many, indirect, data-centric, and asynchronous communication is characterized by a high degree of decoupling [1] well suited for flexible IoT applications.

The notification service is often realized by one or more interconnected publish/subscribe *brokers* that exchange notifications and subscriptions. Usually, brokers implement subscription matching and notification forwarding on the application layer using an overlay network which, on the one hand, eases the development of sophisticated notification dissemination strategies. On the other hand, those brokers suffer from significant system overhead when being built on top of those general-purpose network protocol stacks that are part of our contemporary operating systems [3].

In contrast, with the Linux *extended Berkeley Packet Filter (eBPF)* framework [2], it is possible to load specific application logic into deepest protocol layers in the operating system kernel or driver where network packets can be processed without much overhead. Experiments [5], [6] show a significant performance improvement and reduction in latency. In the following, we briefly discuss limitations of eBPF, sketch an architecture for eBPF-based publish/subscribe, and evaluate the achievable performance improvement.

II. eBPF Challenges

The performance improvement by eBPF is offset by numerous limitations [4] under which developers have to write kernel space programs. To ensure the security and stability of the Linux kernel, eBPF code is based on a restricted instruction set, executed in an in-kernel virtual machine, and verified before loading. Termination of eBPF programs is enforced by not allowing dynamic loops and backward jumps as well as restricting the calling of other eBPF programs in order to avoid infinite recursions. Although predefined data structures and libraries (i. e., helper functions) are available for stateful processing and more complex tasks, respectively, the management of the rather dynamic subscription tables for forwarding notifications becomes very challenging. Moreover, the only events that trigger the execution of network-related eBPF code is the reception and sending of packets making timeouts or periodic tasks (e. g., for the maintenance of data structures) impossible. Finally, for buffering notifications in order to better decouple publishers and subscribers in time, eBPF is particularly not suited.

III. eBPF-based Publish/Subscribe

To better cope with the eBPF challenges and limitations above, we split our publish/subscribe middleware into two main components: an edge broker and a cloud broker. The *edge broker* is built on conventional network libraries, offers a common publish/subscribe API to clients, and helps to decouple publishers and subscribers, for example, by buffering notifications whenever necessary. In addition, the edge broker preprocesses notifications, aggregates subscriptions, and labels their data packets accordingly before sending them into the cloud. There, the *cloud broker* leverages eBPF to process the packets as fast as possible and to forward the notifications to edge brokers with subscribed clients.

A. Cloud Broker Overview

The cloud broker's functions are implemented by several cooperating eBPF programs that are bound to two kernel network hooks. Programs at the *eXpress Data Path (XDP)* hook are run directly after the packet has been received by the network card. We mainly use the XDP hook to filter and classify received packets as notifications or subscriptions.

Due to a preprocessing and appropriate labeling at the edge broker, the registration of new subscriptions can already be accomplished by a subsequent eBPF program at this early stage. However, for forwarding notifications, we pass the packet towards the next hook.

At the *Traffic Control (TC)* hook, the kernel has parsed a packet's fields and metadata and exposes this information in the `sk_buff` data structure. Furthermore, helper functions are now available that ease replacing the packet's destination IP address and recalculating checksums in order to forward the notification to edge brokers with corresponding subscribers.

B. Channel-based and Topic-based Filtering

We support both channel-based and topic-based publish/subscribe. With channel-based filtering, notifications are published in a channel and forwarded to all subscribers of this channel. We use the channel number as index in a `BPF_ARRAY_OF_MAPS` to access the channel's forwarding table implemented as a simple `BPF_ARRAY` that holds all IP addresses of edge brokers with channel subscribers.

With topic-based filtering, notifications are published under a topic and forwarded to all subscribers of this specific topic or a superordinated topic, i. e., a higher-level topic that also includes the more specific one. We let the edge brokers map a notification's topic string to an index number which is attached as label to the notification's packet. This is done for each level of the topic hierarchy. Then, the cloud broker uses a separate `BPF_ARRAY_OF_MAPS` for each hierarchy level and performs multiple lookups with the corresponding label number, respectively, in order to gather all forwarding destinations.

Please note that the depth of the topic hierarchy, the maximum number of channels/topics (per hierarchy level), and the maximum number of subscribed edge brokers per channel/topic has to be statically set at compile time in order to comply with eBPF restrictions (cf. Sect. II). If these do not meet the runtime conditions, we simply ignore topics from levels that are too low, merge them if channel numbers overflow, and in case of too small forwarding tables, replace the oldest entry with a new subscription.

IV. Evaluation

In our evaluation, we compared the performance of the eBPF-based cloud broker to a user space broker of equal functionality that is written in C and uses conventional datagram sockets provided by Linux. We measured the number of forwarded notifications per second for a growing set of subscribers, i. e., subscribed edge brokers, for both channel-based and topic-based publish/subscribe. For topic-based filtering, the topic hierarchy had three levels.

Figure 1 shows the evaluation results. The number of processed notifications decreases in all configurations the more subscribers are present to which a notification copy needs to be forwarded. Channel-based filtering forwards more notifications than topic-based filtering since it only requires one instead of three forwarding table lookups. Finally, the eBPF broker clearly outperforms the user space implementation by a factor of 20 to 25 for both the channel- and the topic-based variant.

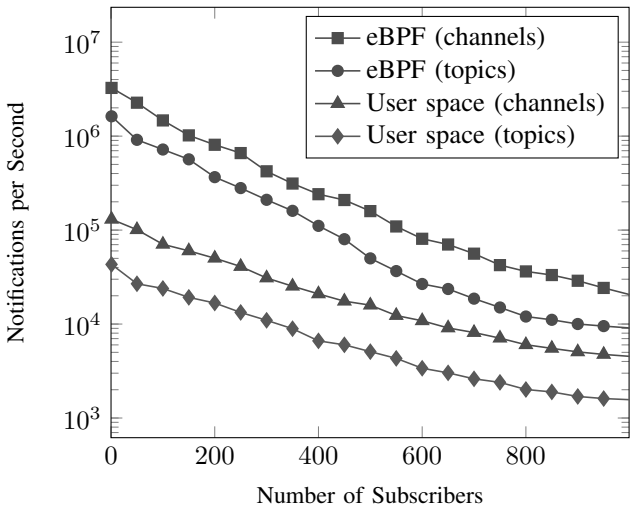

Figure 1. Forwarded notifications for an increasing number of subscribers.

V. Conclusions

In this paper, we leveraged Linux eBPF to speed up the processing and forwarding of notifications in channel-based and topic-based publish/subscribe. We splitted our publish/subscribe middleware into an edge broker programmed with conventional means and an eBPF-based broker in the cloud. By preprocessing notifications and subscriptions at the edge, we were able to ease the forwarding in the cloud and to alleviate some of the limitations of eBPF code. Our evaluation showed that the eBPF-based broker outperforms a conventional user space implementation by a factor of 20 to 25. This enormous performance improvement justifies certain concessions w.r.t. the dynamics of the application environment. In future, we want to work on delivery guarantees for a reliable notification dissemination as well as on strategies to scale out eBPF-based broker deployments in the cloud.

References

[1] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, June 2003.

[2] Linux Foundation. eBPF website. https://ebpf.io.

[3] I. Marinos, R. N. Watson, and M. Handley. Network stack specialization for performance. *SIGCOMM Comput. Commun. Rev.*, 44(4):175–186, Aug. 2014.

[4] S. Miano, M. Bertrone, F. Risso, M. Tumolo, and M. V. Bernal. Creating complex network services with eBPF: Experience and lessons learned. In *IEEE Int. Conf. High Perform. Switch. Routing (HPSR)*, pages 1–8. IEEE, 2018.

[5] S. Miano, F. Risso, M. V. Bernal, M. Bertrone, and Y. Lu. A framework for eBPF-based network functions in an era of microservices. *IEEE Trans. Netw. Serv. Manag.*, 18(1):133–151, Mar. 2021.

[6] D. Scholz, D. Raumer, P. Emmerich, A. Kurtz, K. Lesiak, and G. Carle. Performance implications of packet filtering with linux ebpf. In *30th Int. Teletraffic Congr. (ITC)*, pages 209–217. IEEE, 2018.

[7] Y. Sun, X. Qiao, B. Cheng, and J. Chen. A low-delay, lightweight publish/subscribe architecture for delay-sensitive iot services. In *IEEE Int. Conf. Web Serv. (ICWS)*, pages 179–186. IEEE, 2013.