Content-based Publish/Subscribe in Software-defined Networks

Helge Parzyjegla, Christian Wernecke, and Gero Mühl
Institute of Computer Science
University of Rostock, 18051 Rostock, Germany
{helge.parzyjegla, christian.wernecke, gero.muehl}@uni-rostock.de

***Abstract*—With SDN, content-based publish/subscribe can be implemented on the network layer instead of using an application layer broker network. We present two methods realizing notification distribution with OpenFlow and P4, respectively.**

I. Introduction

Publish/subscribe is a flexible communication mechanism ideally suited for building loosely coupled distributed applications for the Internet of Things (IoT), the cloud, or a regular data center. Application components interact with each other by taking over the role of a publisher and/or a subscriber. *Publishers* produce *notifications* containing the data they want to share whereas *subscribers* consume notifications relevant to them. To determine the relevance of a notification, a subscriber creates a *subscription*. In the *content-based* model, the subscription contains a filter selecting notifications based on their individual content. Likewise, publishers may use an *advertisement* with a filter to describe the content of the notifications they are going to produce. The flexibility of publish/subscribe finally results from the indirect and data-centric communication making it easy to scale and extend applications by replicating components and adding new ones.

To foster the loose coupling of application components, the distribution of notifications is delegated to a separate *notification service*. The notification service is often realized by a set of cooperating *brokers* exchanging advertisements, subscriptions, and notifications to ensure that a published notification is delivered to all subscribers with a matching subscription. Usually, brokers implement matching and routing on the application level and forward notifications between brokers using an *overlay network*. On the one hand, this eases the realization of sophisticated content-based distribution strategies significantly, but, on the other hand, it also introduces a considerable processing delay at each broker (e. g., for traversing the network stack and the (de)serialization of the notification's content) and may lead to inefficiencies if the overlay network does not fit the physical network topology. *Software-defined networking (SDN)*, however, enables new implementations of content-based publish/subscribe. With network elements (e. g., switches) becoming increasingly intelligent, they can take over more and more dedicated broker functions. Eventually, this allows for notification distribution strategies realized completely on the network layer that drastically reduce latency and increase network efficiency.

II. SDN-based Notification Delivery

Although SDN-enabled switches feature processing capabilities, they are not designed to inspect the content of a notification. Hence, the SDN-based delivery of notifications differs from a broker-based approach in that it requires an additional preprocessing step. In fact, the SDN-based publish/subscribe lifecycle works as described in the following.

First, publishers and subscribers both register at the SDN controller. This way, the controller learns where all notification sources and sinks are located in the network and can, thus, construct appropriate delivery trees. The controller installs required forwarding rules at the switches and tells the publishers about those subscriptions that may be matched by a notification they produce. The publisher determines which individual subscriptions match the notification's content and labels the notification correspondingly so that the switches can apply the previously installed forwarding rules. The switches, then, ensure that the notification eventually reaches all subscribers. If advertisements and subscriptions are issued or revoked, the SDN controller is informed again so that it can adapt delivery trees and notify all affected publishers about the changes.

Based on this publish/subscribe lifecycle, we have implemented two approaches for content-based publish/subscribe in software-defined networks. They differ in terms of used protocols and standards (i. e., OpenFlow vs. P4), the encoding and storage of distribution information (i. e., protocol headers and flow rules), and the accuracy of the notification delivery (i. e., perfect vs. approximate with false positives).

A. Managing Forwarding Trees with OpenFlow

Our first approach [8] is implemented using OpenFlow [7], which is currently the most prominent SDN standard. With OpenFlow, however, we are restricted to the header fields of existing protocols (e. g., UDP, IP) and those operations that are defined for these fields in the OpenFlow standard. Since the set of receivers of a notification is determined dynamically from the notification's content, it may be different for each notification. Moreover, the number of receiver combinations grows exponentially with an increasing number of subscribers exceeding quickly the available label or address space of network protocols. Therefore, we decided to encode the set of receivers approximatively using a Bloom filter.

A *Bloom filter* [2] is a probabilistic data structure based on a bit vector of fixed length that uses hashing to store the members of a set, e. g., those subscriptions that are matched by a notification. With a certain (small) probability, a membership query may return a *false positive*, i. e., indicate a match although the subscription does not match. We embed the Bloom filter into the IPv6 source and destination addresses which were not used otherwise resulting in a combined vector of 256 bits. Using subnet/bitmask operations, the switches can examine individual bits of the Bloom filter and check whether a subscription is matched. Based on these checks, we install appropriate forwarding rules for each subscription on the switches, which have to be updated whenever new subscriptions are issued or existing subscriptions are revoked.

We have investigated and compared several optimizations that exploit similarities between active subscriptions in order to reduce the number of required forwarding rules as well as the fraction of false positives [8]. The latter can be kept below a certain threshold at the cost of increased network traffic by distributing a notification with several messages. To cope with the restricted storage capacity of the switches, we have also developed a strategy to merge forwarding rules that, however, leads to more false positives. Nevertheless, this strategy can be combined with the optimizations above to mitigate the effect.

B. Multicast Source Routing with P4

Our second approach [9] is based on *P4* [3] that can be thought of as a next generation OpenFlow. P4 allows to define and process custom protocol headers for publish/subscribe that we use to realize a *multicast source routing*. The distribution tree of a notification is encoded into a stack of message headers that are added by the publisher. When a switch receives such a notification, it examines the message headers to find out to which of its connected neighbors (i. e., switches and subscribers), it has to forward the message.

We have implemented several different methods to encode the distribution tree into the notification message as well as to process it on the switches. We evaluated the methods in a larger emulated network and compared the results with other approaches (e. g., unicast, broadcast, overlay network) with regard to protocol overhead and notification delay. In particular, the evaluation shows that the notification delays can largely be reduced compared to an overlay network.

Please note that the P4 program for processing the header stacks is static and does not depend on the set of active subscriptions. Hence, the switches do not need to be updated when subscriptions are issued or revoked which allows for applications in highly dynamic environments. We also implemented a hybrid approach [10] that uses header stacks to dynamically extend stored forwarding trees. This is well suited for scenarios with a stable subset of subscribers that receive a large fraction of all published subscriptions so that maintaining a forwarding tree actually pays off. For the latter, however, it is crucial to correctly partition and classify the subscribers.

III. Related Work

Bloom filters have many applications in distributed systems including publish/subscribe middleware. *XSiena* [4] encodes all matching subscription predicates into a Bloom filter attached to the notification. Although this avoids a time-consuming reevaluation of predicates at downstream brokers, forwarding decisions are still taken on the application layer causing a substantial delay. *Lipsin* [6] offers a network layer approach that encodes all links of the notification's distribution tree into a Bloom filter enabling network elements to make fast forwarding decisions. Lipsin was implemented using a prototype of a NetFPGA-based router. In contrast, *Pleroma* [1] works on SDN-capable switches using OpenFlow. It assumes that the notification space can be divided into subspaces which are mapped to binary strings representing IP multicast address ranges so that they can be managed using standard wildcard/masking operations without Bloom filters.

There have already been stateless multicast approaches based on source routing. Being very space-efficient, *COXcast* [5] mathematically encodes the whole distribution tree into a single identifier attached to the message header. Unfortunately, it is not fully SDN-ready yet.

IV. Conclusions and Future Work

We have presented two approaches to implement content-based publish/subscribe with OpenFlow and P4, respectively. Both approaches make all forwarding decisions on the network layer enabling low notification latencies. For future work, we want to investigate more efficient encodings tailored to given application scenarios and supported by real-world data sets.

References

[1] S. Bhowmik, M. A. Tariq, B. Koldehofe, F. Dürr, T. Kohler, and K. Rothermel. High performance publish/subscribe middleware in software-defined networks. *IEEE/ACM Transactions on Networking*, 25(3):1501–1516, June 2017.

[2] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. 13:422–426, July 1970.

[3] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, July 2014.

[4] Z. Jerzak and C. Fetzer. Bloom filter based routing for content-based publish/subscribe. In *2nd International Conference on Distributed Event-based Systems*, DEBS '08, pages 71–81. ACM, 2008.

[5] W.-K. Jia. A scalable multicast source routing architecture for data center networks. *IEEE Journal on Selected Areas in Communications*, 32(1):116–123, Jan. 2014.

[6] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander. Lipsin: Line speed publish/subscribe inter-networking. *SIGCOMM Comput. Commun. Rev.*, 39(4):195–206, Aug. 2009.

[7] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow. *ACM SIGCOMM Computer Communication Review*, 38(2):69, 2008.

[8] H. Parzyjegla, C. Wernecke, G. Mühl, E. Schweissguth, and D. Timmermann. Implementing content-based publish/subscribe with OpenFlow. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, SAC '19, pages 1392–1395, New York, NY, USA, 2019. ACM.

[9] C. Wernecke, H. Parzyjegla, G. Mühl, P. Danielis, and D. Timmermann. Realizing content-based publish/subscribe with P4. In *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN 2018)*, pages 1–7, Piscataway, NJ, USA, Nov. 2018. IEEE.

[10] C. Wernecke, H. Parzyjegla, G. Mühl, E. Schweissguth, and D. Timmermann. Flexible notification forwarding for content-based publish/-subscribe using P4. In *2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN 2019)*, pages 1–5, Piscataway, NJ, USA, Nov. 2019. IEEE.