The Power of Software-defined Networking: Line-rate Content-based Routing Using OpenFlow

Boris Koldehofe, Frank Dürr, Muhammad Adnan Tariq, Kurt Rothermel
Institute of Parallel and Distributed Systems
University of Stuttgart
⟨firstname.lastname⟩@ipvs.uni-stuttgart.de

ABSTRACT

A lot of research effort has been invested to support efficient content-based routing. Nevertheless, practitioners often fall back to far less expressive communication paradigms like multicast groups. The benefits of content-based routing in minimizing bandwidth consumption are often rendered useless by simpler communication paradigms that rely on line-rate processing of data packets at the switches of the network providers. Contrary content-based routing protocols face the inherent overhead in matching the content of events against subscriptions leading to far lower throughput rates and higher end-to-end delays. However, recent trends in networking such as software defined networking in combination with network virtualization have tremendous potential to change the picture. In our opinion this will significantly increase acceptance of sophisticated middleware like content-based routing in the future. To support our claims we outline in this paper a reference architecture that may be used to build middleware for Future Internet applications. Furthermore, we provide a solution for realizing content-based routing at line-rate relying on this reference architecture and illustrate research problems that need to be addressed.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Distributed networks; C.2.4 [**Distributed Systems**]: Distributed applications[2cm]; D.2.11 [**Software Architectures**]: Data abstraction

Keywords

Content-Based Routing, Publish/Subscribe, Software Defined Networking, Network Virtualization

1. INTRODUCTION

Content-based routing as provided by publish/subscribe (pub/sub) systems has evolved as a key paradigm for interactions between loosely coupled application components (content publishers and subscribers). The basic idea of content-based routing is to utilize the diversity of information exchanged between application components to increase the efficiency of forwarding. Using content-based forwarding rules (also called content filters) installed on content-based routers (also termed brokers), bandwidth-efficiency is increased by only forwarding content to the subset of subscribers who are actually interested in the published content.

Many middleware implementations for content-based pub/sub have been developed over the last decade (e.g., [15, 4, 14, 8, 3, 20, 21]). These approaches have proven to efficiently support content-based routing between a large number of distributed application components. However, implemented on the application layer, their performance is still far behind the performance of communication protocols implemented on the network layer w.r.t. throughput, end-to-end latency, and bandwidth efficiency. A standard multilayer switch or hardware router can forward packets at line-rate achieving data rates of 10 Gbps and more using dedicated hardware such as TCAM memory. Moreover, they allow for switching delays of only few microseconds. Finally, routing on the network layer is more bandwidth efficient since it avoids sending the same information over the same physical link multiple times, in contrast to an overlay network where multiple logical links might share the same physical link.

Therefore, it would be highly attractive to implement content-based routing directly on the network layer. However, since changes to existing standard network protocols and hardware seemed to be unrealistic (as the slow support of the highly anticipated IPv6 standard shows), current research refrains from network layer implementations, and instead tries to improve application layer approaches along several dimensions: 1) inferring the underlay topology from the overlay topology [9, 12, 6], 2) specific hardware allowing for efficient matching of advertisement and subscriptions on network brokers [18], and 3) reducing the expressiveness of content-routing to topic-based pub/sub [10].

Note, inferring the underlay topology using latency spaces as proposed, for instance, by Vivaldi [6], comes at a significant cost. Despite this effort, it is still hard to accurately infer advanced link state information such as the current link utilization based on observations on end systems. Moreover, relying on dedicated hardware for matching dramatically reduces the scope to which such a middleware can be deployed. Therefore, it seems reasonable to sacrifice the expressiveness of content-based pub/sub if line rate forwarding becomes the major requirement. For instance, LIPSIN [10] proposes topic-based pub/sub utilizing IP multicast for line-rate forwarding.

Instead, we argue that the basic assumption of all of these approaches, which assume that changes to network protocols are practically infeasible, has changed significantly with the advent of new networking technologies, namely, software-defined networking and network virtualization. These trends are going to dramatically impact how to configure pub/sub middleware in the future.

MW4NG '12 December 3-7, 2012, Montreal, Canada

Software-defined Networking. The adoption of the OpenFlow standard [5] in state of the art switches supports a decoupling of the control plane and data (forwarding) plane. In future middleware, it will be possible to configure the forwarding tables of switches directly and "on-the-fly" using a controller process implemented in software running on an external host. Currently, this technology is already heavily used to configure communication flows in datacenters or campus networks in order to optimize network throughput [2], or to configure virtual networks [13]. The OpenFlow standard also gained strong support from network operators and hardware manufactures. Companies such as IBM, NEC, and HP already offer first OpenFlow switches. The example of Google, which has adopted OpenFlow throughout their networks recently, shows that software-defined networking is ready to be used in productive systems [23].

Network virtualization. While it seems highly unlikely that an Internet service provider (ISP) will provide applications direct access to its routers and switches, the advent of virtual networking will allow applications to access and configure a network of virtual routers that in the future even will spread over multiple ISPs. Software-defined networking using OpenFlow will also here be the paradigm to flexibly configure the behavior of switches, which then can be mapped to their physical counterparts without loss in performance.

This paper we show that these trends will indeed allow for a general purpose pub/sub middleware that results in comparable performance to network layer implementations, i.e., pub/sub systems supporting line-rate message forwarding, microseconds switching delay, and close to optimal bandwidth efficiency not only with respect to the overlay network but also the underlay network topology. To this end, we present a reference architecture allowing for the embedding of more sophisticated complex middleware solutions – in particular, content-based pub/sub – by utilizing the OpenFlow specification. Finally, we conclude with research problems we consider worthwhile pursuing in the future.

2. REFERENCE ARCHITECTURE

Figure 1 illustrates the reference architecture we envision for the configuration of group communication middleware in the Future Internet, and on which we will rely in the remainder of the paper to establish line-rate content-based routing. In this architecture, *network virtualization* serves us as a basic abstraction to allocate a network of virtual switches[1]. While state of the art approaches already allow to allocate virtual networks of a single ISP [19], we will assume in line with current research initiatives such as [1, 17] that an application can allocate a virtual network over multiple ISPs. For instance, in Figure 1 the virtual network consists of four switches from three different ISPs. This way it will be possible to provide access points very close to subscribers and publishers and establish routes within the network that provide predictable end-to-end connectivity. The ISPs will provide the application with the communication characteristics of the links connecting virtual switches as well as a specification of the capabilities of the switches.

Once the middleware has selected an appropriate network of switches and connected individual application components to the switches (cf. *virtual network configuration* and *access control* in Figure 1), we still need an abstraction to configure the virtual switches (in particular, their forwarding tables) minimizing bandwidth

[1]In this work, we use the term "switch" to refer to multilayer switches that can base forwarding decisions on layer 2 to layer 4 header information such as MAC addresses, IP addresses, and port numbers

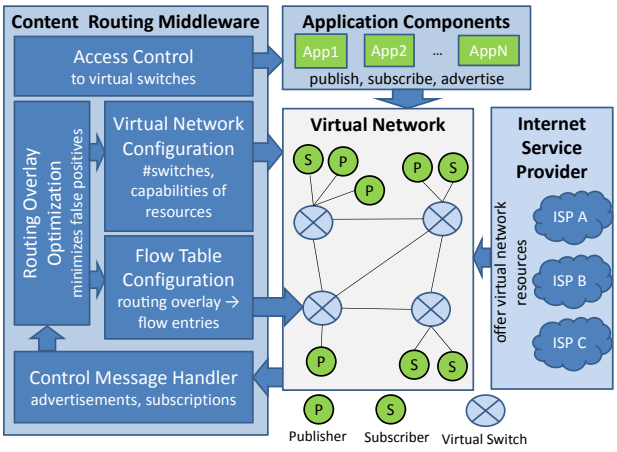

Figure 1: Architecture for configuration of virtual networks.

consumption inside the virtual network, and to map the configuration to the ISP's physical infrastructure (cf. *flow table configuration* in Figure 1). It will be crucial that the mapping of virtualized resources to the physical resources can happen without sacrificing the performance of the switches. In our reference architecture, we rely on *software-defined networking* as the abstraction to configure the virtual switches. The basic idea behind OpenFlow is to let a controller directly modify the forwarding table (also called flow tables) of a switch, i.e., to define the outgoing ports for messages of a certain communication flow. Examples of flows are a TCP connection (defined by source/destination IP addresses and source/destination port numbers), packets from a specific MAC address, or packets labeled with a certain VLAN tag. The definition of flow table entries can either be done proactively a priori to receiving packets of a certain flow, or reactively when the switch receives a packet of an unknown flow without matching flow table entry for the first time. After the flow table entry has been configured, forwarding happens at line-rate using dedicated hardware for matching flow table entries – typically, a hardware switch can match the header information of an incoming packet to all of its flow table entries (up to 150,000 possible entries) in one clock cycle using ternary content-addressable memory (TCAM).

The controller, which configures flow table entries, is typically a process hosted on an external machine connected to the switches via a control network (either implemented as a separate network or "in-band" via the data network). Any data packet for which a switch has no predefined flow configured will be forwarded to the controller. The logic of the controller is in our case defined by the content-routing middleware, i.e., all control messages are received by the *control message handler*, and changes to the content-based routing overlay are computed accordingly by the *content-routing overlay optimization* component. The *flow table configuration* component then implements those changes by reconfiguration of appropriate switches of the virtual network. The component decides whether and on which of the switches a new flow table entry needs to be established or where changes to the current flow tables are required. This way a decoupling between control and data plane is achieved, i.e., for each established flow, packets are forwarded at line-rate and only unknown packets will trigger a reconfiguration of the network.

Since in our reference architecture the middleware performs the reconfiguration of virtual switches, a further mapping stage of flow tables from virtual switches to physical switches of the ISP is needed.

ISPs in addition need to map each flow entry to all physical switches that establish a virtual link. For realizing the virtual link abstraction, the ISPs again can rely on software-defined networking such as OpenFlow. Hence, we assume that processing of flows specified for a virtual link is efficiently performed in hardware.

For the rest of this paper, we assume the availability of at least the OpenFlow standard 1.2 to configure switches (in particular, the functionality to define forwarding actions for groups, i.e., multiple output ports, and IPv6 address support for flow definitions).

3. OPENFLOW-BASED PUB/SUB

In the following sections, we will first describe the basic approach to enable content-based routing using the OpenFlow architecture and afterwards present two concrete realizations of a content-based pub/sub system.

3.1 Overview of Forwarding

In line with our reference architecture the control component of OpenFlow will forward all pub/sub relevant control traffic to the pub/sub middleware's control handler. In particular, the control traffic comprises all subscriptions indicating the interest of a subscriber in events and all advertisements indicating the intention of a publisher to send events with certain attributes. This way the pub/sub middleware establishes i) a *global view on the subscribers* (in more detail, their location in the network represented by the switches they are attached to) and their subscriptions, ii) a *global view on publishers*, their network locations, and the events they might send.

Based on the global view on subscriptions and advertisements, routing overlay optimization and flow table configuration *identify and proactively install suitable flow table entries* on the switches. To this end, content attributes need to be mapped to flow identifiers that can be interpreted by switches for forwarding. Candidates of such identifiers are the header information of standard layer 2 to layer 4 headers (10 tuple of header fields as defined by OpenFlow) since these are the fields that can be interpreted by existing multilayer switches. Using other address information for switching would require modifications to hardware switches, which are unlikely in the near future until pub/sub gains the same importance as protocols like TCP/IP or UDP/IP. Since the IP destination address field is the common field for addressing end systems in a routed network, we decided to use the destination IP address field for defining pub/sub flows. Moreover, we chose IPv6 as target protocol because of the restricted address space of IPv4.

As a constraint, we have to make sure that the flows defined for pub/sub applications do not interfere with other communication flows such as TCP connections, or IP multicast communication flows sharing the same switches. Since pub/sub is a specific kind of group communication, we decided to use IP addresses from the IP multicast address range, in more detail, IPv6 multicast addresses with global scope (address prefix `ff0e::/8`) since we aim for a global pub/sub service spanning several ISPs.[2] Therefore, we have to make sure that other IP multicast flows are not using the same addresses as pub/sub flows. Hence, we have to allocate a certain address range from the global IPv6 multicast addresses for pub/sub.

The number of addresses in this range (denoted as N) is an important parameter influencing the performance of our pub/sub middleware by controlling the granularity of the flows between publishers and subscribers. For instance, a larger N facilitates creation of a large number of flows, thereby saving network bandwidth usage by minimizing the forwarding of unnecessary events on each flow. Considering the huge address space of IPv6, allocating a larger portion for pub/sub (for instance, a prefix of 28 bits, i.e., 100 bit pub/sub addresses) seems to be possible leaving enough space for all other (current and future) IP multicast applications. However, such an allocation is subject to standardization and beyond a technical discussion.

One address from the range (denoted as IP_{fix}) is fixed to be used to identify new subscriptions and advertisements in the system for building the global view on subscriptions and advertisements mentioned above, while the rest of the addresses are used to define flows.

Published events are *matched to installed flows* according to their attributes and *forwarded based on the flow address* (IPv6 multicast address field). Since forwarding is solely based on flows addresses, it is performed by switches in hardware at line-rate and microseconds delay per switching operation. Note, since flow table entries are installed proactively according to advertisements and subscriptions, event messages will not require any additional handling by the controller. This way any additional delays and reduction in throughput are avoided.

3.2 Pub/Sub Operations

As the next step we detail further how subscriptions and advertisements are performed. An application component specifies its interest in receiving certain information by sending a subscription sub_1 to the OpenFlow switch to which the component is connected. The header of the subscription message contains IP_{fix} as a destination address. This destination address is used to identify a new subscription that has to be forwarded to the control handler to build the mentioned global view on subscriptions. We simply achieve the forwarding to the control handler by *not* installing a flow table entry for the address IP_{fix}. Since the default action of OpenFlow is to forward every packet without matching flow table entry to the controller, the controller receives the subscription automatically without the need for an extra flow table entry.

When a new subscription sub_1 is received by the control handler, the routing overlay optimization algorithm is executed to determine the flows for the new subscription. In general, the subscriber will be connected to all flows that cover the subscription sub_1 (i.e., all flows that forward events matching sub_1). This possibly requires the creation of new outgoing ports to existing flow table entries by adding or updating actions (entries) in the flow tables of one or multiple OpenFlow switches. This is performed by the flow table configuration component by sending Flow-mod messages to the corresponding switches. Moreover, the routing overlay optimization algorithm may instruct the flow table configuration to create new flows (or remove existing flows) to optimize event routing from publishers to subscribers.

Similarly, an application component expresses its intent to publish a particular kind of information by issuing an advertisement to the OpenFlow switch. The advertisement is also forwarded to the controller using the address IP_{fix}. By following the same chain of steps new flows will be created or existing flows are updated ensuring each subscriber will be covered by the resulting set of flows.

In the following, we will describe in more detail two possible realization of a content-based pub/sub system using our OpenFlow architecture, namely, channelization and in-network filtering.

3.3 Channelization

An important concern in a content-based pub/sub system is to avoid forwarding of events to the paths in the network where only non-

[2]IPv6 addresses with local organizational scope could also be used, for instance, if the pub/sub system is only to be deployed within a datacenter or local network of one organization. As a drawback, such addresses are not accessible outside the organization.

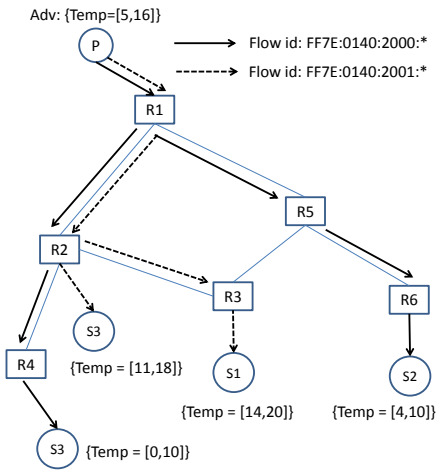

Figure 2: Channelization example.

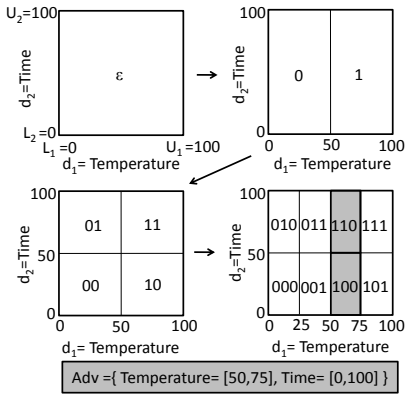

Figure 3: Decomposition of 2-dimensional space.

matching subscriptions are connected. From this point of view, channelization is one promising way to reduce the forwarding of unnecessary events by mapping advertisements and subscriptions to a limited set of channels such that the event dissemination within each channel is very efficient w.r.t. the reduction of unnecessary events. In the setting of our flow-based approach, channels are attractive because they can be easily mapped to flows as shown below.

Typically, two approaches can be used to create channels. The first approach uses absolute (structural) similarity between the subscriptions and the advertisements (such as the area occupied by the intersection of two subscriptions) to calculate their closeness to be placed in the same channel. This approach restricts the content-based model to predefined attributes with ordered data types and known domain (i.e., numeric attributes).

An alternative approach, which often yields more efficient channelization and places no more restrictions on the content-based model, is to rely on the event traffic published/matched by the advertisements/subscriptions in the recent past. However, in addition to control messages dealing with advertisements and subscriptions, the routing overlay optimization requires information on recently published events that in this case needs to be collected in the control network. In this case, each subscriber/publisher periodically forwards the list of recently received events to IP_{fix} which then will be used by the routing overlay optimization algorithm to recalculate and install new channels. Alternatively, this information can also be asynchronously collected from the per-flow statistics (flow packet counter) maintained at each OpenFlow switch by the flow table configuration component.

The routing overlay optimization algorithm will in both cases rely on channelization methods like spectral clustering [22] to create clusters of subscribers and publishers. Each channel is treated as a separate flow and is assigned a unique address from the range of IPv6 multicast addresses reserved by the application (cf. Figure 2). The maximum number of channels is limited by the range of reserved addresses. However, the runtime number of channels in the system depends on the channelization algorithm and can be dynamically adjusted according to the subscriptions and the advertisements (as well as the event traffic in case of second approach) in the system. Our initial results show for uniform as well as zipfian distributions –modeling the diversity of interest of subscribers – up to 100 channels are sufficient to perform efficient content-based filtering in pub/sub systems.

Once the channels are calculated, the routing overlay optimization calculates for each channel a minimum spanning tree which overall results in minimum bandwidth consumption. The flow table configuration component installs a separate flow for each channel by adding/updating actions in flow table entries of the OpenFlow switches in the network. In addition, it is necessary to send to each publisher the flow id (i.e., IPv6 address) of all channels to which it is supposed to forward its publications along with the aggregated subscription of those channels. This step is necessary because a publisher may publish an event that matches subscriptions belonging to multiple different channels. If the channel information is not available at the publishers, then each event should initially be sent to the controller to determine the matching flows (channels), which significantly increases the bandwidth utilization of the link to the controller as well as effects the line-rate forwarding of events.

3.4 In-network Filtering

The channelization approach does not allow for the possibility to prune unnecessary messages within each channel. An event forwarded on a channel is always delivered to the subscribers (issuers) of all the participant subscriptions. In this section, we present an approach to facilitate the filtering (pruning) of events within the switch network. First, we detail the method to decompose subscriptions, advertisements, and events into a spatial representation. Later, we describe how the spatial representation can be used to perform in-network filtering of events in the network of switches.

3.4.1 Spatial Indexing

The content-based schema consisting of d attributes can be modeled geometrically as a d-dimensional space (denoted as Ω) such that each dimension represents an attribute. We employ spatial indexing to divide the d-dimensional space into regular sub-spaces that serve as enclosing approximations for subscriptions, advertisements, and events [21]. As illustrated in Figure 3, any subspace can be identified by a binary string called a *dz-expression*. In particular, dz-expressions fulfill following properties:

1. The shorter the dz-expression the larger is the corresponding sub-space in Ω.

2. A sub-space represented by dz-expression dz_1 is *covered by* the sub-space represented by dz_2, iff dz_2 is a prefix of dz_1.

The subscription/advertisement can be composed of several dz-expressions. For instance, in Figure 3, the spatial representation

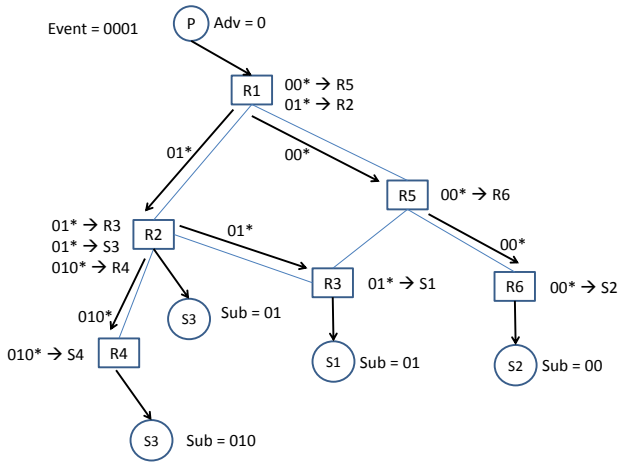

Figure 4: In-network filtering example.

of advertisement $Adv = \{Temperature = [50, 75] \wedge Time = [0, 100]\}$ requires two dz-expressions $\{100, 110\}$. Nevertheless, an event is represented by the smallest (finest resolution) sub-space that encloses the value represented by it.

3.4.2 Event Filtering

The content routing optimization uses the covering (or containment) relation between the spatial representations of newly arrived and the existing advertisements/subscriptions to set up flows between publishers and subscribers. In particular, each flow is associated with a sub-space and is identified by the corresponding dz. On the arrival of new advertisement Adv_1, the routing optimization algorithm checks for the containment relation between the dz of Adv_1 and the dz associated with the existing flows[3], and performs one of the three actions: (1) If the dz of Adv_1 is covered, then the newly arrived advertisement is mapped to the existing flows. For instance, a newly arrived advertisement 0 is mapped to the existing flows 00 and 01. (2) If dz of an existing flow is covered, then the existing flow is divided into sub flows. For instance, existing flow 0 is divided into sub flows 00 and 01 on the arrival of an advertisement 00. (3) If no containment relation exists, then a separate flow is created for the newly arrived advertisement.

Similarly, the arrival of a new subscription may divide an existing flow into sub flows. For instance, in Figure 4, the arrival of $sub = \{00\}$ divides the flow 0 into two sub-flows $\{00*, 01*\}$ (note that only few least significant bits from the bits representing the range of reserved IPv6 addresses are shown to reduce complexity). The symbol $*$ is used to represent standard wildcard/masking operations, which are supported by hardware switches for matching IP addresses using Class-less Interdomain Routing (CIDR). Such wildcards are naturally supported by TCAM memory typically used in switches to store "don't care" values $(*)$ besides 0 and 1. Therefore, an event (e.g., 0100) can be matched against the dz of the flow (e.g., $01*$) in hardware by the switch during forwarding. The creation of two sub flows reduces unnecessary events by limiting the forwarding of events matching $01*$ to the OpenFlow switches $R5$ and $R6$ as well as subscriber $S2$ in the example. Likewise, the advent of subscription $sub = \{010\}$ creates a new flow $010*$ to avoid forwarding of events matching $011*$ to the network path connecting subscriber $S3$.

[3]In case an advertisement is represented by multiple dz-expressions, the covering relation is checked for each dz separately.

3.4.3 Discussion

It is worth noting that the total sub-spaces (created as a result of spatial indexing) depend on the number of decompositions of the d-dimensional space. Each decomposition step generates sub-spaces with finer granularity, capable of representing subscriptions/advertisements with higher accuracy. For instance, using the decomposition of Figure 3, a subscription $sub_1 = \{Temperature = [0, 100] \wedge Time = [0, 25]\}$ can be represented by the sub-spaces $\{000, 001, 100, 101\}$. These sub-spaces do not provide an accurate representation, and hence may match events that do not belong to the original subscription (sub_1) resulting in the dissemination of unnecessary events in the switch network. However, if another decomposition step is performed on dimension d_2, then sub_1 can be accurately mapped by the newly generated sub-spaces, i.e., $\{0000, 0010, 1000, 1010\}$.

Clearly, fine grain sub-spaces are desirable as they avoid forwarding of unnecessary events. However, the decomposition steps needed to generate sub-spaces with finer granularity also increase the length of the dz-expressions (representing those sub-spaces). In practice, the length of dz-expression and hence the number of sub-spaces are limited by the range of IPv6 multicast addresses reserved by the application. In Section 3.1, we argued that reserving an address range with 100 bits seems to be possible as it leaves sufficient addresses to be used for other purposes.

An important research challenge in this direction is to develop methods to lower the number of bits needed to perform efficient in-network filtering of events. For instance, one promising solution is to perform spatial indexing only on those dimensions that correspond to more selective attributes (i.e., filtering on the attributes results in less number of unnecessary messages). The attributes for the spatial indexing can be dynamically selected by the controller as a result of periodic collection of traffic statistics from the OpenFlow switches. We envision the study of different methods to perform efficient in-network filtering of events utilizing a smaller number of bits as an ongoing future work.

4. CONCLUSION AND FUTURE WORK

In this paper, we have shown that software-defined networking provides a powerful abstraction to configure middleware – in particular, publish/subscribe middleware – for the Future Internet with the potential to yield significant performance gains. While our concrete future work will be on extending and evaluating the approaches towards content-based routing, there is a number of interesting research questions that are worthwhile to pursue by a larger research community. Those comprise:

Minimizing flow table size. Typically, a switch only has up to 150,000 flow table entries which are shared between all applications. In this paper, we have outlined two approaches that can adjust the trade-off between the number of flow table entries and bandwidth efficiency (number of false positives). However, further research is required to find an optimal solution given a restricted number of flow table entries.

Scalable controller. The controller has to perform computational complex tasks such as optimal route calculation (distribution trees) or subscriber clustering. In particular, these tasks become challenging in large-scale and highly dynamic systems with larger topologies, many publishers and subscribers, high churn rates, dynamic link state, etc. Although a single controller might look like a potential bottleneck at first sight, it could still be implemented scalably by utilizing, for instance, the large resources of data centers ("the cloud"). This requires suitable algorithms that scale up to many cores and scale out to multiple machines. Another possibility

to improve scalability is a distributed controller as described next.

Decentralized control and coordination. Although we assumed a single logical controller in this paper, the control plane could be distributed to several controllers to improve scalability and robustness. However, this raises several questions: How many controllers do we need and where to place them [7]? And how to coordinate controllers in order to achieve a consistent and optimal behavior, while letting each controller base its decisions on a local and/or aggregated view? For instance, controllers could be organized hierarchically, or in a (flat) peer-to-peer topology, both requiring different coordination concepts.

Correct operation and verification. Distributed protocols make it hard to achieve correct operation such as loop-free forwarding or constant reachability, in particular, during transitional phases like the re-configuration of distribution trees. However, central control and a globe view on the system – as used in this paper – facilitate the design of correct algorithms and ease the verification of the running system as first approaches demonstrate [11, 16].

Quality of Service (QoS). Many applications such as network control systems rely on QoS properties like a maximum end-to-end delay. This requires the implementation of resource reservation mechanisms or scheduling algorithms for forwarding. Since forwarding is done on the network layer, we can benefit from existing layer 3 protocols and the access to switches/routers – assuming suitable interfaces for virtualized switches/routers –, which are likely to outperform application layer approaches.

Virtual network abstractions. Providing a virtual network spanning resources from multiple providers is a problem that we did not focus on in this paper. However, for achieving optimal performance one question is how many and which physical switches and routers should be exposed in the virtual network? In a multi-provider (ISP) scenario, one might also be able to choose from switches of alternative providers based on advanced optimization metrics like monetary costs.

5. REFERENCES

[1] B. Ahlgren, P. A. Aranda, P. Chemouil, S. Oueslati, L. M. Correia, H. Karl, M. Söllner, and A. Welin. Content, connectivity, and cloud: ingredients for the network of the future. *IEEE Communications Magazine*, 49(7):62–70, 2011.

[2] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation (NSDI)*, pages 19–19, 2010.

[3] J. A. Briones, B. Koldehofe, and K. Rothermel. SPINE : Adaptive Publish/Subscribe for Wireless Mesh Networks. *Studia Informatika Universalis*, 7(3):320–353, 2009.

[4] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[5] O. M. E. Committee. *Software-defined Networking: The New Norm for Networks*. Open Networking Foundation, 2012.

[6] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: a decentralized network coordinate system. *ACM SIGCOMM Computer Communication Review*, 34:15–26, 2004.

[7] B. Heller, R. Sherwood, and N. McKeown. The controller placement problem. In *Proceedings of the First Workshop on Hot Topics in Software-defined Networks (HotSDN)*, pages 7–12, 2012.

[8] H.-A. Jacobsen, A. K. Y. Cheung, G. Li, B. Maniymaran, V. Muthusamy, and R. S. Kazemzadeh. The PADRES publish/subscribe system. In *Principles and Applications of Distributed Event-Based Systems*, pages 164–205. 2010.

[9] X. Jin, W. Tu, and S. H. G. Chan. Scalable and efficient end-to-end network topology inference. *IEEE Transactions on Parallel and Distributed Systems*, 19(6):837–850, 2008.

[10] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander. LIPSIN: line speed publish/subscribe inter-networking. In *Proceedings of the ACM SIGCOMM conference on Data communication*, pages 195–206, 2009.

[11] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey. VerfiFlow: Verifying network-wide invariants in real time. In *Proceedings of the First Workshop on Hot Topics in Software-defined Networks (HotSDN)*, pages 49–54, 2012.

[12] M. Kwon and S. Fahmy. Path-aware overlay multicast. *Computer Networks*, 47(1):23–45, 2005.

[13] J. Marias, E. Jacob, D. Sanchez, and Y. Demchenko. An OpenFlow based network virtualization framework for the cloud. In *Proceedings of the IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 672–678, 2011.

[14] G. Mühl. *Large-Scale Content-Based Publish-Subscribe Systems*. PhD thesis, TU Darmstadt, November 2002.

[15] P. Pietzuch. *Hermes: A Scalable Event-Based Middleware*. PhD thesis, University of Cambridge, Feb 2004.

[16] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for network update. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 323–334, 2012.

[17] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Cáceres, M. Ben-Yehuda, W. Emmerich, and F. Galán. The reservoir model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, 53(4):1–11, 2009.

[18] M. Sadoghi, H. Singh, and H.-A. Jacobsen. fpga-ToPSS: line-speed event processing on fpgas. In *Proceedings of the 5th ACM international conference on Distributed event-based system (DEBS)*, pages 373–374, 2011.

[19] G. Schaffrath, C. Werle, P. Papadimitriou, A. Feldmann, R. Bless, A. Greenhalgh, A. Wundsam, M. Kind, O. Maennel, and L. Mathy. Network virtualization architecture: proposal and initial prototype. In *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures (VISA '09)*, 2009.

[20] A. Tariq, B. Koldehofe, G. Koch, and K. Rothermel. Providing probabilistic latency bounds for dynamic publish/subscribe systems. In *Proceedings of the 16th ITG/GI Conference on Kommunikation in Verteilten Systemen (KiVS)*, pages 155–166, 2009.

[21] M. A. Tariq, B. Koldehofe, G. G. Koch, I. Khan, and K. Rothermel. Meeting subscriber-defined QoS constraints in publish/subscribe systems. *Concurrency and Computation: Practice and Experience*, 23(11):2140–2153, 2011.

[22] M. A. Tariq, B. Koldehofe, G. G. Koch, and K. Rothermel. Distributed spectral cluster management: A method for building dynamic publish/subscribe systems. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems (DEBS)*, pages 213–224, 2012.

[23] Urs Hölzle. OpenFlow @ Google. Open Networking Summit, 2012.