SDN-like: The Next Generation of Pub/Sub

Technical Report

Kaiwen Zhang and Hans-Arno Jacobsen

University of Toronto

Abstract. Software-Defined Networking (SDN) has raised the boundaries of cloud computing by offering unparalleled levels of control and flexibility to system administrators over their virtualized environments. To properly embrace this new era of SDN-driven network architectures, the research community must not only consider the impact of SDN over the protocol stack, but also on its overlying networked applications. In this big ideas paper, we study the impact of SDN on the design of future message-oriented middleware, specifically pub/sub systems. We argue that key concepts put forth by SDN can be applied in a meaningful fashion to the next generation of pub/sub systems. First, pub/sub can adopt a logically centralized controller model for maintenance, monitoring, and control of the overlay network. We establish a parallel with existing work on centralized pub/sub routing and discuss how the logically centralized controller model can be implemented in a distributed manner. Second, we investigate the separation of the control and data plane, which is integral to SDN, which can be adopted to raise the level of decoupling in pub/sub. We introduce a new model of pub/sub which separates the traditional publisher and subscriber roles into flow regulators and producer/consumers of data. We then present use cases that benefit from this approach and study the impact of decoupling for performance.

1 Introduction

Software-Defined Networking [20] offer unprecedented levels of flexibility and control to administrators of cloud-based datacenters. By employing a logically centralized controller, which raises the level of abstraction of networking components into a single unified view, the complexity of configuring the network is significantly reduced, which translates to cost savings for large corporate datacenters. Furthermore, the separation of the control and data planes allows for more flexible and powerful routing policies to be expressed, which results in improved performance. As an example, Google's inter-datacenter WAN powered by an SDN has achieved 95% network utilization [11]. While still in an early stage, SDN is expected to fully realize the vision of the Infrastructure-as-a-Service provider model.

In light of this new approach to networking, the research community must adapt networked applications to harness the potential of SDN. One fundamental type of applications is message-oriented middleware, such as pub/sub [7]. Since those applications are abstractions of the underlying network layer, it is essential

that they are capable of leveraging the capabilities of the protocol stack. For instance, [15] proposes a pub/sub architecture where the controller is employed to disseminate routing information to SDN-enabled switches using the content-based matching semantics of the pub/sub system. By mapping pub/sub content to OpenFlow flow entries, the resulting proof-of-concept system is capable of achieving line-speed matching and dissemination of publications.

The above line of work represents the most straightforward approach to integrating SDN with pub/sub. However, we argue that there exists a more fundamental study of SDN which will benefit pub/sub. By distilling the key concepts behind the success of SDN, we can reason about their relative impact within the context of pub/sub. At a higher level, many of the concerns encountered in the networking community have an equivalent in pub/sub. For instance, overlay construction for pub/sub is analogous to topology construction of the underlying network. Along the same lines, we can extrapolate that the design principles behind SDN, which are motivated by traditional networking issues, can be applied to pub/sub to solve similar challenges.

In this big ideas paper, we present a new model of pub/sub which is "SDN-like". We build our model based on two key concepts of SDN: The logically centralized controller and the separation of the control and data plane. Enhancing the traditional pub/sub architecture with a centralized controller raises the level of control over the topology of the network by introducing a monitoring entity and a bootstrapping process. The separation of the control and data plane can be achieved in pub/sub through the decoupling of the traditional publisher and subscriber roles into advertiser/producer and interest manager/consumer, respectively. We describe how separating the control plane (advertisements and subscriptions) and the data plane (publication production and consumption) in pub/sub pushes further onward the commonly established idealogies of pub/sub and can benefit certain application scenarios. This decoupling allows for more powerful features to be supported by pub/sub systems, such as installing specific control policies over a subset of publishers and subscribers.

The contributions of this paper are thus as follows:

- We propose a new breed of SDN-like pub/sub, where we argue that integrating key concepts borrowed from SDN can solve analogous problems in pub/sub.
- We present the design of a logically centralized controller for monitoring and controlling the pub/sub overlay.
- We advocate and demonstrate the benefits bedind a division of existing pub/sub roles to decouple the control and data plane: Advertiser/producer for publisher and interest manager/consumer for subscriber.
- We illustrate that the support for policy-based control of pub/sub agents will allow for more meaningful access to data.
- We present a proof-of-concept architecture with the matching protocol suite to power our SDN-like pub/sub model.
- Finally, we present a study of use cases that will benefit from our model,

2 Background

In this section, we describe in more detail our pub/sub reference model. We also elaborate on SDN, its key concepts, and the advantages it offers over traditional networking.

2.1 Content-based publish/subscribe

Pub/Sub is a messaging paradigm that allows for loosely coupled communication between data producers (called publishers) to data consumers (called subscribers). Events (referred to as publications) flow from the publishers through an overlay network of brokers, which route the message traffic towards the intended recipients. In the content-based matching model [4], subscribers can express their interest as a conjunctive list of predicates. Publications, which are lists of attribute-value pairs, are considered to match if they satisfy all the predicates of a subscription, in which case they must be delivered to the subscription source.

In terms of routing, a variety of routing algorithms exist. Advertisement-based and subscription-based forwarding respectively flood advertisements or subscriptions through the overlay [2]. In the case of advertisement-based forwarding, the subscriptions are floated towards matching publishers using the reverse path from the advertisements. The delivery tree for publications is computed on a hop-by-hop basis, with each broker matching the publication to the next set of hops containing a matching subscription.

Another routing model we consider is *rendezvous-based*. One broker in the overlay is designated as the rendezvous node and is in charge of computing the pub/sub matching [21]. The rest of the brokers simply relay all incoming traffic to the rendezvous node. This model simplifies the routing process and reduces the matching overhead by having a centralized matcher, at the expense of non-optimal routing paths and potentially limited scalability.

We classify the publication flow as constituting the *data plane*, whereas the control plane is comprised of advertisements and subscriptions. The pub/sub system is therefore in charge of disseminating the message traffic in the data plane to the correct recipients as guided via the information from the control plane.

One key property of pub/sub is its loosely coupled nature. This decoupling typically spans three dimensions: Space, time and synchronization. By employing a pub/sub middleware, data producers and consumers are not aware of the identities of other participants, at the time the data is produced or consumed, or they need not to directly exchange data with one another. Decoupling is a critical aspect of pub/sub, since the lack of coordination allows pub/sub to leverage highly asynchronous protocols for scalability. Furthermore, decoupling reduces the complexity of the pub/sub model, which supports only a limited number of functions. This decoupling is made possible due to the variety of application scenarios with loosely synchronized distributed components for which pub/sub constitutes an ideal choice as a dissemination layer.

2.2 Software-defined networking

SDN is a new paradigm for networking, abstracting the underlying network as an unified entity, which can be manipulated through a controller [20]. Although not a clean-slate proposal for a future Internet architecture, SDN does require specialized hardware for switches and routers to be controllable from a remote location (i.e., controller). The two properties of SDN we focus on are:

Separation of the control and data plane: SDN-enabled switches receive routing instructions in the form of policies over the traffic metadata. This is realized through the OpenFlow standard, which classifies traffic into multiple flows, with each flow associated to a sequence of actions [19]. For instance, a simple OpenFlow-enabled pub/sub solution could classify flows per topic, with each topic being associated to a multicast group corresponding to the subscribers of that topic.

Logically centralized controller: The aforementionned decoupling is leveraged through the use of a logically centralized controller, such as NOX [12] or Floodlight [9]. The controller maintains a unified view of the network and enforces a set of global policies. Common applications for a controller include monitoring of the topology and dynamic rerouting for load-balancing purposes. Although logically centralized, the controller itself can be scaled through physical distribution [23].

Both properties together significantly reduce the complexity of configuring individual switches, since they only respond to a single SDN controller. Furthermore, the centralized view of the network allows for more dynamic policies to be enforced by the controller, resulting in more efficient and scalable networks.

3 Centralized controller for pub/sub

We propose the use of a logically centralized controller in pub/sub for monitoring and reconfiguration of the overlay broker network. The unified view of the overlay allows for dynamic routing policies in case of congestion and failures, akin to its counterpart in SDN. In addition, we employ the pub/sub controller during a boostrap process, to setup the overlay and disseminate configuration properties to the brokers and clients.

Fig. 1 shows the general architecture of a pub/sub system with a controller. Every node in the system, which includes clients (publishers and subscribers), must establish a connection to the controller through a bootstrapping process. The controller can return the necessary configuration properties to the node. For instance, new brokers require the identity of neighbors within the determined topology. The bootstrapping process could also be used to provide a unique identifier to each node.

Communication to the controller is regulated through a separate channel as it is the case in SDN. The separation of channels reduces the complexity associated with handling mixed traffic over one channel and allows this channel to be tuned for communication with the controller. For instance, a client needs to establish

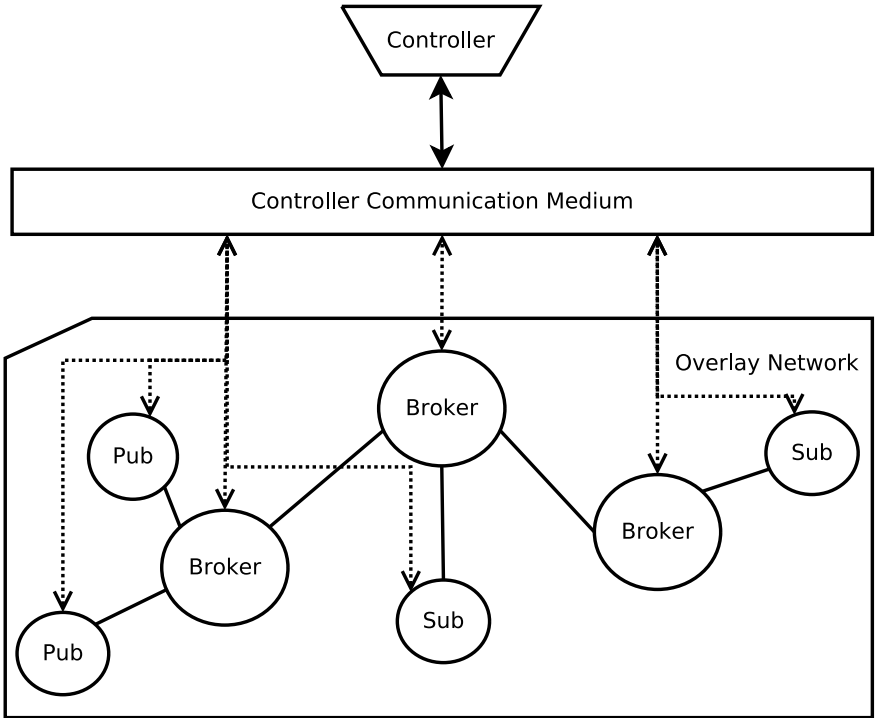

Fig. 1. Controller architecture

only a short-lived TCP connection to the controller for bootstrapping purposes. Passing this traffic through the overlay would unnecessarily burden the system.

The main role of the logically centralized controller is topology maintenance. By having global knowledge of the network, the controller can adaptively reconfigure the topology to be fault-resilient and efficient (i.e., exhibit low node fanout, be forced acyclic, or have a set number of redudant path between every pair of system edge brokers, where redundancy may go as far as requiring the existance of physically disjoint path.) Although there exists a number of decentralized solutions for efficient overlay construction [5], we argue that providing global knowledge can help the controller make more informed decisions. This design would eschew the need for complex distributed protocols, which reduces the complexity of operating brokers. In this setting, the brokers are lightweight entities which listen to instructions from the logic maintained at the controller. Furthermore, the controller requires an incremental overlay construction algorithm since it is aware of churn in the system. Currently incremented solutions for overlay constructions are reactive in nature [6] and seek to restablish sought properties after the churn has occured. By enhancing the controller with global knowledge, we should be able to achieve a higher prediction accuracy and create topologies that are resilient despite churn.

Another role for the centralized controller is dynamic routing policies. We can establish a parallel between the publication space of pub/sub and the flows of OpenFlow to maintain on-demand quality of service. For instance, publications of a certain kind (i.e., a certain topic or including certain attribute-value pairs) could be prioritized. To solve congestion, traffic can be labelled according to pub/sub meta-data and rerouted to alternative paths. To support this feature, the logically centralized controller needs to fetch data from brokers on a regular basis to monitor the current state of the traffic. This requires the brokers to maintain long-lived connections to the controller, which raises scalability questions. To address this problem, we can either employ another overlay-based channel for broker-controller interactions (such as another pub/sub system), or distribute the controller (while maintaining logical centrality).

4 Separation of control and data in pub/sub

In Sec. 2.1, we described how pub/sub decoupling enables publishers and subscribers to produce and consume data without space, time or synchronication coupling. The lack of coordination and the limited degree of coupling gives the pub/sub system ample flexibility to provide relaxed consistency and reliability guarantees while maintaining simple matching semantics in a scalable and efficient manner. However, one dimension which remains coupled is the control and data plane. In other words, the producers and consumers of data in pub/sub remain in control of the data itself. For instance, this means that a subscriber submits subscriptions in order for content to be delivered to itself. In this case, we see that the subscriber controls the type of data that is being delivered to itself. By explicitly sending its own subscriptions, each subscriber is completely aware of the type of publications it expects to receive.

We seek to lift this binding by introducing the concept of control and data decoupling in pub/sub. To do so, we break down the traditional roles of publishers and subscribers into advertisers and producers as well as interest managers and consumers, respectively. Advertisers and interest managers operate on the control plane, installing policies which regulate the advertisements (ads) and subscriptions (subs) of producers and consumers, respectively. At the most fundamental level, advertisers and interest managers are able to use *remote advertisements and subscriptions*, i.e., perform advertising and subscribing on behalf of other clients. We thus broke down the coupling of the data and control flows by allowing for controls to direct data to a different destination as opposed to from where the controls were originating.

We argue that the data and control binding in traditional pub/sub is an an artificial construct with no real justification behind it. Due to the loosely coupled nature of pub/sub applications, there was no need to differentiate between data and control. The use cases assume that clients either formulate simple ads or subs over a set of predetermined topics (channel-based) or a limited publication space (topic-based or content-based). Since there is no coordination amongst clients once the system is online, publishers and subscribers are given the freedom to

control their message content. However, we have discovered that new applications for pub/sub contain complex specifications which require more sophisticated pub/sub semantics, which can translate in uncertainties in the advertisement and subscription spaces [18]. In other cases, pub/sub is employed to support Big Data applications where subscribers are not expected to consume the entire stream of publications, but would rather limit themselves to a summary, which could be obtained via top-k filtering [25] or aggregation [10].

In those situations, the data sinks and sources might not be the most suitable entities to formulate their own ads or subs. A third party, equipped with more intimate knowledge of the current state of the system, is able to express or transform ads and subs accordingly. Another possibility is for the third party controller to leverage the flexibility in ads and subs to enhance the internal performance of the pub/sub system in a transparent manner to the end-user. Separating the control and the data plane allows for remote parties to make such adjustments whenever adequate, for example, when the principal entities would be unable to do so.

Furthermore, the model we propose can be considered a generalization of pub/sub. This decoupling is orthogonal to other properties of pub/sub and to its matching semantics. The traditional roles of publishers and subscribers are retained as physically co-located data and control entities. From the end-user point of view, each pair of roles can be considered as a logically unified publisher or subscriber. However, the SDN-like pub/sub system is agnostic to the binding and considers each role in the pair as separate entities. What the decoupling allows is for data consumers to receive data without specifying its own subscriptions or for a subscriber, who has its own subscriptions, to have its subscription space altered by remote interest managers. We therefore envision a model where mixed roles are compatible with one another: The system can allow controllers, data sources and sinks, to be located anywhere in the topology (see Fig. 2). Advertisers and interest managers are able to control the data flow from and to publishers and subscribers as well as producers and consumers alike.

We also note that while the separation of concerns is inspired from SDN, the purpose of the decoupling is different. In SDN, the separation works in tandem with the centralized controller whereas in SDN-like, the logically centralized controller presented in Sec. 3 is by and large orthogonal to this property. While it is feasible for the centralized controller to act as a producer and interest manager, it is not a necessary condition. For instance, we allow multiple advertisers to effect the same producer. The relationship between control roles can be structured either as a hierarchy, where higher level entities have the ability to modify or override the ads and subs of lower level ones, or as a collection of administrative domains, where each control entity is in charge of regulating a region of the publication space. Implementation of SDN-like pub/sub will however require changes to the routing protocols, which we will discuss in Sec. 5.

We now delve into the details of the new roles, their principal usage, and the new functions required to empower those roles.

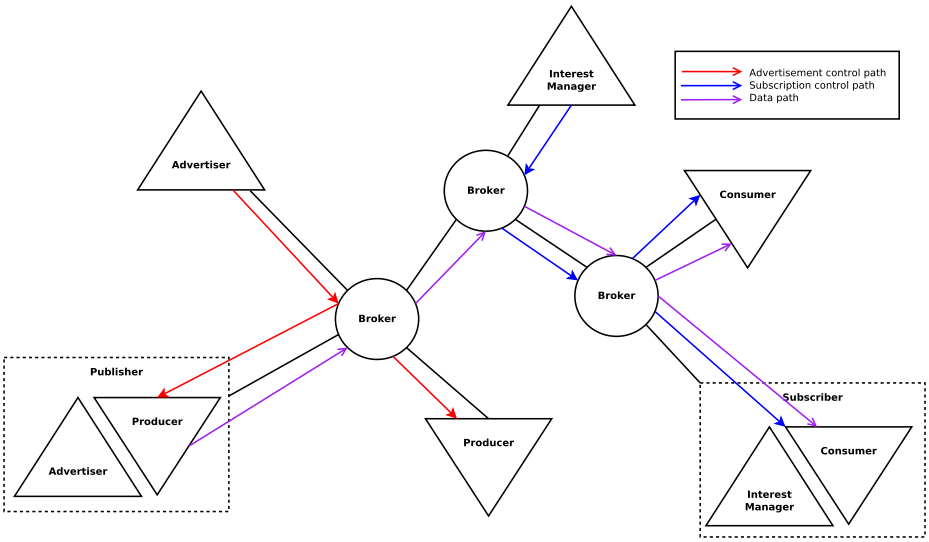

Fig. 2. Mixed SDN-like topology with decoupled control and data plane

4.1 Advertiser and producer roles

We define the functions of the advertiser at two levels: advertisements and publications.

Advertisement level - First, advertisers are allowed to create *advertisement policies* which regulate what advertisements a producer will obtain. In a traditional pub/sub model, an advertisement is an auxiliary type of control message used to set routing paths for a publisher. In our context, we leverage the property that a publisher must advertise over a certain space before publishing within that region to employ advertisements as an *access control* mechanism.

Since producers are decoupled from advertisers, they are not aware of the advertisements they possess and can potentially publish data outside of their advertised range. In that case, two scenarios can occur: Either the publication is silently dropped, or a feedback message is returned to the source. Both type of responses are valid depending on the application. Silently dropping publications might be useful for lightweight publishers (e.g., sensors) that are specialized in outputting events at a fixed frequency, while feedback might guide a self-tuning publisher to internally filter on its own before publishing, or to package its publications in a format that is compatible with the advertisements.

Beside the traditional advertisements targeting single producers, advertisers can enact advertisement policies based on the metadata supplied by producers. One possibility is to employ policies when producers publish data in a highly structured manner. For example, each publisher advertises on the topic of its own country. This can be formulated as an advertisement policy which extracts the country from each advertiser's metadata and produces the corresponding advertisement. Although this type of simple advertisements could easily be advertised manually by each publisher using existing models, the presence of an advertisement policy creates an invariant which can be leveraged to optimize the system, either through clustering techniques or by pruning the data structure storing advertisements.

Another way to employ policies is by altering existing advertisements. A monitoring advertiser can prioritize certain producers by restricting others and reducing their advertising space. Top-k filtering can also be achieved at a per-source level by analyzing publications from various producers and "turning off" those less relevant.

Publication level - Secondly, advertisers can be equipped to express *publication* policies. Those policies can formulate the appropriate publication semantics to attach to a particular content to be published. For example, using producer metadata and by inspection of a publication payload, a broker can decide to publish that publication over a certain topic. This strategy could be used for top-k filtering: A producer could start off with its publications being disseminated in a lower level channel before the advertiser learns enough about the producer to elevate it to a higher status, and delivering its future publications into a topic for higher quality content.

Another reason for using *publication* policies is to push the decoupling of pub/sub semantics even further. By using *publication* policies, the producer does not need to be aware of pub/sub semantics at all. It is simply handing this data to the broker, which attaches to it the necessary pub/sub header such that the content will be forwarded to interested parties. For security and privacy issues, it might be desirable to expose as little information as possible about the pub/sub system to the producers so they cannot infer anything about the system based on the pub/sub semantics they publish on.

4.2 Interest manager and consumer roles

At the receiving end, the decoupling of control and data can be used to satisfy very dynamic subscription patterns. Interest managers submit *subscription policies* which control the subscriptions consumers have based on collected metadata and the state of their current subscriptions.

One use for subscription policies is to support fine-grained subscriptions with constant churn. Due to a massive publication space, certain applications demand that their subscribing clients submit very fine-grained subscriptions and continuously increment them as needed. This is, for instance, true for location-based

applications, where subscribers subscribe to a small area around their current location and continuously update as they move. In such cases, a subscription policy can be parametric (see [14]) to serve as a template for a subscription everytime the location metadata is updated. As for advertisements, the policy serves as an invariant which can be taken into account when optimizing the system.

Another use is for consumers with fuzzy or unknown interest. In such cases, the consumer may be lacking the necessary knowledge about the state of the system to create the precise subscription. In the traditional model, the subscriber would have to subscribe first to a larger space of content and then downsize it to the relevant content, which is inefficient. Another possibility is that the interest of a consumer is conditional upon the state of the system. For instance, it is conceivable for a consumer to subscribe to the publications belonging to topics associated with publishers that are located near the subscriber. In order for a subscriber to create those subscriptions, it would have to subscribe to the position updates of every publisher and create new subscriptions whenever one of those publishers is within range of the subscribe. A much better approach is to let an interest manager keep track of the position of every producer and consumer in the system and remotely generate subscriptions whenever a consumer is close to a producer. In such scenario, it is sufficient for a single interest manager to monitor the position updates of all entities in the system, rather than having every subscriber monitor everything.

Finally, subscription policies can be used for generating subscriptions for flexible consumers that are efficient with regards to the current state of the system. This is the case when subscribers only require a summary or a sample of the most relevant publications. The interest manager can gauge the granularity of the subscription to satisfy such consumers. Another possibility is that a consumer is currently interested in multiple topics but do not require data from every topic. A smart interest manager can evaluate, based on current traffic amongst other things, which subscriptions particular consumers can unsubscribe to alleviate the load of the system.

5 Reference architecture

We present a reference architecture for SDN-like pub/sub. This will server as a proof of concept by demonstrating the feasibility of our model, shown through the reusability of existing pub/sub components (such as the matching engine) to serve the additional functionalities required. We also explain the various protocols and interactions found in our model.

The architecture and its subscription language is based on PADRES [8], a content-based pub/sub system operating over a federated overlay broker network. Publishers and subscribers are considered clients - we will extend that notation for advertisers/producers and interest managers/consumers. Each client connects to a single broker to access the system (called edge broker). The overlay consists of brokers connected to one another, which each broker only aware of

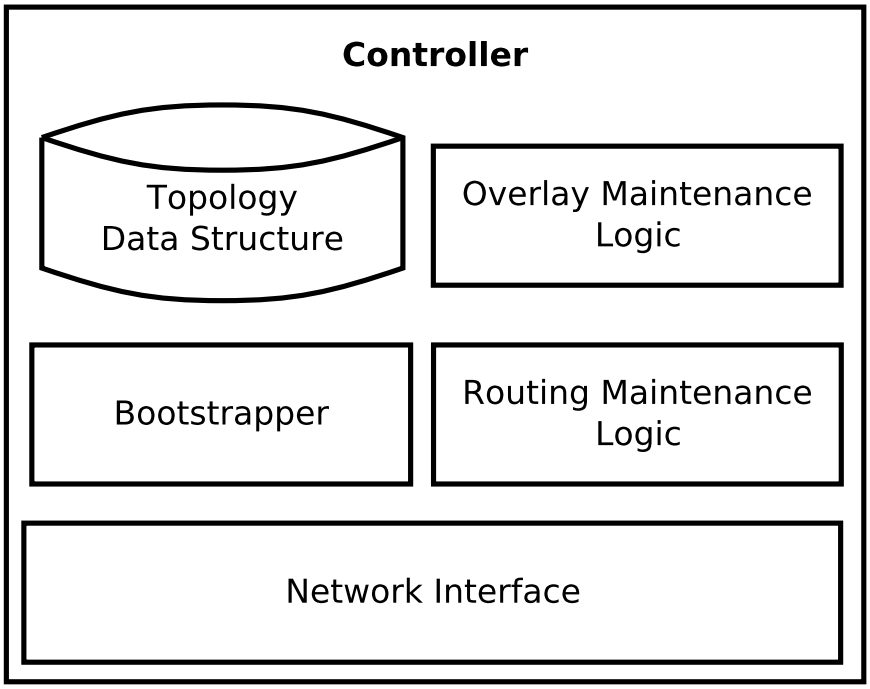

Fig. 3. Controller components

its neighbors. The topology can be cyclic: the routing protocol ensures cycle-free delivery paths [17].

5.1 Controller architecture

Fig. 3 lists the various components of the centralized SDN-like controller:

Bootstrapper
Each node joins the system by first contacting the controller. The bootstrapper module builds the configuration properties to be installed by the node. A counter is maintained by the bootstrapper to assign an unique identifier to the node. The bootstrapper also polls the overlay maintenance component to determine the placement of the node in the overlay. If it's a client, the bootstrapper will determine which edge broker it must connect to. For a broker, it builds a list of neighboring brokers to connect to. The topology data structure is then updated.

Network interface
Nodes establish a direct connection to the controller via TCP. Brokers must maintain long-lived connections and listen for instructinos from the controller, while clients can disconnect once they pass the bootstrapping sequence. Clients may receive commands from the controller via their edge

broker. Clients must also establish a connection to the controller when leaving the system, which will remove the node from local storage.

Topology data structure
The controller collects various statistics about the brokers, which are packaged as regular broker information messages. These messages also serve as heartbeat messages for failure detection purposes. The information is compiled in the topology data structure, where the statistics for individual brokers is maintained.

Overlay maintenance logic
This component runs an incremental overlay construction algorithm to decide the optimal placement of a node at the moment it enters the system. Furthermore, the overlay is maintained by regularly monitoring the state of the topology as stored in the controller, and computing the necessary overlay transformations required. Those operations are then sent to the appropriate nodes. In order to contact a client, the command is instead sent to its edge broker. For instance, a broker may be asked to shed clients to a different broker. Those clients will then proceed in a transitionary migration phase [13]. The overlay maintenance is also in charge of repairing the topology when failures are detected. Clients who were previously connected to the failed node will eventually time out and restart the bootstrapping process in order to be assigned to a new edge broker.

Routing maintenance logic
The routing maintenance module leverages the information stored to detect congested nodes and links. Using a cost model, the component evaluates possible routing changes to alleviate congestion. If none are satisfactory, the component instead looks for possible topology reconfiguration. Any changes made are stored in the topology database and communicated to the overlay maintenance logic.

5.2 Policy mechanisms

We now describe the mechanisms associated with policies. This involves the definition of metadata and policy languages, routing the related information and generating new advertisements and subscriptions.

Metadata language In order to interact with policies set by advertisers/interest managers, producers and consumers expose some metadata to the pub/sub system. Metadata contains basic information about the client (such as ID) as well as application-specific information (such as geographical location). Metadata messages are formulated in the same format as publications, as lists of attribute-value pairs.

A producer/consumer is responsible for updating its metadata whenever a change occurs by sending a new metadata message. A tradeoff between the size of the metadata and the update frequency is therefore observed. Including highly dynamic data in the metadata will trigger a high update rate which can cause a significant overhead for the system.

Policy language Policies contain two parts. The first part is a list of predicates which are matched against the state of every producer/consumer, which consists of its metadata and current advertisements/subscriptions. The second part is the set of instructions to be followed whenever a matching producer/consumer is found.

Essentially, the conditional section of a policy takes on the same format as a subscription. We can therefore reuse the matching engine to match policies over metadata. The difference is that policy matching constitutes the reverse operation: instead of matching a publication against known subscriptions, we are instead matching subscriptions (*policies*) against known publications (*metadata*). Since publications can be considered subscriptions containing solely equality predicates (ie. no ranges), data structures used to store subscriptions can easily be adapted to store metadata. What is needed is an indexing structure to query subscriptions and advertisements by source for fast lookup during policy matching involving the advertisement/subscription status of a given producer/consumer.

Once matched, the possible instructions include:

Insert ad/sub x Create the specified advertisement/subscription for the matched target. The ad/sub can contained variable data which is replaced by metadata fields at the time of generation.

Insert unad/unsub x Finds ads/subs matching the specified argument x and send an unadvertisement/unsubscription message to remove x. If x is not found, ignore this command.

Modify x y Finds ads/subs matching x and modify them according to y. y contains modifications to the ad/sub, which could be in the form of insertion, deletion, or modifications of predicates. y can use read values for attributes in the existing predicates of x.

Routing Metadata and policies can routed according to a variety of strategies. Strictly speaking, they must be routed to a quorum of brokers in order to guarantee that at least one can match every pair of metadata and policy. We propose three different strategies:

Metadata flooding Metadata are flooded in the network. Policies can then be matched directly at the edge broker of corresponding advertiser/interest manager. This strategy is beneficial only if the workload contains more metadata than policies. The disadvantage is that edge brokers connected to advertisers/interest managers must collect all advertisements (already done in advertisement-based forwarding) and all subscriptions in order to compute matches involving ads/subs.

Policy flooding Policies are flooded in the network. Metadata can then be matched directly at the edgde broker of the producer/consumer. This approach also has the advantage that it is compatible with any forwarding model, since the edge broker will contain all the subscriptions of collected metadata and therefore has the necessary knowledge to computing matches for those producers/consumers.

Rendezvous-based Policies and metadata are routed towards a designed rendezvous broker. This broker also require complete knowledge of all ads and subs and can therefore compute matches for every producer/consumer. This approach is beneficial if used in conjunction with the rendezvous based forwarding approach for ads/subs.

6 Use cases for SDN-like pub/sub

We now present a sample of application scenarios that motivate our design. Most of the use cases presented are already established in the pub/sub literature: we demonstrate how our design facilitate or enhance such applications. We also present novel uses of pub/sub possible with our model.

6.1 Big Data analytics

Support for event-based solutions have been rising in the area of Big Data analytics. Complex event processing systems are cited as the enabler for realtime analytics, with industrial support from companies such as Twitter (Storm [22]). Metrics data collected at real-time are considered first citizens and processed to manipulate business processes. For instance, pub/sub can be used for dynamic service composition in service-oriented architecture [16], with each service provider modelling their specifications and interactions as subscriptions and publications respectively. In certain cases however, the business logic spans across services and requires the use of coordinators to enforce global safety constraints [24]. In this case, the function of these coordinators is provided by our control nodes.

6.2 Application performance management

Another example can be found in the APM use case. Here, metrics data is collected to monitor the health of the system. For instance, an intrusion detection system operates by filtering over a stream of incoming events and dispatching alerts over anomalies. This can be supported using the content-based pub/sub model to obtain the precise substream of outlying events. However, the monitoring subscription might be frequently changing as it tunes itself to statistical values of the metrics attributes, which requires parametric subscriptions [14]. Such type of subscriptions are easily accomodated by our interest manager abstraction, which can fetch the required values (eg. via a key-value store) and install the appropriate the subscriptions at runtime.

6.3 Client notification for long-lived objects

Publish/subscribe can be used to notify or disseminate updates about a particular object. This is the case for Google, which developed its own pub/subbased notification system for its web services [1]. Another example includes massively

multiplayer online games, where players obtain a partial view of the game shown through its respective client. Pub/Sub subscriptions are used to model the interest of clients vis-à-vis objects found in the game [3]. Players can subscribe to game objects based on location (eg. nearby objects), social relationship (eg. friends), and other attributes (eg. ranking of the player). The amount of game state a player needs to formulate accurate subscriptions can be prohibitely large that it can defeat the purpose of employing interest management techniques in the first place. To alleviate this problem, we can employ our interest managers to collect larger amount of data about the game and compute precise subscriptions for the players, thus leveraging interest management techniques more efficiently.

7 Conclusion

We presented *SDN-like*, a new pub/sub model which borrows properties from software-defined networking. Our design for a centralized controller is used for bootstrapping purposes and topology control. We then extended pub/sub decoupling along the control/data plane and presented its benefits through example use cases. Our reference architecture shows how SDN-like can be implemented in a modular manner on top of an existing pub/sub engine.

While our work is inspired by SDN, the model itself can be realized without it. However, our future work will seek to implement to use an SDN-enabled underlying network (eg. using OpenFlow) to drive our SDN-like design in an efficient manner.

References

1. Adya, A., Cooper, G., Myers, D., Piatek, M.: Thialfi: A client notification service for internet-scale applications. In: Proc. 23rd ACM Symposium on Operating Systems Principles (SOSP). pp. 129–142 (2011), http://sigops.org/sosp/sosp11/proceedings/2011-Cascais/printable/10-adya.pdf
2. Aguilera, M.K., Strom, R.E., Sturman, D.C., Astley, M., Chandra, T.D.: Matching events in a content-based subscription system. In: Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing. pp. 53–61. PODC '99, ACM, New York, NY, USA (1999), http://doi.acm.org/10.1145/301308.301326
3. Boulanger, J.S., Kienzle, J., Verbrugge, C.: Comparing interest management algorithms for massively multiplayer games. In: ACM SIGCOMM NetGames Workshop (2006)
4. Carzaniga, A., Rosenblum, D.S., Wolf, A.L.: Design and evaluation of a wide-area event notification service. TOCS (2001)
5. Chen, C., Jacobsen, H.A., Vitenberg, R.: Divide and conquer algorithms for publish/subscribe overlay design. In: Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems. pp. 622–633. ICDCS '10, IEEE Computer Society, Washington, DC, USA (2010), http://dx.doi.org/10.1109/ICDCS.2010.87

6. Chen, C., Jacobsen, H.A., Vitenberg, R.: Reinforce your overlay with shadows: Efficient dynamic maintenance of robust low fan-out overlays for topic-based publish/subscribe under churn. Tech. rep., University of Toronto, University of Oslo (2012), http://msrg.org/papers/TRCJV-DynShadow
7. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.: The many faces of publish/subscribe. ACM Comput. Surv. 35(2), 114–131 (Jun 2003)
8. Fidler, E., Jacobsen, H.A., Li, G., Mankovski, S.: The PADRES distributed publish/subscribe system. In: ICFI (2005)
9. Floodlight OpenFlow Controller: http://floodlight.openflowhub.org/
10. Frischbier, S., Margara, A., Freudenreich, T., Eugster, P., Eyers, D., Pietzuch, P.: ASIA: application-specific integrated aggregation for publish/subscribe middleware. In: Proceedings of the Posters and Demo Track. pp. 6:1–6:2. Middleware '12, ACM, New York, NY, USA (2012)
11. Google: SDN based inter-datacenter WAN using openflow. White paper (2012)
12. Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., Shenker, S.: Nox: towards an operating system for networks. SIGCOMM Comput. Commun. Rev. 38(3), 105–110 (Jul 2008), http://doi.acm.org/10.1145/1384609.1384625
13. Hu, S., Muthusamy, V., Li, G., Jacobsen, H.A.: Transactional mobility in distributed content-based publish/subscribe systems. In: Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems. pp. 101–110. ICDCS '09, IEEE Computer Society, Washington, DC, USA (2009), http://dx.doi.org/10.1109/ICDCS.2009.73
14. Jayaram, K.R., Jayalath, C., Eugster, P.: Parametric subscriptions for content-based publish/subscribe networks. In: Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware. pp. 128–147. Middleware '10, Springer-Verlag, Berlin, Heidelberg (2010), http://dl.acm.org/citation.cfm?id=2023718.2023728
15. Koldehofe, B., Dürr, F., Tariq, M.A., Rothermel, K.: The power of software-defined networking: line-rate content-based routing using openflow. In: Proceedings of the 7th Workshop on Middleware for Next Generation Internet Computing. pp. 3:1–3:6. MW4NG '12 (2012)
16. Lee, J.G., Whang, K.Y., Han, W.S., Song, I.Y.: The dynamic predicate: integrating access control with query processing in xml databases. The VLDB Journal 16(3), 371–387 (Jul 2007), http://dx.doi.org/10.1007/s00778-006-0037-7
17. Li, G., Muthusamy, V., Jacobsen, H.A.: Adaptive content-based routing in general overlay topologies. In: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware. pp. 1–21. Middleware '08, Springer-Verlag New York, Inc., New York, NY, USA (2008), http://dl.acm.org/citation.cfm?id=1496950.1496952
18. Liu, H., Jacobsen, H.A.: Modeling uncertainties in publish/subscribe systems. In: Proceedings of the 20th International Conference on Data Engineering. pp. 510–. ICDE '04, IEEE Computer Society, Washington, DC, USA (2004), http://dl.acm.org/citation.cfm?id=977401.978095
19. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J.: Openflow: enabling innovation in campus networks. SIGCOMM Comput. Commun. Rev. 38(2), 69–74 (Mar 2008), http://doi.acm.org/10.1145/1355734.1355746
20. O. M. E. Committee: Software-defined networking: The new norm for networks. Open Networking Foundation (2012)
21. Pietzuch, P.R., Bacon, J.: Hermes: A distributed event-based middleware architecture. In: ICDCS (2002)

22. Storm, distributed and fault-tolerant realtime computation: http://storm-project.net/
23. Tootoonchian, A., Ganjali, Y.: Hyperflow: a distributed control plane for openflow. In: Proceedings of the 2010 internet network management conference on Research on enterprise networking. pp. 3–3. INM/WREN'10, USENIX Association, Berkeley, CA, USA (2010), http://dl.acm.org/citation.cfm?id=1863133.1863136
24. Yoon, Y., Ye, C., Jacobsen, H.A.: A distributed framework for reliable and efficient service choreographies. In: Proceedings of the 20th international conference on World wide web. pp. 785–794. WWW '11, ACM, New York, NY, USA (2011), http://doi.acm.org/10.1145/1963405.1963515
25. Zhang, K., Sadoghi, M., Muthusamy, V., Jacobsen, H.A.: Distributed ranked data dissemination in social networks. In: Proceedings of the 33rd IEEE International Conference on Distributed Computing Systems. ICDCS '13, Philadelphia, PA, USA (2013)