applied sciences

MDPI

Article

SDNPS: A Load-Balanced Topic-Based Publish/Subscribe System in Software-Defined Networking

Yali Wang [1,2], Yang Zhang [1,*] and Junliang Chen [1]

[1] State Key Laboratory of Networking and Switching technology, Beijing University of Posts and Telecommunications, Beijing 100876, China; yaliwan_g@163.com (Y.W.); chjl@bupt.edu.cn (J.C.)
[2] College of Computer and Information Engineering, Henan Normal University, Xinxiang 453007, China
* Correspondence: yangzhang@bupt.edu.cn; Tel.: +86-1061198020

Academic Editor: Christos Verikoukis
Received: 31 December 2015; Accepted: 14 March 2016; Published: 24 March 2016

Abstract: Publish/subscribe systems on the traditional Internet suffer from poor scalability and high delay in the face of the Internet of Things (IoT) environment. Being customizable, the paradigm of software-defined networking (SDN) provides a chance to establish an IoT-specific network. In this paper, we propose an SDN-based publish/subscribe system named SDNPS, which can construct and fine-tune topic-connected overlays for the sake of disseminating events efficiently and non-redundantly based on a global topology overview. It organizes topics as a Huffman-like topic tree and codes them into binary strings so that filtering and forwarding events can be operated directly on SDN-configurable switches, which helps to reduce end-to-end latency. This hierarchical organization form of topic tree makes it possible to incrementally construct and store overlays, which contribute to reducing the time and space complexity of routing computation. More specifically, it achieves a better tradeoff between load-balancing of the overall optimization objective and the minimal forwarding cost of per-topic overlay.

Keywords: publish/subscribe system; load balancing; overlay network; software-defined networking

1. Introduction

The coming of the Internet of Things (IoT) brings about more and more various smart services. Occurring with a massive stream of data, these services are typically viewed as event-driven. Generally speaking, these massive amounts of data from IoT have the following characteristics: (1) predictability, which means these data show cyclical fluctuations as massive sensors send sensing data periodically; (2) asynchronous and multicasting-suitable, which means the similar nature of events, which may have multi-senders and multi-receivers; even receivers are concerned with the what in a communication instead of the specific who sending events; similarly, multi-senders do not care about who will receive data; (3) time-sensitive, which means most of the data need to be delivered within a prescribed time, *etc.*

Therefore, how to deliver such complex multi-source sensor data in real time efficiently is a vital problem [1,2]. Fortunately, the publish/subscribe paradigm can exactly facilitate this data dissemination mode instead of using traditional request-reply messaging. A publish/subscribe system is a universal many-to-many communication paradigm [3], especially for those applications with loosely-coupled entities. In a topic-based publish/subscribe system, events are published with specific identifiers called topics. The publish/subscribe paradigm then guarantees disseminating every new event to those subscribers who have expressed their interest in the topic similar to the event [4,5].

However, under the traditional network architecture, due to the lack of global traffic information, publish/subscribe systems suffer from insufficient utilization of the physical network infrastructure, such as traffic imbalance on different links. In such a largely distributed IoT environment, the emerging mass data mega-trends also worsen the traffic imbalance phenomenon. To address this concern, some methods, including static load balancing (preplanned allocation bandwidth) and dynamic load balancing (dynamic allocation bandwidth during run-time), were proposed [6–8]. However, this problem is not solved, essentially owing to the global knowledge insufficiency under the distributed management of the traditional Internet. On the one hand, the filtering operation, which mainly saves bandwidth in a publish/subscribe system, is performed by specific components called brokers. This imposes a significant delay by lengthening the end-to-end path with a detour to the brokers and a processing delay for matching events against filters' rules. On the other hand, the best-effort service provided by the traditional Internet cannot meet delay-sensitive requirements.

Recently, software-defined networking (SDN) [9,10] emerged as a new-style network architecture that decouples the control plane from the forwarding plane. Maintaining a network-wide view of up-to-date datapath elements and link state information, SDN makes it possible to enable on-demand resource allocation, self-service provisioning and network virtualization. OpenFlow as the actual standard for SDN enables many filtering operations to match received packets efficiently through specifying the interface to install and modify flows directly on SDN-configurable switches. Therefore, it is possible to flexibly control the forwarding plane, which is conducive to outperforming the traditional Internet in the overall performance of the network, as well as assuring the desired performance for diverse applications, by adopting a traffic management mechanism [11].

In this paper, we propose and construct a topic-based publish/subscribe system in an SDN environment named SDNPS. The implemented prototype deploys traffic engineering by making full use of the centralized control nature under SDN. Firstly, we obtain the global overview of the topology by abstracting and aggregating the network link state. Then, we predict the traffic distribution for some time to come in terms of the predictability of the mass data from IoT. Finally, we calculate minimum overlay networks per topic and extract multicast routing paths in light of the well-known shortest path algorithm. Our design makes a better tradeoff between global load balancing of links and the minimum cost forwarding for per-topic events.

The contributions of this paper are summed up in the following.

- In SDNPS, we organize all of the topics into a topic tree in terms of their natural language semantics. On account of the subscription coverage relationship among topics at different levels of the topic tree, we present a topic-connected overlay constructing algorithm, which solves the specified multicasting problem among publishers and subscribers. This algorithm achieves minimum forwarding costs for every topic, as well as load balancing for the whole network. By means of incrementally generating and storing overlays, the time complexity and the space complexity of this algorithm are both significantly reduced.
- SDNPS can filter and forward events directly on SDN-configurable switches with the aid of dexterously mapping topics to binary identifiers and embedding these identifiers into packet headers as matching fields. This helps to reduce end-to-end latency. We devise traffic engineering in SDNPS, which performs routing computation based on traffic prediction. It also dynamically adjusts routing in the presence of physical topology changing induced by traffic bursting and link/switch failures, as well as subscription topology changing brought by new (un)subscription events.

The remainder of this paper is structured as follows. In Section 2, the related work is reviewed and discussed. Section 3 presents the system architecture; other corresponding aspects, including topic management, topology management and strategy management, are also described. Section 4 presents the problem statement and model for the minimal cost topic-connected overlay problem (in short MCTCO), then a heuristic algorithm for this problem is described in detail. Section 5 exhibits the performance evaluation, and conclusion remarks are given in Section 6.

2. Related Work

In general, the related work can be classified into two categories: (1) routing optimization of topic-based publish/subscribe systems; (2) SDN-based publish/subscribe systems.

2.1. Routing Optimization of Topic-Based Publish/Subscribe Systems

There have been several famous topic-based publish/subscribe systems, e.g., SCRIBE [9], Bayeux [10], TERA [12], Corona [13] and NICE [14], *etc*. Nevertheless, early research work mostly focused on the realization of distributed systems; the work for the routing optimization problem is scanty.

Chockler *et al.* first presented the theoretical problem for a minimum topic-connected overlay network (in short, Min-TCO) [15] in the conference of the Association for Computing Machinery (ACM) symposium on the principles of distributed computing (PODC) in 2007. The optimization objective of Min-Tco was to minimize nodes' average degree. Additionally, they proposed a greedy merge algorithm called GM to solve this problem. Chen *et al*. also tried to address the issue of minimum average degree topic-connected overlay join problem (MinAvg-Tco-Join) [16] and proposed a divide and conquer algorithm aiming at this problem. The difference is that Chockler pursued the total optimal routing cost, while Chen was committed to efficiently constructing the overlay when nodes dynamically join two or more topic-connected overlays. The latter did to some extent in subsequent work by devising algorithms for a topic-connected overlay design, which sought a balance between time efficiency and the number of edges [17].

Onus *et al.* tended to minimize the maximum degree of topic-connected overlay (MinMax-Tco) [18], and presented the low degree topic-connected overlay problem [19]. They proved that the two problems are both NP-complete.

Darugar *et al*. proposed a web services network architecture and integrated publish/subscribe system, which presents a topic as a routing entity, including the functionality for topic creation, subscription and publication in accordance with the basic modes of web services [20]. This architecture can be implemented across any suitable computer network. However, how to deploy it on SDN is not discussed in this patent.

There were other systems that did not attempt to minimize the average degree or maximum degree of the overlay. In [9,12], a separate overlay, such as a multicast tree, was maintained for each topic through a distributed protocol. However, the average degree would be roughly twice the average subscription size.

Other systems, like SIENA [21], turned to reducing the number of non-interest relay nodes, while relaxing topic-connectivity, which can potentially decrease the overlay degree. They did not dispose of the tradeoff between the extra overhead incurred by forwarding unwanted events and the overlay degree.

2.2. SDN-Based Publish/Subscribe Systems

The current network architecture has seriously affected the network expansion because of its insufficiency to cater to the massive proliferation of services and users. As an evolution of programming networks, SDN has successfully attracted momentous attention from both academia and industry. On the academic side, the OpenFlow Network Research Center [22] has been created. Some work on standardization for SDN at the Internet Engineering Task Force(IETF), the Internet Research Task Force (IRTF) and other organizations was also developed. On the industry side, the Open Network Foundation was created for promoting SDN and standardizing the OpenFlow protocol.

The work in [23] first studied the impact of SDN on the design of future message-oriented middleware, such as publish/subscribe systems. This paper pointed out that publish/subscribe systems could adopt a logically centralized controller model w.r.t. maintenance, monitoring and

control for the overlays. The authors presented a new publish/subscribe model, which is SDN-like with some properties borrowed from SDN.

The Google Corporation deployed a private WAN called B4, which adopted a software-defined networking architecture [24] to connect Google's data centers across the planet. Through the mechanism of centralized traffic engineering service, B4 achieved nearly 100% link utilization.

Koldehofe *et al.* proposed a reference architecture for building middleware, which befitted future Internet applications, accompanying a solution for realizing content-based routing at the line-rate relying on this architecture [25]. This architecture had some reference significance for us, even if it was for a content-based publish/subscribe system and there were no experimental data to support it.

Jokela *et al*. implemented a multicast forwarding fabric called LIPSIN, which was suitable for large-scale topic-based publish/subscribe systems [26]. By placing a Bloom filter into data packets, LIPSIN achieved efficient multicasting on the network layer. The work in [27] proposed and evaluated a content-based publish/subscribe middleware in SDN, which achieved high performance in forwarding speed and bandwidth management. The work in [28] proposed a methodology for both vertical and horizontal scaling of the distributed control plane as an expansion to [27].

Hakiri *et al*. proposed a data-centric publish/subscribe paradigm for proactive overlay SDNs and presented a solution for realizing an SDN controller that operated in a distributed manner [29]. Vilalta *et al.* proposed an end-to-end orchestration for IoT services using an SDN/NFV-enabled edge node under SDN [30].

Syrivelis *et al*. proposed a particular architectural context for ICNconcerning how SDN and ICN could concretely be combined and outlined a possible realization in a novel design for ICN solutions [31]. They also pointed out several possible testbed deployments. However, they did not implement it, and there was no experimental verification as a consequence.

3. Architecture

This section delves into the overall architecture of SDNPS. Learning from the traditional Internet, we adopt hierarchical thinking to organize brokers and manage routing. All of the participants are partitioned into multiple clusters according to their regional characteristics (belonging to the same data center or LAN). Each cluster is a logically-autonomous area containing a representative broker and some agent brokers, as well as a certain number of SDN-configurable switches, communicating with other clusters through border switches. All of the clusters are treated equivalently. Above this, we adopt a global server to manage topology and to compute routing holistically. It is important to note that we devise an alternative distributed schema (it is still under testing) in order to avoid single point failure and heavy burden on the server. As depicted in Figure 1, the system is logically structured in three layers: the switch hardware layer, the cluster controller layer and the global management layer. The switch hardware layer does not run complex control protocols. Its primary task is filtering and forwarding events. The controller layer runs both the OpenFlow protocol and network control applications. A controller is meanwhile a representative broker of the cluster where it is located. The global management layer consists of a major server and a standby server, which have global information about the physical topology and subscription topology. The servers enable central traffic engineering by constructing topic-connected overlay networks based on global traffic prediction and link information collection.

As depicted in Figure 2, we devise a strategy-driven topic tree aggregation routing schema. From the subscription list and topic tree, we extract the subscription topology in incremental storage mode. From link state database(LSDB) and cluster information, we extract the cluster-level physical topology. Then, we compute multicast routing paths per topic based on the subscription topology and cluster-level physical topology, as well as the strategy base in our schema.

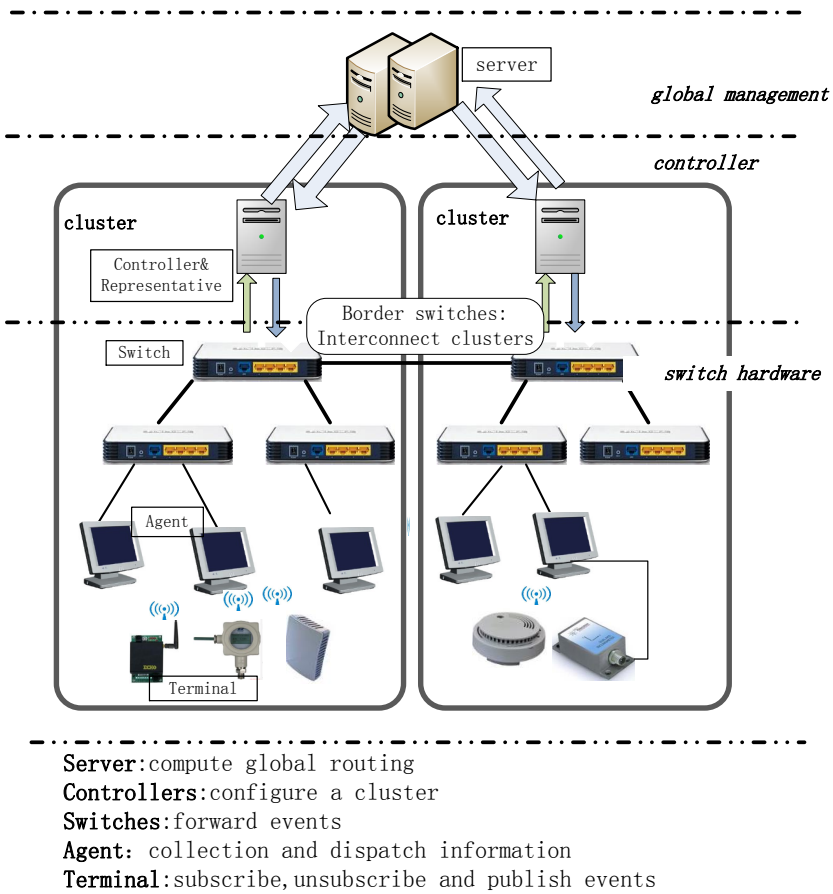

Figure 1. System architecture overview.

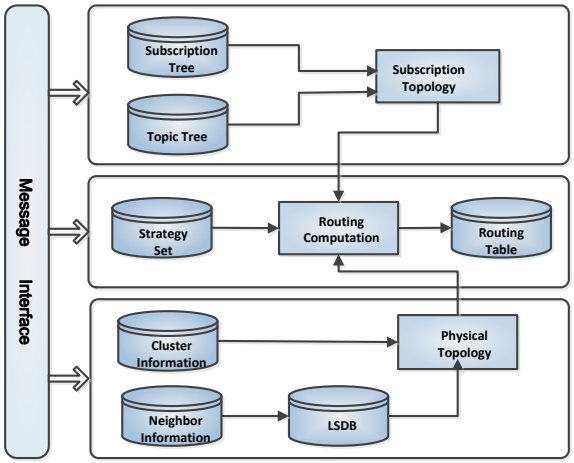

Figure 2. Topology/routing management.

3.1. Topology Management

SDNPS maintains two kinds of topologies: subscription topology and physical topology.

For the former, every cluster (to be more exact, the representative broker) has global subscription information for all topics, which constitutes to the subscription topology. Each new (un)subscription event should be broadcast to all of the clusters, so that every cluster can receive it and then update its subscription topology. The representative broker of the cluster, which originally generates

(un)subscription events, also answers for forwarding this (un)subscription information to the server. This is the subscription topology.

For the latter, on the one hand, a controller detects the physical link states of its own cluster, which forms an intra-cluster topology; on the other hand, it maintains the reachability information to the neighbor clusters they selected by periodically sending out heartbeat information, then applying network virtualization [32] technology to abstract the physical network to the virtual network with tuples (source cluster, destination cluster, and bandwidth), which forms the cluster-level topology. When centralized routing computation is adopted, every controller delivers its cluster-level topology formatted with JSON (JavaScript Object Notation) to the server. Then, the server forms an inter-cluster topology by aggregating and abstracting these data and computing inter-cluster routing paths. When distributed routing computation is adopted, every controller maintains this inter-cluster topology through a distributed protocol and computes inter-cluster routing paths, respectively. In this paper, we adopt the centralized routing computation for brevity. In subsequent work, we will adopt distributed computation for better flexibility and reliability. This is the physical topology.

Each update of the cluster topology or subscription topology would trigger a routing recalculation, so as to add or delete paths among relevant switches.

3.2. Topic Management

In topic-based publish/subscribe paradigm, topics contact publishers and subscribers. However, topics are not arbitrary strings in actual systems. To prevent from topic bursting, topics must be registered to the server before they are put into use. In SDNPS, topics are structured as a topic tree in which a topic covers all of the topics in its descendant nodes. That is to say, if a subscriber subscribes to a topic, then it also automatically subscribes to those topics on its subtrees. Due to the subscription cover relationship between father topic and child topics in the topic tree, we devise an incremental minimal cost topic-connected overlay algorithm, which significantly reduces the time and space complexity.

Now that SDN switches can perform filtering operations at low latency [27], we can manipulate flow matching to support this work. The topic tree is an arbitrary multi-bifurcated tree. Transforming the topic tree into a binary tree and then coding this binary tree by the left branch with zero and the right branch with one, we get a binary string named the topic-expression (tp) for each topic. Figure 3 exhibits the methods of encoding a topic tree in this paper.

In particular, all of the events matching the topic tree constitute the event space called Ω, which is a one-dimensional space. Ω can be divided into subspaces according to topics. Any subspace can be identified by a topic-expression. In order to define the subscription cover relation of topics, two functions are defined: $dlsuffix(tp)$ is used for deleting all of the rear "1" of tp; $dlzero(tp)$ deletes the rear "0" of tp. If tp_i equals $dlzero(dlsuffix(tp_j))$, we say topic tp_i is the father of topic tp_j. It is obvious that topic-expression has a partial relation. It fulfills the following characteristics: (1) a subspace tp_i covers the subspace tp_j, iff tp_i is a prefix of $dlzero(dlsuffix(tp_j))$, written $tp_i \succ tp_j$; (2) if $tp_i \succ tp_j$, the subscriber set S_j of tp_j is a subset of the subscriber set S_i of tp_i, written $S_j \subseteq S_i$.

For a topic tp_i, there are three kinds of events: publish events, subscribe events and unsubscribe events. Subscribe and unsubscribe events should be broadcast to all of the nodes. Therefore, flooding is an appropriate method. Publish events should be routed to those subscribers who have subscribed to them beforehand. SDN-configurable switches must identify these three types of events so that they can perform different processing. For this purpose, we encode event type simultaneously. Two-bit binary coding meets this requirement. This two-bit type code for events is stitched with tp_i, forming the identifier (called *typetp*) of a topic tp_i. When an event is generated, the *typetp* is enclosed in its packet header. It is noteworthy that the subspace relationship mentioned above allows the events whose topics have the cover relation to share a common subpath. Recall that the routing of a publish/subscribe system is an obvious multicast problem, so we adopt the IPV6 multicast address to embed *typetp*. Figure 3 describes this process.

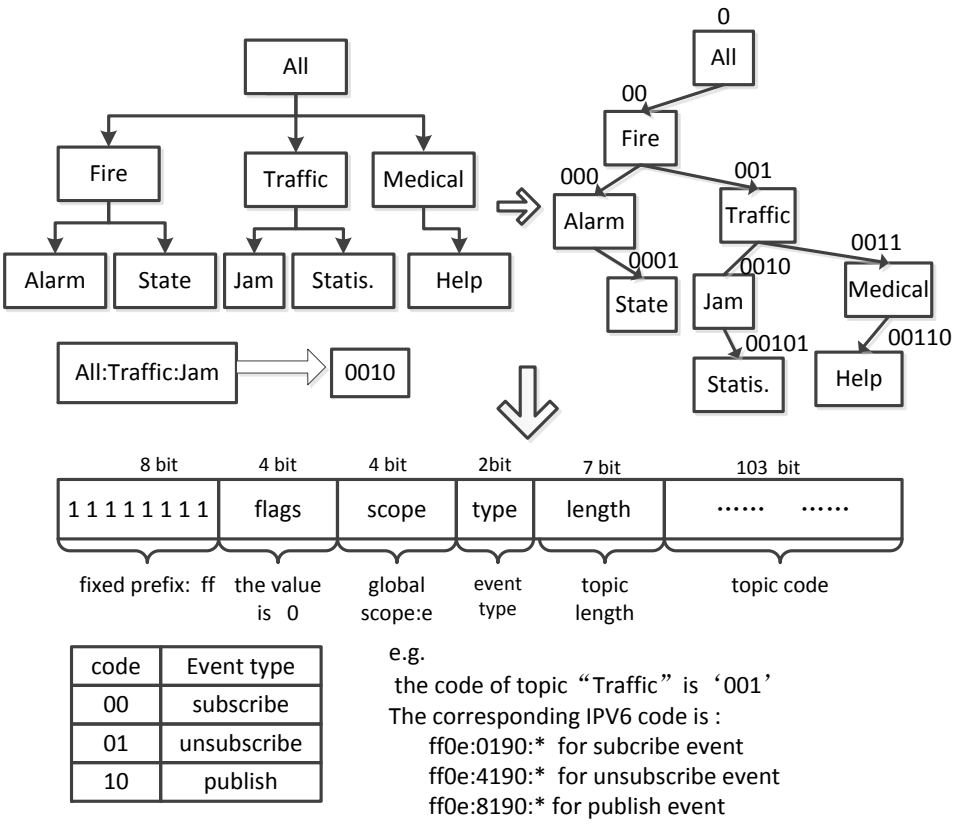

Figure 3. Topic identifier.

The length of the IPV6 address is 128 bits. According to Figure 3, except for the frontal 25 bits, which are occupied by a fixed section, there are 103 bits for topic codes. Theoretically, the code space is 2^{103}. This means that there are at most 2^{103} topics, and it is adequate for our system. In fact, the excessive number of topics, which is associated with the items of flow entries in flow tables, may increase the workload for flow table lookup and degrade the forwarding performance at SDN-configurable switches. Therefore, the flow table entries' aggregation is indispensable to address this problem. This can be easily implemented in light of the hierarchy of the topic tree.

3.3. Strategy Management

Sometimes, some clusters may prohibit events with specific topics from passing through for security or privacy reasons. We adopt a strategy to define these constraints. The introduction of a strategy makes it possible to distinguish different powers among clusters, which contributes to preventing the system from some unfavorable conditions, such as information leakage, network traffic anomalies, *etc*. The organization of the strategy is also in accord with the topics in the topic tree. The strategy information is formatted as an XML file, and any cluster can download and store this file. The administrator is responsible for the configuration of the strategy, including addition, deletion and modification of the strategy. When a new strategy message comes, the strategy module will merge it with the existing strategy data. If there is an update to the strategy data, this will be reflected in the subscribe list, and the corresponding routing computation will be triggered.

The strategy is usually implemented in the forwarding plane by setting the filter conditions. However, even if efficient filtration technology, such as a Bloom filter, is used, this still causes a delay. Nevertheless, we innovatively adopt another method in SDNPS. We impose a strategy on the routing schema in the period of routing computation. When computing the routing path for topic t, relative strategy constraints will be considered. If the strategy is cluster level, the routing path should not include clusters constrained by it. However, if the strategy is broker level, routing computation will

not be affected; those brokers constrained by the strategy autonomously decide whether to receive this type of event or not.

3.4. Traffic Engineering Architecture

We design traffic engineering for our system to achieve approximate optimal performance. All of the functions are deployed on the server and controllers. The server accounts for seeking optimized routing paths among clusters while controllers split flows among multiple paths to balance traffic. Figure 4 shows an overview of the traffic engineering architecture.

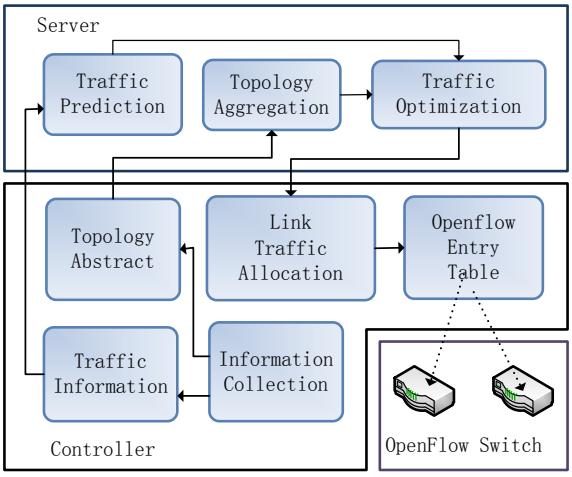

Figure 4. Traffic engineering overview.

The controllers operate over the following aspects:

(1) The information collection module provides information for traffic prediction. It is responsible for gathering statistics information, including packet size and the number of every flow, the queue size for every port of the switch, *etc.*
(2) The link traffic allocation module allocates traffic evenly among multiple links between any two connected clusters. Besides this, it also answers for installing and updating flow tables.
(3) The topology abstract module detects switch connection information when the system starts and abstracts away topology information. When there is a link state change at run-time, it reports this information to the topology aggregation module and traffic prediction module. We call it link-level abstraction, which lays the basis for the link traffic allocation module in the controllers and the traffic aggregation module in the server.

The server achieves the following functions:

(1) The topology aggregation module chalks up the network topology graph in which vertices represent clusters and edges represent the links between clusters by consolidating topology information from multiple controllers and then aggregating trunks between clusters. We call it cluster-level abstraction, which is the basis of inter-cluster routing computation. This abstraction significantly reduces the size of the graph, thus simplifying the routing optimization algorithm.
(2) The traffic prediction module is responsible for predicting future traffic for a period of time in terms of traffic statistics information of the past time. Treating prediction values as inputs to the optimization algorithm can gain more reasonable routing congestion avoidance. The reason why traffic can be predicted is mainly due to the characteristics of data in the Internet of things (IoT) environment.
(3) The traffic optimization algorithm computes optimal or suboptimal routing paths according to the global network topology and traffic value,s which are obtained from the traffic prediction

module and the topology aggregation module mentioned above. This module will be described in detail in the remainder of this paper.

3.5. Event Routing

The routing of SDNPS is divided into two levels: routing among clusters, also known as inter-cluster routing, and routing in a cluster, also known as intra-cluster routing. Because the server has holistic information about publish information, subscribe information and a global topology, it takes charge of computing inter-cluster routing. Similarly, the controllers are responsible for intra-cluster routing.

3.5.1. Inter-Cluster Routing

The topology abstract module, which is run on the controller, has the topology information of the cluster in which it is located. Naturally, it can also perceive its adjacent clusters through the link information of border switches. We map the cluster-level routing table into the link-level routing table by allocating the traffic evenly among relevant links and then write it into flow tables of switches.

When a border switch receives an event, it just searches for its flow tables. If there is a matched item, it processes the event in terms of the corresponding instructions. If a table-miss event occurs, it forwards the packet or its header to the controller in order to get further instructions. Substantially, the routing is accomplished through flow tables. The installation of the flow tables is essential for efficient routing. The controllers are responsible for installing flow tables of the switches attached to them in terms of global routing computation (we call it inter-cluster routing), which is completed by the server. The inter-cluster routing schema is substantially a multicast routing on the experience of "link matching" [33] and multi-stream multi-source multicasting routing [34].

3.5.2. Intra-Cluster Routing

By abstracting clusters into nodes, we get a cluster-level topology, which lays the foundation for the inter-cluster routing mentioned above. However, every cluster may have multiple border switches to connect with other adjacent clusters, so the routing table obtained from inter-cluster routing must be mapped into flow entries on switches.

When a broker receives an event published by a sensor or other terminal device, which is connected to it, it firstly identifies the type and the topic (t) of the event, then searches the topic tree for its binary code. Following this, it multicasts this event using UDP by encapsulating the type and topic code into the packet header as the IPV6 address. This IP address is the IPV6 multicast address for topic t. Finally, it delivers this event to the representative broker by reliable transmission, called TCP, aiming at ensuring that every event can be transferred to the representative broker. When a broker receives an event from another broker or switch, it checks its subscription list to decide whether to receive it or not.

4. Traffic Optimization Algorithm

4.1. Problem Definition

The publish/subscribe system is represented by an undirected graph $G(V, E)$ with n nodes and m edges, where V is a set of nodes (clusters) and E is a set of links, respectively. Assume that $|V| = n$, $|E| = m$. For each link $e = (i, j)$, a non-negative parameter named the bandwidth capacity $c(e)$ is associated with it.

Postulate that there are in total $|T|$ topics in SDNPS; $|PubInt_t|$ events for topic t, which may be originated from a set of publishers (source nodes), need to be disseminated to all subscribers denoted by $SubInt_t$ (destination nodes), who have registered their interests in topic t beforehand, where $PubInt_t \subseteq$ V and $SubInt_t \subseteq$ V. Note that $|PubInt_t| \geq 1$, $|SubInt_t| \geq 0$ and $1 \leq t \leq |T|$. For events tagged with topic t, there are $|PubInt_t|$ publishers; each of them has a different publish probability.

Let $Pubpr_t(i)(1 \leq i \leq |PubInt|)$ denote the probability of node i publishing events tagged with topic t, $Pubpr_t(i) \in [0,1]$. We denote $\delta_t = (PubInt_t, SubInt_t, Pubpr_t)$ as a multicast session for topic t.

4.2. Problem Model

First, we model the nodes's publish and subscribe interests formally: given a set of nodes V and a set of topics T, a publish interest function *PubInt*, which is defined to be a probability-valued function over domain $V \times T$, and a subscription interest function *SubInt*, which is similarly defined to be a Boolean-valued function over the same domain. That is to say, we say a node is interested in publishing an event with topic t iff *PubInt(v,t)* $\in$ (0,1), and a node is interested in receiving a topic t iff *SubInt(v,t) = true*.

Then, The problem is formally defined as follows: Given $G = (V, E)$, link bandwidth capacity c: C(E)$\rightarrow R_+$, a publish interest function *PubInt* and a subscribe interest function *SubInt* over $V \times T$, a topic $t \in T$, we define the topic-connected subgraph G_t = (V_t, E_t) of G for topic t to be a minimal support graph induced by $\delta_t = (PubInt_t, SubInt_t, Pubpr_t)$, such that the cost is minimum. It is worth noting that the 'minimum' can be measured in different ways, such as minimal cost or minimal numbers of nodes and edges. We call it 'the minimal cost topic-connected overlay' problem (MCTCO).

Definition 1 (The bottleneck bandwidth of the path)**.** *Given G = (V, E), r(e) is the available bandwidth of edge e (e ∈ E), a path path(s, d) between s and d is described by a sequence of links e(s, i), e(i, j), e(j, k), ⋯, e(u, d), each of which belongs to E. The bottleneck bandwidth of path(u, v) is given in the following:*

$$rp(path(u,v)) = \min(r(s,i), r(i,j), r(j,k), \cdots, r(u,d)) \tag{1}$$

Definition 2 (MCTCO)**.** *Given G = (V, E), a cost matrix c(E), a topic t, two interest function $PubInt_t$ and $SubInt_t$ over $V \times T$, construct a topic-connected overlay network G_t = (V_t, E_t), such that $\Sigma_{e \in E_t} C(e)$ is minimum, limited $(PubInt_t \bigcup SubInt_t) \subseteq V_t \subseteq V$, $E_t \subseteq E$.*

The following lemma goes immediately after Definition 2.

Lemma 1. *MCTCO is NP-hard.*

Proof. Consider the distinguished Steiner tree problem in graphs. Given an undirected graph G(V, E), an accident with a cost function c: $c(E) \rightarrow R_+$, and a terminal set D where $D \subseteq$ V(G), find a tree Tr in G, such that D $\subseteq V(Tr)$ and $C(E(Tr))$ is minimum. Karp *et al.* has proven that the Steiner tree problem is one of the classical *NP*-hard problems [29].

Now, consider the MCTCO problem. According to the definition, the constructed overlay network must be acyclic; that means a tree spanning all of the publish nodes (set S) and subscribe nodes (set D). There are probably some indispensable relay nodes. Note that the MCTCO is identical to the Steiner tree problem if we select one node s from S as the root node and treat $S - \{s\} \bigcup D$ as terminal nodes. Therefore, MCTCO can be reducible to the Steiner tree problem.

Hence, the MCTCO problem is *NP*-hard.

Note that the above problem considers only one topic. For a set of topics, we can construct a set of minimal topic-connected overlay subgraphs. To avoid the imbalance of link traffic, when constructing minimal topic-connected overlay subgraphs, link residual bandwidth is considered as an important factor. Following, we define the residual bandwidth of graph G.

Definition 3 (The residual bandwidth of graph G)**.** *Given G = (V, E).* $\Upsilon(e)$ *is the residual bandwidth of link e, the residual bandwidth of graph G is defined as the minimum available bandwidth for all links in E. That is,*

$$\Re(G) = min_{e \in E}(\Upsilon(e)) \tag{2}$$

With the above definitions, the load-balanced MCTCO problem can be mathematically stated as follows:

Maximize $\Re(G)$.
Subject to:

$$G_t(V_t, E_t) \subseteq G(V, T) \qquad \forall t \in T \tag{3}$$

$$|E_t| = |V_t| - 1 \qquad \forall t \in T \tag{4}$$

$$\forall e \in E \quad \xi_T(e) = \begin{cases} 1 & if \quad e \in E_T \\ 0 & if \quad e \notin E_T \end{cases} \tag{5}$$

$$\aleph(e) = \sum_{t=1}^{|T|} \sum_{i=1}^{|PubInt|} Pubpr_t(i) \cdot b \cdot \xi_T(e) \tag{6}$$

$$\aleph(E) \leq C(E) \tag{7}$$

$$\Upsilon(e) = C(E) - \aleph(e) \tag{8}$$

The objective function $\Re(G)$ measures the maximum residual bandwidth of graph G using constraint set Equation (8). Among the links of graph G, maximizing the bottleneck bandwidth of the link, which is the minimal residual bandwidth of all links, contributes to balancing traffic. When $\Re(G)$ is less than zero, the instance is infeasible. Constraint set Equations (3) and (4) ensure that every topic-connected subgraph is acyclic. Constraint Equation (5) is a decision function, which indicates whether a link e of graph G is an edge of subgraph G_T or not. Constraint Equation (6) defines the consumed bandwidth of link e in graph G; here, b is a bandwidth unit. Constraint Equation (7) restricts that the consumed bandwidth cannot go beyond the capacity of link e.

4.3. The Load-Balanced Topic-Connected Overlay Algorithm

Aiming at the objective function mentioned above, we put forward a heuristic algorithm. This algorithm is evolved from the algorithms solving the multi-stream multi-source multicasting routing problem (MMMRP) [34]. There are other incidental algorithms proposed for inter-cluster routing.

Algorithm 1 Inter-cluster routing algorithm.

Require: *G = (V, E), C(E)*
// C(E) is the bandwidth capacity matrix
Ensure: $max(minR(E))$
$//min(R(E)) = min(R_{e_{ij}})$, for $\forall i, j\ e_{ij} \in E$
//R(E) is residual bandwidth matrix, equals *C(E)* initially.
1: **for** each topic t **do**
2: $G_t = MSG(S, t))$;
//call for MCTCO algorithm
3: cb = $\sum_{i=1}^{|t.PubInt|} t.Pubpr(i) \cdot b$;
4: **for** each $e \in E(G_t)$ **do**
5: *c(e) = c(e)-cb*;
//update the residual bandwidth;
6: **end for**
7: **end for**

The main idea of inter-cluster routing algorithm (Algorithm 1) is as follows. For every topic t, we construct a topic-connected overlay subgraph G_t that spans its publish nodes and subscribe nodes (Line 2). Following, we compute the consumed bandwidth $\sum_{i=1}^{|PubInt|} Pubpr_t(i) \cdot b$ of links in G_t (Line 3) with the node's publishing probability considered. Then, we update the residual bandwidth by subtracting $\sum_{i=1}^{|PubInt|} Pubpr_t(i) \cdot b$ from c_j if e_j is in G_t (Line 5). These are repeated until all topics are processed (Lines 1–7).

Algorithm 2 Minimal cost topic-connected overlay algorithm (MCTCO).

Require: $G = (V, E)$, $PubInt$, $SubInt$, P
Ensure: G_t is a tree, $SubInt \bigcup PubInt \in G_t$, $G_t \subseteqq G$
$min(\sum c(e_{ij}))$, for $\forall i, j\ e_{ij} \in G_t$

1: **if** T.father is NULL **then**
2: $G_t = \emptyset$; modify $c(E)$ by strategy set for topic t;
3: **for** each source $s_i \in PubInt$ **do**
4: Tree T_i = WidestPTathTree (G, s_i);
5: **end for**
6: **for** each $d_j \in$ SubInt **do**
7: EnQueue (Q, d_j);
8: **end for**
9: **while** not empty (Q) **do**
10: currnode = DeQueue (Q);
11: **while** TRUE **do**
12: nextnode = the nextnode in path(currnode, S_1);
13: nextwidth = rp(path(currnode, S_1));
14: **for** each $s_i \in PubInt - \{S_1\}$ **do**
15: tempwidth=rp(path(currnode, s_i)) in T_i;
16: **if** tempwidth $>$ nextwidth **then**
17: nextnode = the nextnode in path(currnode, s_i);
18: nextwidth = tempwidth;
19: **end if**
20: **end for**
21: Add (currnode, nextnode) to G_t;
22: **if** nextnode *in* PubInt **then**
23: break;
24: **end if**
25: currnode = nextnode;
26: **end while**
27: **end while**
28: traverse graph G_t to get all the connected subgraph;
29: Add the widest edges which can connect two subgraphs to G_t until G_t is connected;
30: **else**
31: TF = T.father; $G_t = G_{t.father.topic}$;
32: TFnodes = TF.subscribelist $\bigcup$ TF.publishlist;
33: Tnodes = T.subscribelist $\bigcup$ T.publishlist;
34: Bnodes = Tnodes-TFnodes;
35: **for** each node N in Bnodes **do**
36: Add the widest path from N to V_{G_t} to G_t;
37: **end for**
38: **end if**

Algorithm 2 is constructing a topic-connected overlay network. Remember that we organize topics as a topic tree. Owing to the overlay relationship between child topic and its father topic, when

a topic is in the first layer in the topic tree, we build the overlay network completely *ab initio* for this topic; otherwise, we generate the overlay network for a topic from its father topic overlay network by adding some vertices and edges. For the first case, we firstly generate $|PubInt|$ (also denoted with $|S|$) spanning trees, which are widest (Lines 2–4). There are several algorithms proposed for this, e.g., a modified Dijkstra's algorithm or a modified Bellman–Ford algorithm, *etc*. In this paper, we adopt a modified Kruskal's algorithm, which has the same time complexity asymptotically and a faster run-time compared to the modified Dijkstra algorithm. Then, we find the widest path from each destination vertex (SubInt) to any source vertex (PubInt) (Lines 6–27); this procedure will generate several disjoint and acyclic subgraphs. Finally, we merge these subgraphs into a minimum spanning tree (also known as a minimum overlay network) (Lines 28–29). For the latter case, we first figure out the vertices that are not in the father topic's overlay network (denoted as $G_{t.father}$), but meanwhile in its own publishlist or subscribelist (Lines 31–34). Then, we add these vertices to $G_{t.father}$ to form a connected spanning tree adopting the Kruskal algorithm (Lines 35–37).

Algorithm 3 depicts the routing fine-tuning operations for physical topology change. In fact, any changes on the subscription topology or physical topology would trigger a routing update. For the first case, when a node v adds to the subscribelist for topic t, the routing update algorithm first checks the overlay G_t; if $v \notin G_t$, it finds the shortest and widest path from v to G_t and adds this path to G_t. On the contrary, if a node v unsubscribes from topic t, it first checks whether v is a leaf node in G_t or not. If v is a leaf node, it will remove node v, as well as the edge attached to v. Otherwise, v is a relay node, and there is nothing to do. There is no difficulty in performing this update, so this algorithm is omitted in this paper. For the latter, as any link can suffer from a failure, the traffic flowing through the invalid link originally should take detours via other links. When a new link comes into operation, it should share the traffic of other links to balance the load as much as possible. To avoid traffic vibrating, this process moves mildly. It just depend on subsequent adjustment. Algorithm 3 carries out routing fine-tuning when the link state changes. Note that the routing update induced by node failure is included in Algorithm 3, because this situation equals multiple links' failure.

Algorithm 3 Routing fine-tuning algorithm for physical topology change.

Require: $G = (V, E)C(E), L$
$//C(E)$ is the residual bandwidth capacity matrix
$//G_t$ is overlay network for topic t
Ensure: $max(minR(E))$
$//min(R(E)) = min(R_{e_{ij}})$,for $\forall i, j\ e_{ij} \in E$
//R(E) is residual bandwidth matrix, equals C(E) initially.
1: **if** the links set is invalid
2: find the affected overlay network set GA;
3: **foreach** G_t in GA
4: merge the connected subgraphs split by failed links
5: **endfor**
6: **else** //there is newly added link
7: update the residual bandwidth matrix
8: **endif**

4.4. Complexity Analysis for Algorithm 2

Lemma 2. *The time complexity of Algorithm 2 is O(max($n \cdot \log m$, $n \cdot |PubInt| \cdot |SubInt|$)).*

Proof. The time complexity of WidestPathTree(G, s_i) is O($n \cdot \log m$). Here, n is the number of vertices and m is the number of edges. This procedure is executed $|PubInt|$ times. Therefore, the time complexity of Lines 2–4 is O($n \cdot |PubInt| \cdot \log m$).

The time complexity from Lines 5–27 in Algorithm 2 is O($n \cdot |PubInt| \cdot |SubInt|$). Here, $|PubInt| \leq n$, $|SubInt| \leq n$.

Lines 28–29 adopt Prim's algorithm to merge several connected components into a connected graph; the time complexity of the worst case is O(n^2). If the Fibonacci heap and adjacency list are used for storing edges and weights, the time complexity can be reduced to O($m + n \cdot log(n)$).

Therefore, the time complexity of constructing a topic-connected overlay network *ab initio* is the sum of these three items discussed above, that is O(max($n \cdot \log m$, $n \cdot |PubInt| \cdot |SubInt|$)).

The function of Lines 31–36 equals Line 29; the same algorithm can be adopted. Therefore, the time complexity of constructing a topic-connected overlay network based on its father topic overlay network is O($m + n \cdot log(n)$).

Therefore, the total time complexity of Algorithm 2 is O(max($n \cdot \log m$, $n \cdot |PubInt| \cdot |SubInt|$)).

5. Performance Evaluation

This section is dedicated to an analysis for the performance evaluation of the proposed SDNPS. A series of experiments are conducted for evaluating the proposed algorithms.

5.1. Performance Evaluation of SDNPS

The SDNPS has been evaluated on a simple SDN testbed consisting of commodity PC hardware and virtualization technologies. Figure 5 presents the experimental tandem topology with three hops. All link rates are 10 Mb/s, and the packet lengths are 200 bytes. The elements are virtualized by three IBM servers with an eight-core CPU and 16 G of memory. We would like to stress that using virtual machines does validate our schema, but gives very conservative performance bounds. We test the performance of SDNPS from the following aspects.

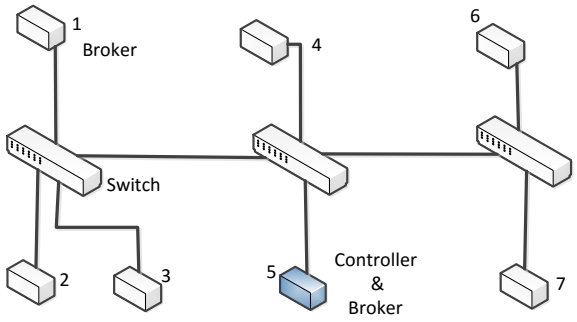

Figure 5. Experiment topology.

5.1.1. Delay and Loss Rate for 1:1 Transmission

This set of experiments is devised to study the end-to-end delay characteristics of the SDNPS on the aforementioned testbed. we first analyze end-to-end delay between a pair of publisher and subscriber connected via a one-hop path (from Node 1 to Node 2), two-hop path (from Node 1 to Node 4) and three-hop path (from Node 1 to Node 6) in the topology, as depicted in Figure 5. For every test, 100,000 UDP packets are sent continuously from a publisher to a subscriber. We recorded the time period from the time that the first packet is sent from the publisher to the time that the last packet was received by the subscriber as end-to-end delay. As every test is repeated 1000 times, we compute the average value for end-to-end delay and packet loss. Table 1 depicts the results.

Table 1. Average delay and loss rate for 1:1 transmission.

Hops	Source	Destination	Average Delay (ms)	Average Loss Rate
1	1	2	2439.16	4.23%
2	1	4	2754.45	3.21%
3	1	6	15,098.67	0.00%

From Table 1, the average delay rises very quickly with the increase of the hop count. However, there is a dramatic phenomenon: the greater the delay, the lower the packet loss rate. The main reason is due to the performance of the virtual machine we adopt. The reception capacity of the virtual terminal node limits the performance of SDNPS.

5.1.2. Delay and Loss Rate for 1:m Transmission

This set of experiments studies the delay and loss rate characteristics of the aforementioned testbed when faced with one publisher and multi-subscribers. The same experimental methods and evaluation methods are adopted. Table 2 presents the results. Compared to Table 1, the number of subscriptions does not impact delay significantly, as the average delays of these three tests increased (albeit slightly). This also validates our design. If unicast routing is adopted, there will be four-times the packets injected into the network for 1:4 transmission. In this case, the performance worsens greatly. Thus, we can draw a conclusion: SDNPS reaps benefits from its multicast routing.

Table 2. Average delay and loss rate for 1:m transmission.

Hops	Source	Destination	Average Delay (ms)	Average Loss Rate
1	1	2,3	2679.00	5.85%
2	1	2,4,5	13,248.00	0.00%
3	1	2,4,6,7	18,974.67	0.01%

5.1.3. Delay and Loss Rate for M:1 Transmission

In this section, we conduct a set of experiments to evaluate the performance of the SDNPS when facing multi-publishers and one subscribers. Table 3 shows the results. For every test, there are two publishers sending 100,000 packets simultaneously to one subscriber. There is no doubt that the bottleneck is still on the side of end hosts.

Table 3. Average delay and loss rate for m:1 transmission.

Hops	Average Delay (ms)	Average Loss Rate
1	4872.0	6.41%
2	22,138.0	0.12%
3	23,843.0	0.00%

5.1.4. Delay and Loss Rate w.r.t. Flow Entries

Tables 4–6 present how the flow entries impact the performance of SDNPS under one-to-one transmission mode. During the experiment, the flow table items of each switch vary between 1000 and 50,000 entries. With the augments of the flow entries at the switches, the delay is on the increase (but slightly) accordingly. This indicates that the entry size of the flow table has a slight effect on the processing delay of switches.

Table 4. Evaluation w.r.t. flow entries within 1 hop.

Flow Entries	Average Delay (ms)	Average Loss Rate
1000	2476	4.78%
5000	2927	2.15%
10,000	3044	4.98%
50,000	3200	2.53%

Table 5. Evaluation w.r.t. flow entries within 2 hops.

Flow Entries	Average Delay (ms)	Average Loss Rate
1000	2791	3.29%
5000	3325	0.00%
50,000	3574	0.21%

Table 6. Evaluation w.r.t. flow entries within 3 hops.

Flow Entries	Average Delay (ms)	Average Loss Rate
1000	15098	0.00%
5000	17486	0.00%
50,000	17959	0.00%

5.1.5. Throughput

From the above experiments, a lower delay means a higher loss rate. Additionally, we also clarify that the processing limitation at end hosts mainly causes the packet loss. In this experiment, we evaluate the ability of the switch and the end hosts to handle high incoming event rates within one hop. A publisher sends packets at varying rates. Beyond a certain packet rate, some packets are dropped by end hosts. We repeat this experiments many times, then find the fact that when the delay for 100,000 packets is larger than 3500 ms, the loss rate is fairly small. In another set of comparison experiments, we substitute a fast machine for the virtual machine as the end hosts; the delay for 100,000 packets should keep at least 2200 ms in order to get a very low loss rate. We can infer that the switches have the ability to forward events to the end hosts.

5.1.6. Normality and ANOVA Tests

To validate our experiments listed above, we perform some statistical analyses, including normality tests, which indicate whether the data from our experiments submitted to a normal distribution, and ANOVA tests, which analyze whether the factor we considered significantly affects the experimental results.

For normality tests, we conduct one-dimensional probability distribution verification leveraging the Kolmogorov–Smirnov test (k-s test). In this way, we verify the normal distribution characteristic of all of the experiments listed as Tables 1–6 at the confidence level of 95% ($\alpha = 0.05$).

In addition, we conduct one-way ANOVA analysis for the experiments listed as Tables 4–6. We take the delay time as samples to analyze whether the size of flow entries has a significant effect on the end-to-end delay of one-to-one transmission mode. The results are listed as Tables 7–9. The results indicate that the size of flow entries has an effect on delay under the three experimental scenarios.

Table 7. One-way ANOVA table for Experiment 4.

Source	SS	df	MS	F	Prob > F
Columns	2.89589×10^8	3	96,529,767.33	10,824.11	0
Error	3.56365×10^7	3996	8918.3	-	-
Total	3.25226×10^8	3999	-	-	-

Table 8. One-way ANOVA table for Experiment 5.

Source	SS	df	MS	F	Prob > F
Columns	2.24876×10^8	2	162,438,146.02	23,342.45	0
Error	2.08559×10^7	2997	6958.92	-	-
Total	3.45732×10^8	2999	-	-	-

Table 9. One-way ANOVA table for Experiment 6.

Source	SS	df	MS	F	Prob > F
Columns	4.67267×10^9	2	2,336,332,543.5	65,192.3	0
Error	1.07405×10^8	2997	35,873.6	-	-
Total	4.78007×10^9	2999	-	-	-

5.2. Performance Evaluation of Algorithm MCOTO

The algorithms for constructing topic-connected overlays for topics are different in light of their level in the topic tree. If a topic is in the first level, the algorithm is similar to MMMR [31]. The difference is, the output of MMMR is a forest; in our algorithm, we merge the forest into a connected graph, and we call it CMMMR. Consider an extreme situation: if the topics are all in the first level of the topic tree, our algorithm is typical CMMMR. If the topic tree degenerates into a single-branch tree, The majority of overlays are constructed by expanding the overlays of their father topics, and we call it EMCOTO. Otherwise, we call it MCOTO.

We implement our algorithms in Java to measure the effect under the three cases in terms of: (1) the execution time; (2) the maximum diameter; (3) the residual bandwidth.

The experiments are based on the following assumptions: The network is homogenous, in which the available bandwidths for the links are identical. The workload ranges: $|T| \in [40, 80, 120, 160, 200]$ or $|T| \in$ [1000,10,000]. The values of $p(t)$ are distributed according to a Zipf distribution (with $\alpha = 2.0$).

5.2.1. The Execution Time

The workstation used in the experiments is an Intel(R) Core(TM) (i5-3230 at 2.6 GHz) machine. We fixed the number of nodes at 200 and varied the number of topics in the range of {40,80,120,160,200}, while the number of relevant nodes (including publishers and subscribers) is 30, in which 20% are publishers. These relevant nodes are generated randomly. In MCOTO, the number of topics at the first level is half of the total number. The results are shown in Figure 5. Note that the time magnitude is 10^4 ms. As we can observe, the shape of the topic tree greatly affects the efficiency of the algorithms. The execution time under three cases increases along with the number of topics (Figure 6). When the topic tree is a single-branch tree, the execution time is the smallest. When all of the topics have no inclusion relationship, the execution time is maximum.

5.2.2. Topic Diameter

The topic diameter is defined as the maximum shortest distance between any two nodes on the same topic-connected overlay. Here, distance is measured with hop count. We use the same experimental parameters in Section 5.2.1. We compute the maximum shortest distance of all of the topic overlays, then take the mean value as the topic diameter. Figure 7 is a comparison of our algorithms in different cases for the diameter metric. The diameter increases for all three case. However, in EMCOTO, the diameter increases slowly. The main reason is that we expand the overlays from other relevant overlays that have been generated so that the number of the edges is less than the overlays we construct *ab initio*.

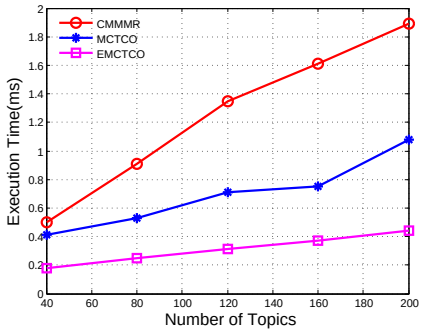

Figure 6. The execute time in three cases.

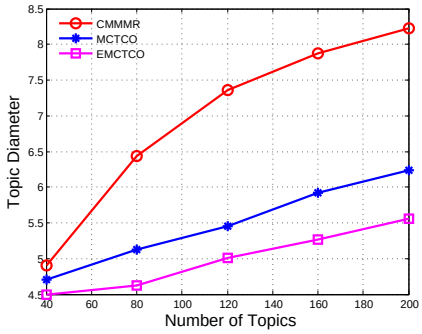

Figure 7. The topic diameter on different topic numbers.

5.2.3. The Residual Bandwidth

The residual bandwidth is defined in Definition 3. Here, we still adopt the aforementioned parameters. However, the publish events for every topic vary between 10,000 and 100,000. Assume that the bandwidth of all of the links is equal to 1000 units. Every publish events occupies 0.02 unit per link in the overlay. Figure 8 shows how different workloads impact the residual bandwidth.

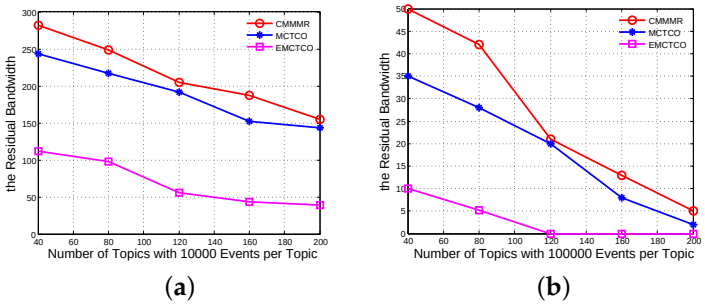

Figure 8. The residual bandwidth. (**a**) 10,000 events per topic; (**b**) 100,000 events per topic.

From the experimental results, the performance scales in terms of the topic tree. To obtain a better tradeoff between execution time and the residual bandwidth, the topic tree should be skillfully devised. Besides the natural language semantics of topics, the relevant node set is a primary factor to organize a topic tree.

6. Conclusions

In this paper, we attempt to construct a topic-based publish/subscribe system in an SDN environment.

The improvements provided by SDNPS constitute the basis for a highly reliable event diffusion infrastructure able to efficiently balance the load of links and avoid imprudent forwarding for events. By encoding topics and embedding them into packet headers as IPV6 multicast addresses, we can directly perform filtering and forwarding operations on SDN switches without detours to brokers. As a result, this significantly reduces the end-to-end latency.

The paper pays most of its attention to the inter-cluster routing mechanism incidentally with several different aspects of the event dissemination mechanism and proposes algorithms to construct topic-connected overlay networks, which in fact are a set of minimal cost Steiner trees induced by relevant publishers and subscribers. Particularly, our algorithms are interrelated with the topic tree. In this way, we can restrict the event dissemination scope to those pertinent nodes only, which greatly saves the bandwidth. Simultaneously, we also take the residual bandwidth into account. By maximizing the residual bandwidth as the optimization objective, we realized a load-balanced routing schema.

For all that, there are still some rough edges. These can be ameliorated later. In the future, we are going to work on two aspects of the system. The first fundamental aspect that must be pursued is the real-time requirement for some events, e.g., real-time alarm event for the Internet of Things (IoT). End-to-end latency can be cut down further by manipulating and fully capitalizing on the programmable characteristic of SDNs. Second, we will extend our experiments and test the system with reference to [35].

Acknowledgments: This research is supported by the National Natural Science Foundation of China under Grant Nos. 61372115, 61132001; the "973" program of the National Basic Research Program of China Grant No. 2012CB315802; the National High-tech R & D Program of China (863 Program) under Grant No. 2013AA102301.

Author Contributions: Yali Wang and Yang Zhang conceived of the whole paper. Yali Wang and Yang Zhang designed and performed the experiments. Yali Wang analyzed the data. Yali Wang wrote the paper. Yang Zhang and Junliang Chen revised the whole paper.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

SDN	software-defined networking
JSON	JavaScript Object Notation
MCTCO	minimal cost topic-connected overlay

References

1. Banno, R.; Takeuchi, S.; Takemoto, M.; Kawano, T.; Kambayashi, T.; Matsuo, M. Designing overlay networks for handling exhaust data in a distributed topic-based Pub/Sub architecture. *J. Inf. Process.* **2015**, *23*, 105–116.
2. Sun, Y.; Qiao, X.; Cheng, B.; Chen, J. A low-delay, lightweight publish/subscribe architecture for delay-sensitive IoT services. In Proceedings of the 2013 IEEE 20th International Conference on Web Services (ICWS), Santa Clara Marriott, CA, USA, 27 June–2 July 2013; pp. 179–186.
3. Eugster, P.T.; Felber, P.A.; Guerraoui, R.; Kermarrec, A.M. The many faces of publish/subscribe. *ACM Comput. Surv.* **2003**, *35*, 114–131.
4. Chockler, G.; Melamed, R.; Tock, Y.; Vitenberg, R. Spidercast: A scalable interest-aware overlay for topic-based pub/sub communication. In Proceedings of the 2007 Inaugural International Conference on Distributed Event-Based Systems, Toronto, ON, Canada, 20–22 June 2007; pp. 14–25.
5. Voulgaris, S.; Gavidia, D.; van Steen, M. Cyclon: Inexpensive membership management for unstructured p2p overlays. *J. Netw. Syst. Manag.* **2005**, *13*, 197–217.

6. Jafarpour, H.; Mehrotra, S.; Venkatasubramanian, N. Dynamic load balancing for cluster-based publish/subscribe system. In Proceedings of the Ninth Annual International Symposium on Applications and the Internet, Seattle, WA, USA 20–24 July 2009; pp. 57–63.
7. Cheung, A.K.Y.; Jacobsen, H.A. Load balancing content-based publish/subscribe systems. *ACM Trans. Comput. Syst.* **2010**, *28*, 9.
8. Baldoni, R.; Querzoni, L.; Tarkoma, S.; Virgillito, A. Distributed event routing in publish/subscribe systems. In *Middleware for Network Eccentric and Mobile Applications*; Springer: Heidelberg, Germany, 2009; pp. 219–244.
9. Castro, M.; Druschel, P.; Kermarrec, A.M.; Rowstron, A.I. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE J. Sel. Areas Commun.* **2002**, *20*, 1489–1499.
10. Zhuang, S.Q.; Zhao, B.Y.; Joseph, A.D.; Katz, R.H.; Kubiatowicz, J.D. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video, Port Jefferson, NY, USA, 25–26 June 2001; pp. 11–20.
11. Liotou, E.; Tseliou, G.; Samdanis, K.; Tsolkas, D.; Adelantado, F.; Verikoukis, C. An SDN QoE-Service for dynamically enhancing the performance of OTT applications. In Proceedings of the 2015 Seventh International Workshop on Quality of Multimedia Experience (QoMEX), Costa Navarino, Greece, 26–29 May 2015; pp. 1–2.
12. Baldoni, R.; Beraldi, R.; Quema, V.; Querzoni, L.; Tucci-Piergiovanni, S. TERA: Topic-based event routing for peer-to-peer architectures. In Proceedings of the 2007 Inaugural International Conference on Distributed Event-Based Systems, Toronto, ON, Canada, 20–22 June 2007; pp. 2–13.
13. Ramasubramanian, V.; Peterson, R.; Sirer, E.G. Corona: A High Performance Publish-Subscribe System for the World Wide Web. In Proceedings of the Symposium on Networked Systems Design and Implementation, San Jose, CA, USA, 8–10 May 2006; Volume 6, pp. 115–117.
14. Banerjee, S.; Bhattacharjee, B.; Kommareddy, C. *Scalable Application Layer Multicast*; ACM: New York, NY, USA, 2002; Volume 32.
15. Chockler, G.; Melamed, R.; Tock, Y.; Vitenberg, R. Constructing scalable overlays for pub-sub with many topics. In Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, Portland, OR, USA, 12–15 August 2007; pp. 109–118.
16. 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 (ICDCS), Genova, Italy, 21 June 2010; pp. 622–633.
17. Chen, C.; Jacobsen, H.A.; Vitenberg, R. Algorithms based on divide and conquer for topic-based publish/subscribe overlay design. *IEEE/ACM Trans. Netw.* **2015**, *24*, 422–436.
18. Onus, M.; Richa, A.W. Minimum maximum-degree publish-subscribe overlay network design. *IEEE/ACM Trans. Netw.* **2011**, *19*, 1331–1343.
19. Onus, M.; Richa, A.W. Parameterized maximum and average degree approximation in topic-based publish-subscribe overlay network design. *Comput. Netw.* **2015**, doi:10.1109/ICDCS.2010.54.
20. Darugar, P.T.; Martinez, F.; Toth, P.K. Network Publish/subscribe System Incorporating Web Services Network Routing Architecture. US Patent 7,349,980, 25 March 2008.
21. Baldoni, R.; Beraldi, R.; Querzoni, L.; Virgillito, A. Efficient publish/subscribe through a self-organizing broker overlay and its application to SIENA. *Comput. J.* **2007**, *50*, 444–459.
22. Nunes, B.A.; Mendonca, M.; Nguyen, X.N.; Obraczka, K.; Turletti, T. A survey of software-defined networking: Past, present, and future of programmable networks. *Commun. Surv. Tutor. IEEE* **2014**, *16*, 1617–1634.
23. Zhang, K.; Jacobsen, H.A. SDN-like: The next generation of pub/sub. *IEEE Trans. Intell. Transp. Syst.* **2013**, *14*, pp. 883–893.
24. Jain, S.; Kumar, A.; Mandal, S.; Ong, J.; Poutievski, L.; Singh, A.; Venkata, S.; Wanderer, J.; Zhou, J.; Zhu, M.; *et al*. B4: Experience with a globally-deployed software defined WAN. In *ACM SIGCOMM Computer Communication Review*; ACM: New York, NY, USA, 2013; Volume 43, pp. 3–14.
25. 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, Montreal, QC, Canada, 3–7 December 2012; p. 3.

26. Jokela, P.; Zahemszky, A.; Rothenberg, C.E.; Arianfar, S.; Nikander, P. LIPSIN: Line speed publish/subscribe inter-networking. *ACM SIGCOMM Comput. Commun. Rev.* **2009**, *39*, 195–206.
27. Tariq, M.A.; Koldehofe, B.; Bhowmik, S.; Rothermel, K. PLEROMA: A SDN-based high performance publish/subscribe middleware. In Proceedings of the 15th International Middleware Conference, Bordeaux, France, 8–12 December 2014; pp. 217–228.
28. Bhowmik, S.; Tariq, M.A.; Koldehofe, B.; Kutzleb, A.; Rothermel, K. Distributed control plane for software-defined networks: A case study using event-based middleware. In Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems, Oslo, Norway, 29 June–3 July 2015; pp. 92–103.
29. Hakiri, A.; Berthou, P.; Patil, P.; Gokhale, A. Towards a publish/subscribe-based open policy framework for proactive overlay software defined networking. *ISIS* **2015**, Available online: http://www.isis.vanderbilt.edu/biblio/ (accessed on 29 August 2015).
30. Vilalta, R.; Mayoral, A.; Pubill, D.; Casellas, R. End-to-End SDN Orchestration of IoT Services Using an SDN/NFV-enabled Edge Node. In Proceedings of Optical Fiber Communication Conference, Anaheim, CA, USA, 20–24 March 2016.
31. Syrivelis, D.; Parisis, G.; Trossen, D.; Flegkas, P.; Sourlas, V.; Korakis, T.; Tassiulas, L. Pursuing a software defined information-centric network. In Proceedings of the 2012 European Workshop on Software Defined Networking (EWSDN), Darmstadt, Germany, 25–26 October 2012; pp. 103–108.
32. Tseliou, G.; Adelantado, F.; Verikoukis, C. Resources negotiation for network virtualization in LTE-A networks. In Proceedings of the 2014 IEEE International Conference on Communications (ICC), Sydney, Australia, 10–14 June 2014; pp. 3142–3147.
33. Banavar, G.; Chandra, T.; Mukherjee, B.; Nagarajarao, J.; Strom, R.E.; Sturman, D.C. An efficient multicast protocol for content-based publish-subscribe systems. In Proceedings of the 19th IEEE International Conference on Distributed Computing Systems, Austin, TX, USA, 31 May–4 June 1999; pp. 262–272.
34. Chen, Y.R.; Radhakrishnan, S.; Dhall, S.; Karabuk, S. On multi-stream multi-source multicast routing. *Comput. Netw.* **2013**, *57*, 2916–2930.
35. Munoz, R.; Mangues-Bafalluy, J.; Vilalta, R.; Verikoukis, C.; Alonso-Zarate, J.; Bartzoudis, N.; Georgiadis, A.; Payaro, M.; Perez-Neira, A.; Casellas, R.; *et al.* The CTTC 5G end-to-end experimental platform: Integrating heterogeneous wireless/optical networks, distributed cloud, and IoT devices. *IEEE Vehicul. Technol. Mag.* **2016**, *11*, 50–63.