Programming Network via Distributed Control in Software-Defined Networks

Boyang Zhou and Chunming Wu*
College of Computer Science
Zhejiang University
Hangzhou 310027, China
{zby,wuchunming}@zju.edu.cn

Xiaoyan Hong
Department of Computer Science
University of Alabama
AL 35487, USA
hxy@cs.ua.edu

Ming Jiang
Institute of Soft. and Intelli. Tech.
Hangzhou Dianzi University
Hangzhou 310027, China
jmzju@163.com

***Abstract*—Programming a network for innovative services or for function improvements has never been easier using Software-Defined Networking (SDN). However, the programming tasks can also be significantly complicated by the asynchrony of data plane states and complexities of service control states. In order to reduce the complexity for programming a network service in Distributed Control Plane of SDN, we propose a proGRAmming Control (GRACE) layer as a generic solution, which provides two key features, namely, reconfigurability and reusability. Their implementations deal with aforementioned challenges, thus to achieve the consistency of the data plane states and the reusability of the service control states at the distributed controllers. This paper introduces the reconfigurability and reusability with their design goals and their impact on the programmability of DCP. We further use two popular network services, ICN (Information-Centric Networking) and CDN (Content Distribution Networks) to illustrate these concepts. NS-3 simulations and PlanetLab emulations are conducted to show the advantage of using the GRACE layer for ICN and CDN. Results show that the ICN Interest delay is reduced by 19.6% and CDN request delay is reduced by 81% in extremely harsh network conditions.**

I. Introduction

Programming a network for innovative services or function improvements has never been easier thanks to the separation of the controller plane and data plane in Software-Defined Networking (SDN)[1]. Further, distributed control plane (DCP) of SDN has enabled network service developments in a large scale with flexibility and configurability[2][3][4]. In DCP, the data plane is partitioned into multiple domains of switches. Each domain is controlled by its controller in the control plane in the centralized manner. The controllers collaborate with each other in DCP to support network services. For each service, a controller configures its switches to execute the forwarding logics of the service logic; and many controllers disseminate their control states among themselves to ensure they achieve the service logic with consistency.

Programming a network with DCP is a process of transferring the designed service logics to the running code at control plane and also writing the forwarding logics and control states to the data plane. Moreover, as the state-of-art Internet research and development demonstrate, new innovative designs merge consistently and quickly. Thus, programming a network becomes a frequent event, enabling evolution of the functionalities of the running services.

The process of programming the network contains two major tasks. One task is updating the new service logics at all the controllers according to the new research/development ideas. The other task is updating the forwarding states at the data plane. The former requires the control states at all the controllers to be synchronized and consistent about the new service logics. The latter requires all the forwarding states to be consistent, which in turn, requires all the controllers to reconfigure their switches in a synchronized way.

However, these tasks can be significantly complicated by asynchrony in communications and instability of the data plane states that are maintained at the controllers, as the controllers can undergo physical device or link failures, software or configuration malfunctions as well as highly unbalanced traffic load. When different controllers perform a series of reconfigurations at runtime, the data plane states become uncontrollable if the transient states occur in reconfiguring. Such a issue can trigger further issues such as service unavailability, flow interruption and security holes. These latter problems have also challenged the current Internet routers[5][6], critically impacting on the service availability in a similar way caused by the asynchrony of data plane states, e.g. a research has shown that the transient unavailability of ASes in BGP contributes to 90% packet loss in a Sprint network study[7].

In addition, considering when multiple services running in DCP, each service has to deal with these challenges in order to achieve the consistency of control and data plane states, thus duplicate efforts are spent to solving the same problem. It is desirable that the controller can provide the common functions to these services and the services can share certain common modules with each other. These functions are specialized to dealing with the physical conditions.

The state-of-the-art of the SDN controller functions is far from sufficient in offering efficient methods to deal with the aforementioned underlying challenges with decreased complexities[8][9][3][10], which limits programmability of DCP in supporting novel network services.

The reconfigurability and reusability are two properties of DCP that we propose to address the complexity in programming a network. Specifically, DCP will be able to achieve the reconfigurability of the data plane states and the reusability of the control states of many services. In this paper, we introduce the concepts of reconfigurability and reusability, the principles they enable to control the programming process, and an abstract architectural solution named as a proGRAmming Control (GRACE) layer to facility the programming tasks in DCP.

Moreover, we use two cases, namely, the Information-

* corresponding author, E-mail: wuchunming@zju.edu.cn

978-1-4799-2003-7/14/$31.00 ©2014 IEEE

Centric Networking (ICN)[11] and the Content Distribution Networks (CDN)[12], together with the DCP inter-domain control protocol to demonstrate these concepts and our solution at work. The two properties realized in the GRACE layer can improve the performance of the network controls services.

The main focus of the paper is to demonstrate that programming a network can be easier with the support of reconfigurability and reusability. Specifically, the paper contributes in three aspects:

(1) For the first time in literature, the detailed definitions of reconfigurability and reusability for the DCP are given, along with the requirements in achieving the goals when developing corresponding technical solutions. The paper also reasons over the importance of the two features to the network programmability.

(2) An abstract framework of the GRACE layer is given. GRACE holds the implementation of the protocols that achieve the reconfigurability and reusability. It sits between the service logic layer and the OpenFlow protocol[13] of the controller. The protocols implemented within GRACE handle the aforementioned asynchrony and instability during the dynamics process of updating the distributed data plane states. Leveraging main focus of the paper, the descriptions about these protocols and GRACE stay at a conceptual level with only a little detail.

(3) Two network service examples are analyzed for better illustrations of how to program using distributed control in SDN with the support of the GRACE layer. Both NS3 simulation[14] and PlanetLab[15] emulation results are given to show the performance gain. Specifically, ICN showcases both the reconfigurability and reusability; and CDN showcases reusability. Both of them produce reduced delay in harsh network conditions.

These contents are organized as follows. The two key properties of network programmability of the distributed control plane and the GRACE layer are introduced in Section II. The examples using GRACE layer are analyzed in Section III. And Section IV gives the evaluations in both NS3 ndnSim and PlanetLab in. Section V and VI discuss the related work and conclude the paper, respectively.

II. Network Programmability

In this section, we first introduce the two major features of network programming, namely, reconfigurability and reusability, and then briefly describe the GRACE layer to implement these features.

A. Reconfigurability of Data Plane States

The reconfigurability is the ability to safely update the current forwarding states at the data plane to the new states, without interrupting the data flows. During the update period, one or multiple controllers write to the forwarding states of their switches via the OpenFlow (OF) protocol. The forwarding states control the different switch resources to decide the output behaviors of the incoming packets at the switch level. Examples include the forwarding rules in the flow table, the rate limiting on the input queues as well as up and down statuses of the ports.

During the updating process, the service availability could be disrupted due to the losses of the in-flight packets. Because, when some switches current forwarding states are changed to the new states, the others may be not yet, generating the transient state problem. Such hardness increases the complexities in developing a safe service on DCP, and deteriorates the flexibility in its programmability.

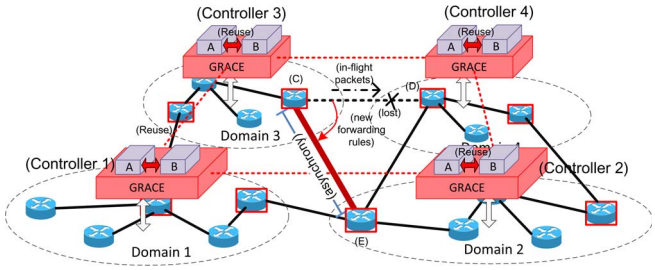


Fig. 1. Network Model of Distributed Control in SDN

The transient state problem is generic for different types of services, to avoid the inconsistency of updating the data plane states. The problem can be specified as Eq.1. The set of $\{S_1, S_2, S_n\}_{lt}$ is a collection of all the forwarding states of a service that is running on the controllers at the logical time lt. Given two logical times in a continuous control sequence as lt_1 and lt_2, an update to one of S_i leading to state deviation between two logical times. The transient states are defined as the intermediate state sequences when there are at least two state deviations. Thus, the dynamic reconfiguration process.

$$(S_1, S_2, ..., S_n)_{lt} \longrightarrow (S_1^*, S_2^*, ..., S_n^*)_{lt+1} \quad (1)$$

Taken routing and firewall as examples (the service A and B in Fig.1). For the routing service, suppose initial path from C to E goes through D. When the latency of link CE reduces to a point that the controllers 2, 3 and 4 agree to the new route of CE directly. The old route of CE is via D. (CE is denoted as the red bold link in Fig. 1), The forwarding states of switches C, D and E will be reconfigured by their controllers. If the three switches are not updated at the same time, then either loop will form if the switch C is updated earlier than D and E (in-flight packets will be dropped when TTL drops to zero); or black hole will form at node D when switch D is updated earlier than C (the link DE is removed causing in-flight packet drops). For the firewall service, suppose an operator reconfigure the filters of D and E to discard all packets from C. When D is updated earlier than E, the old forwarding states of E could still let packets pass via E and then reach to D. The configuration is not safe.

The impacts of the above examples are on service availability and security. However, the root cause is the same, the transient state problem of the data plane. From the examples, the transient state problem can be solved by enabling the reconfigurability of data plane states

The goal of the proposed reconfigurability property is to ensure safe updates on the data plane states, i.e, to solve the transient state problem. Thus, when controllers are programmed to support a higher-level service logic, the lower-level logics of network control should have the ability to guarantee that the service will see the consistent data plane states. More precisely, the technical solution that offers reconfigurability should conceal the asynchrony of data plane, but deliver consistent states. Our approach is to build the GRACE layer with the reconfiguration primitive in dealing with the transient state.

B. Reusability of Control States

The reusability is the ability that two or more services share common control states. In DCP, the control states are

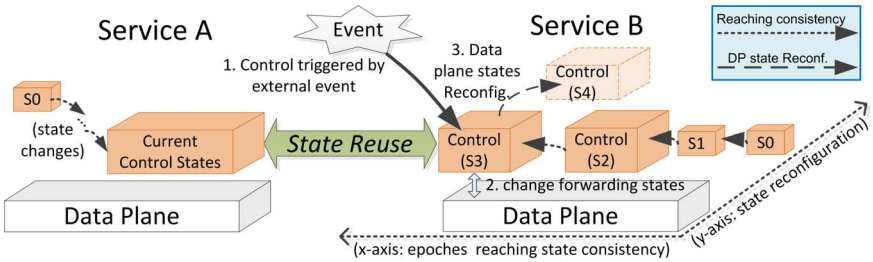


Fig. 2. Programming in DCP

maintained at the controllers. They are used by each controller for two purposes: (i) to configure the forwarding states of switches in its domain; and (ii) to coordinate with other controllers to perform service logics.

Usually, the control states of a service are isolated from the control states of the others. Two artifacts are generated: one is that each service will need to have their own functionality to deal with issues relating to the control states, and the other is that when one service pushes the network conditions to change, other services are not able to learn until a performance bottleneck occurs.

As such, allowing a service to reuse the control state of another service should be able to improve the programmability in that, exposing the network control state for others to use helps the performance of the other service. They can optimize their traffic based on the control dynamics, e.g. path congestion states in a routing service. As such, the accessibility of the control states must be carefully managed.

Our approach to achieve the reusability is to push the control states into the GRACE layer with two access models, namely, active or passive access respectively.

C. Programmability with Distributed Network Control

We use Fig.2 to illustrate that the two tasks of network programming can be easier using the reconfigurability and reusability. Two services A and B are depicted in the figure, each showing the control plane and the data plane. The service states are the combination of the data plane states and the control states. The control plane decides the new control states and then configures the switches in its domain with the new states.

With Service B, we demonstrate two programming cases that call for the reconfigurability. Fig.2 shows the state changes in a two dimension coordinate system. Along the x-axis, the control states of the service B transit from S0 to S3 in time epochs, eventually reach a complete state when the control states at all the controllers are all updated to the new service logic. During this time series, the reconfigurability ensures that the corresponding forwarding states reach consistence on the same values too. Fig.2 also shows an external event which triggers the service B to update the forwarding states at the data plane, transiting from S3 to S4 (show at y-axis). The safe transition between the two states is ensured by the reconfiguration primitive which is implemented in the GRACE layer (see Subsection II.D).

In Fig.2, service A is used to show how service B can reuse the states of service A. The most common example is for service B to reuse the control states of network topology from service A while its control states change, supposing service A has implemented the reconfigurability and reusability, and offers interfaces for others to access.

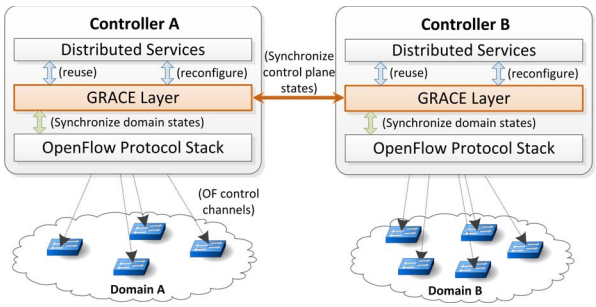


Fig. 3. High-Level Architecture of GRACE Layer

D. GRACE Layer

Fig.3 shows the high-level architecture of the GRACE layer. The layer sits between the OpenFlow protocol stack and the distributed services. The stack provides the function to encapsule OpenFlow messages and is already implemented by the NOX controller[8]. The layer can control the switches in the current domain via the stack according to the service logics (see the arrow between the OpenFlow protocol stack and the GRACE layer in Fig.3). The two layers on two different controllers can synchronize their control states with each other (see the arrow between controllers A and B in Fig.3).

The layer implements mechanisms for the reconfigurability to achieve the consistency in handling the deviations of data plane states between the two data plane configurations. It also implements mechanisms for the reusability to safely open the access of the control states of a service to other services. The core primitives are provided by the GRACE layer to allow the services to access these mechanisms (see two arrows between distributed services and OpenFlow protocol stack in Fig.3). They conceal the complexity of the technical details in handling the dynamics and synchronization of the control states and data plane states during service runtime. The two interfaces are briefed below.

The *reconfiguration primitive* is used to update the forwarding states of the data plane. The execution of the primitive generates strictly locked phases of the reconfiguration process. There are four phases. The first phase takes indication from the service about the (yet-to-be-committed) new forwarding states. The second phase collects consensus to see whether all the controllers have the same new forwarding states and be ready for next phase. In the phase, all the controllers consent on a global tag by using the Paxos algorithm[16]. Then, all the controllers write the new forwarding states to the switches via the OF protocol. The final phase collects confirmations from all the controllers about the completion of the updates.

The *state reuse primitive* opens and manages the access of the control states of the current service to other services. To achieve that, the control states of a service are read-only for other services. The different services isolate their control

states with each other to avoid the state collisions. Further, all the control states of a service are organized using Chord (a distributed hashing table (DHT)[17]). The control states are stored as key-value pairs on the DHT nodes. The DHT is modified to add the updating triggers for active access. When a service wishes to reuse the control states of another service, it registers to the trigger of the control states of interest via the primitive interface. When a change occurs to the control state, message will send to the controller of the current service about the latest values of the control state. For example, instead of building a congestion module of its own, a service can learn the changes of congestion levels by simply reusing the topology states or the routing states of another service via the primitive.

III. Case Analysis

In the previous section, we have introduced how the reconfiguration primitive and the state reuse primitive can help programming a network with ease. In this section, we analyze two popular network service examples with their programming implementations in DCP. We use them to demonstrate the benefits of the reconfigurability and reusability supported by the GRACE layer. These two cases will reuse the control states of the inter-domain routing service (IDR), which is developed for controller communications. We start with introducing the routing service IDR.

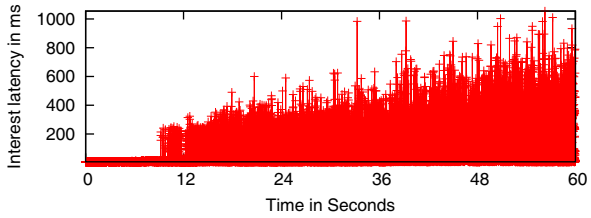
A. Inter-Domain Routing

The Inter-Domain Routing (IDR) in this paper is specially developed for controlling of the data forwarding between domains. The IDR can be seen as a type of the link state routing service. It adopts the idea of consistency-based link state routing, which has shown better flexibility than the common dynamic routing approach in the Internet architecture[2][18][3][4]. IDR builds on the state reuse primitives of the programming control layer. So it computes routes to reach other domains based on the consistent states of the network topology. As shown in Fig.1, the red dash lines form the topology of the controllers.

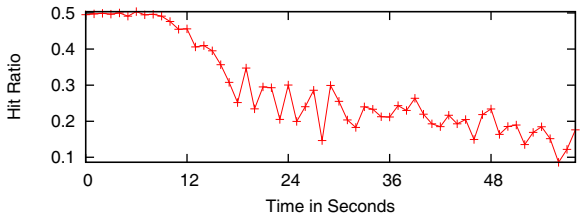
IDR takes three steps to converge to new routes in the distributed manner: (1) Each controller obtains and aggregates the switch states in its domain, integrates them into the global topology states; (2) Each controller uses Paxos[16] to obtain consensus on the consistence of topology states and then uses Dijkstra algorithm to compute the domain level routing path (the intra-domain routes are computed by the routing service in the original controller realization); and (3) The leader controller calls the reconfiguration primitive to update the new behavioral states of its switches. Thus, this step can be safely executed without the transient state problem via the primitive (see Subsection II.D). In addition, if the topology states are changed, such as the link congested, every controllers re-executes the steps 2 and 3.

B. Information-Centric Networking

ICN enables the information providers to deliver contents at subscribers requests by receiving senders Interest packets[11]. The forwarding information bases (FIBs) of the ICN at the switches are used to forward the Interest packets by the content prefixes. FIBs are computed by the content routing in ICN. The routing is realized in the service logic layer to compute the best routes to the content prefixes. When a switch has the content that is requested by an Interest packet, its corresponding data packets will reversely traverse the path where the Interest



(a) Interest delay



(b) hit ratio

Fig. 4. Interest Delay and Hit Ratio for the ICN Nodes Impacted by Link Congestions

packet comes from. The data content are cached at each switch by the least recently used (LRU). The state-of-the-art ICN routing logics are realized in the global fashion: (i) Each controller disseminates its content prefixes as the control states to others in a periodical manner for the consistence of content states (i.e. the control state); (ii) Each controller compute the best paths to the content prefix from the current node using the shortest path first algorithm; and (iii) Each controller updates the behavioral states of switches by the FIBs. However, the content routes need to be changed in time to avoid congestion. The artifact can be observed for the route changes in ICN. When the Interest packets with different prefixes are densely flooded into all switches by a leaf switch, the performance of ICN will be deteriorated, because hit ratio of caching is largely dropped and queues are congested.

We simulate the ICN with NS3 ndnSim[14]. Our simulation is to investigate how well the ICN switches react to the congestion events (the simulation setups can be found in Sec IV.A). The sending rate is exponentially distributed within 0-1000req/s. We observed that there is a sharp increase of delay at the execution time of 9 second (see Fig.4(a)), correspondingly, there is a sharp decrease of the content hit ratio starting at the 9 second (see Fig.4(b)). This is because that after 9 seconds of execution, the content storages of the switches are unable to cache further data records, which causes either Interests loss, or longer paths to retrieve contents after more Interests propagations.

In terms of ICN control logic, the large amount of content requests changed the states of underlying links into congestion. Because the content routing protocol measures the states and computes new routing paths by converging steps, it is hard for the control logic to capture the highly dynamic link congestions. In addition, changing routing paths also drives the ICN control logic into the transient states, which decreases the availability of the ICN service. In addition, the routing convergence will cause the service flows of ICN will be interrupted in reconfiguring the data plane states. Because the asynchrony of the data plane states is still existed in multiple nodes.

To deal with such problem, we implemented the ICN control logic with the support of GRACE layer. ICN routing can

reuse IDRs topology states, thus the routing can be converged quickly in reacting to congestions changes. When the FIBs are changed by new routes, the reconfiguration primitive is triggered by such updating event to safely deal with changes of behavioral states of FIBs without the impact of transient states (see Subsection II.D).

We will show results in Section IV where the quality of the ICN control and the ICN performance are improved.

C. Content Distribution Networks

CDN is consisted of a centralized CDN content publisher to manage content replicas, and many CDN content distributors to distribute the contents[12]. It offers better scalability by delivering replica from locations closet to the requesting end users. However, the performance of the current CDNs is impacted by the dynamics of the networks as well as the complexities of the topology discovery and the routing. Such that, delivering the SLA (Service Level Agreement) assured CDN flows (such as delay upper bound) can be difficult. One current solution is to integrate CDN with ALTO/P4P[19], so that the CDN can choose the best distributor for the users. However, the current approaches on ALTO still have the fundamental limitations: The network topology cannot be accurately estimated with ease from the application layer[19]. And routing is hard to be optimally decided based on the topology. With CDN, the GRACE layer can safely provide a way to reuse the IDRs routing states (see Subsection II.D). With the network control states information, CDN application can improve its estimation and calculate the best routing path.

D. Case Summary

In summary, IDR, ICN and CDN showcase our approach to improve the programmability with DCP. They show the usefulness in optimizing the application performance with the reconfigurability of the switch states and the reusability of the control states. Specifically, IDR and ICN reconfigure the behavioral states of the forwarding rules in their FIBs according to underlying topology changes by reusing the topology states. As shown in Fig.5, three services of IDR, ICN and CDN are shown. The circles in each service are the functions of the control planes belonging to the service. The write and blue rectangles in each service are the control states and data plane states of the service. The dashed red arrows are the reuse dependences of control states for the three services. IDR exposes the topology states and routing states that are reusable for ICN and CDN (see Subsections III.B and III.C). ICN and CDN can adjust their network control objectives according to changes of the control states of IDR. Specifically, ICN reconfigures its data plane states if the IDRs topology states reveal that a link is congested. And CDN updates its content routing if the states show that the routing states are changed. The blue arrows are the concrete execution steps of the control functions. The reconfiguration and reuse are alternatively executed for each service to achieve its control objective. In later section, we will show both have better performance.

IV. Performance Evaluation

In this section, we show that the GRACE layer improves the performance of both ICN and CDN when they undergo network congestion events which force them to update the forwarding states in the data plane. Such improvement is brought by the control flexibility of DCP enabled by the layer.

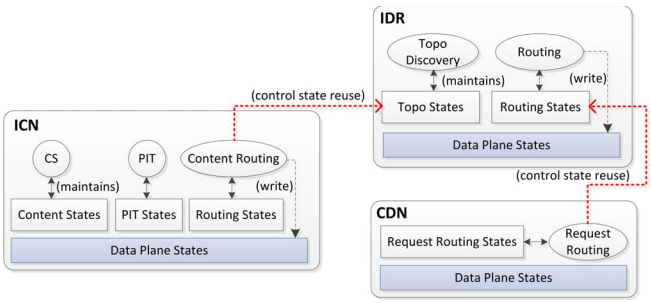


Fig. 5. Example of Reconfiguration and Reuse of Control states

A. Performance Evaluation of ICN

We evaluate ICN to show: (1) the reconfigurability of the control states safely transit states of the data plane at run time; and (2) the reusability improves performance by being aware of changes of topology states. The prototype of the ICN running on the GRACE layer is experimented in simulation by the NS3 ndnSim[14].

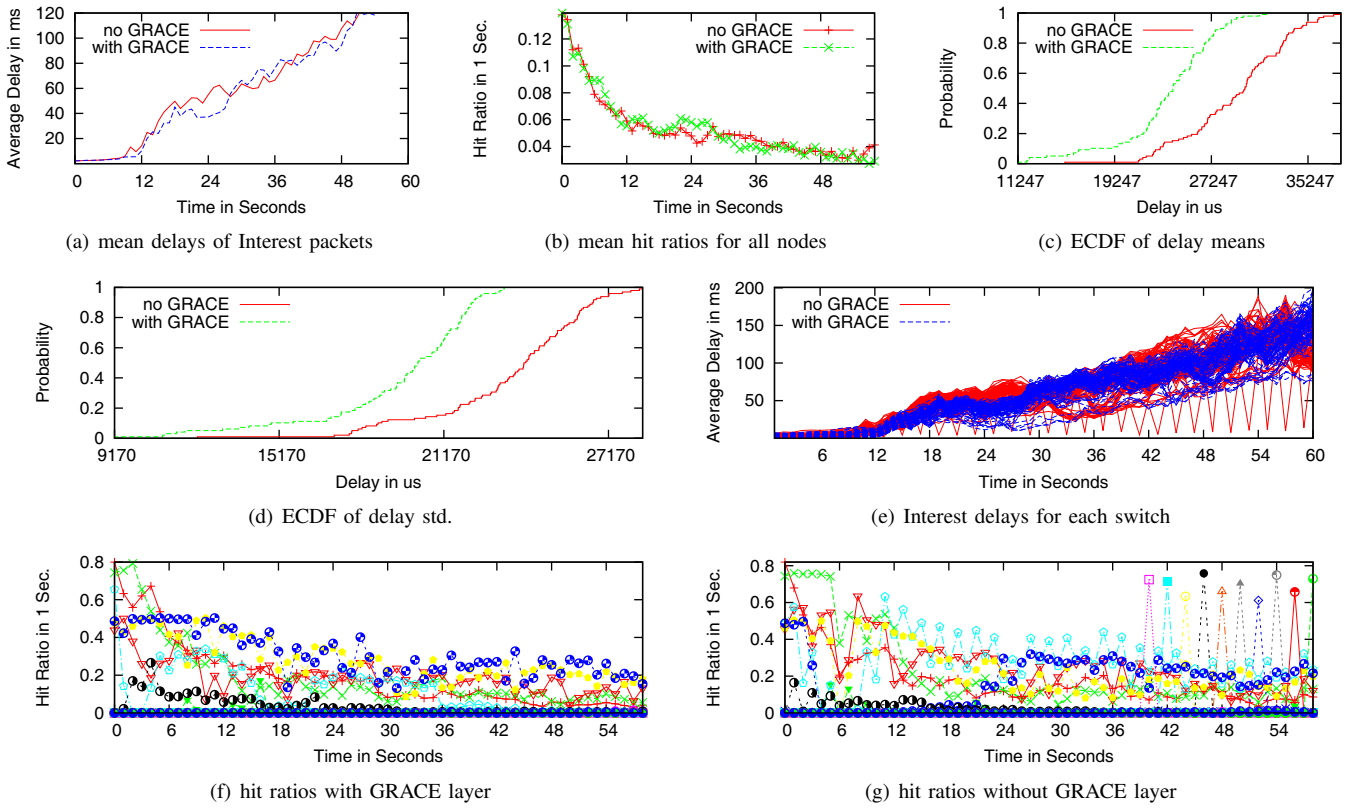
We use a synthesis network topology with 200 nodes generated by BRITE in which we uses the Waxman ($\alpha = 0.5, \beta = 0.8$) model and the bandwidth is uniformly distributed between 100Mbps to 1000Mbps[20]. The topology has 200 nodes and 99 of them are leaf nodes. In simulation, there are two content origins for ICN: (1) a testing content that is requested by all the 99 leaf nodes with a frequency uniformly distributed within 0-1000 req/s and runs in 0-60s; and (2) a background content that is requested by all nodes in the topology from 0-30s with a frequency exponentially distributed within 0-1000 req/s. Each testing request for each leaf node is sent with its unique prefix of leaf name and packet sequence number. The starting and stopping times of the background content are the ON/OFF model with period of 2s. Each node contains a distributed SDN controller that runs ICN over the GRACE layer and an ICN switch.

The simulation results are given in Fig.6. They show that when the ICN network is congested by heavy Interest flows (i.e. 0-30s), the data plane states can stabilize quickly in the presence of rapid changing congestion conditions at different links, which reduces the propagation latency of the Interest packets. It is because of the reuse of the control states of IDR. Fig.6(e) shows an 5% increase the hit ratio of the content storage on the average for all Interest requests of all nodes (see Fig.6(b)). However, when we randomly choose 30 nodes (see Fig.6(f) and Fig.6(g)), the results show that the ratio of highly fluctuation reduces, which suggests that the hit ratios of the ICN on GRACE are more stabilized than the one without GRACE under the highly dynamic congestion cases. Also, Fig.6(a) and Fig.6(c) show that the Interest delays reduced 19.6% on average. Further, the data delay with GRACE is more stabilized than the one without GRACE, which is indicated by the fact that mean of the standard deviation of the request delays is lowered by 17.1% than the results taking average of all paths (see Fig.6(a) and 6(d)). This is because nodes of ICN rapidly react to the congestions and reconfigure its control.

B. Performance Evaluation of CDN

We show that the CDN running on the GRACE layer can improve performance of routing control through evaluation on PlanetLab. The prototype of the CDN on GRACE is based on CoralCDN[12].

We choose uniform distribution because it is sufficient for our evaluation purpose. We consider using more realistic distributions in the future work.



(a) mean delays of Interest packets

(b) mean hit ratios for all nodes

(c) ECDF of delay means

(d) ECDF of delay std.

(e) Interest delays for each switch

(f) hit ratios with GRACE layer

(g) hit ratios without GRACE layer

Fig. 6. Data Delays and Hit Ratios for ICN Networks(All links are Impacted by Heavy Interest Packets)

To evaluate the CDN performance, we randomly choose 9 nodes from PlanetLab that emulates a physical network topology based on Minimum Spanning Tree (MST) using the average link costs 5 times more than the measured TCP latency (see Fig.7). In the topology, there are three distributors of CDN, namely, PL2-AUKLAND, PL5-UCSE and PL0-JAIST, one content requester at PL4-UCSB and one node for content routing at PL1-WASEDA. To emulate a network with the highly fluctuated latencies, the links in the paths between the distributors and the requester are mixed with two types of traffic to create the control dynamics: the first type is a dense Poisson traffic with throughput of 1000 req/s to emulate the CDN requests, and the second is the real background traffic of the PlanetLab, which will interfere with large latency fluctuations, to emulate parallelized service traffics. Some measurements of the underlying testbed are: (1) high packet loss rate is observed in that 15% of the links have a loss rate more than 10%; and (2) high latency deviation is observed in that 3.7% of the links have the latencies more than 1000ms. And each node contains a controller and a switch.

The emulation results are given In in Fig.8. The large delay gap between the two CDNs (with and without the GRACE layer) during 0-599s is mainly caused by higher fluctuations of the PlanetLab background traffic; while the smaller delay gap between the two lines during 600-2000s is mainly caused by the more stable Poisson background traffic. We observed that the content request delay is much lower and more stable when the GRACE layer is used. The delay is also within the acceptable range, according to Fig.8, the mean value and the standard deviation are 386ms and 329ms, respectively. In contrast, a native deployment of CDN (without the GRACE layer)

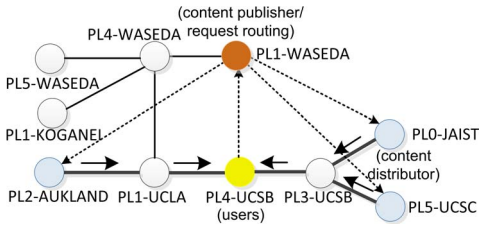


Fig. 7. Emulation Topology of CDN

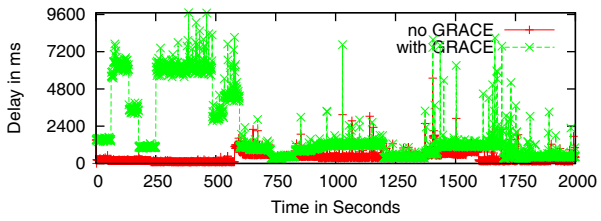


Fig. 8. Content Delays for CDN Nodes on PlanetLab Impacted by Highly Fluctuated Congestion States of Links

produces the mean and the standard deviation to be 2033ms and 2105ms, respectively. Once the users sending requests, the three paths are efficiently decided for the distributors by the routing state that are consistent among them, so the optimal path decision is then delivered to the users. Overall, the layer reduces 81% average request delay of CDN. It indicates that the reusability of topology states cab help CDN to optimize its control performance.

V. Related Work

Recent work on the programmability of the SDN con trollers have three major trends: (1) The approaches of the centralized network control like the OpenFlow/SDN paradigm

work on the programmability of a single SDN controller, including creating a SDN programming language to lower the complexities of the switch configuration like NetCore[21] and the incremental consistent update[22]. However, the programmability of distributed controls is not concerned.

(2) The approaches of patching on the current distributed control protocols improve the programmability for a specific protocol. In detail, the works like those in RCP[2], 4D[18] and the consensus routing[5] improve the performance of BGP by turning the dynamic route computation fashion of the inter-AS routing into the distributed system centric one. The consensus routing focuses on minimizing the transient unavailability of ASes in BGP. Most prior works[23][24] for the Internet control focused on protocol specific approaches of adjusting network metrics. And a work[25] proposes a method that seamlessly migrates one configuration of intra domain routing to another with routing stability. However, these work all lacks of an architectural solution to improve the programmability of all kinds of services.

And (3) a few work has addressed the programmability via distributed control in OF/SDN[3][4][26][10], adopting the distributed system oriented control with better flexibility than the dynamic routing control[2][5][18]. Onix realizes the programmable control platform to support the diverse network services like DCN and IP routers etc. by treating the networking problems as the distributed system problems[3]. However, the dynamic programming process is not controlled for the services of DCP, thus decreasing the service availability; in addition, the distributed services cannot share their control states to lower their design complexities.

Hence, none of these researches addresses programmability of the distributed network control via enabling the reconfigurability of the data plane states and the reusability of control states. Our work brings new direction of research.

VI. Conclusion

In this paper, we introduced two core features, the reconfigurability of data plane states and the reusability of the service control states, to reduce the complexity of design, implementation and update forwarding states in handling physical network dynamics in DCP. The two features are realized in the GRACE layer. Our case analysis demonstrated how to program a network service using the two features. The simulation and emulation results further validate the benefits of having reconfigurability and reusability by showing performance gains under harsh conditions. Together, they offer in-depth understandings on the two concepts and their usefulness for future Internet design.

Acknowledgment

This work is supported by the National Basic Research Program of China (973 Program) (2012CB315903), the Key Science and Technology Innovation Team Project of Zhejiang Province (2011R50010-05) and the National Natural Science Foundation of China (61379118 and 61103200). This work is sponsored by the Research Fund of ZTE Corporation, and also supported in part by the BBN/NSF Project 1783.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan *et al.*, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[2] N. Feamster, H. Balakrishnan, J. Rexford *et al.*, "The case for separating routing from routers," in *SIGCOMM*, 2004.

[3] T. Koponen, M. Casado, N. Gude *et al.*, "Onix: A distributed control platform for large-scale production networks," *OSDI*, 2010.

[4] C. Rothenberg, M. Nascimento, M. Salvador, C. Corrêa, S. Cunha de Lucena, and R. Raszuk, "Revisiting routing control platforms with the eyes and muscles of software-defined networking," in *SIGCOMM*, 2012.

[5] J. John, E. Katz-Bassett, A. Krishnamurthy, T. Anderson, and A. Venkataramani, "Consensus routing: The internet as a distributed system," in *USENIX*, 2008.

[6] L. Vanbever, S. Vissicchio, C. Pelsser *et al.*, "Lossless migrations of link-state igps," *IEEE/ACM Transactions on Networking*, vol. 20, no. 6, pp. 1842–1855, 2012.

[7] U. Hengartner, S. Moon, R. Mortier, and C. Diot, "Detection and analysis of routing loops in packet traces," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. ACM, 2002, pp. 107–112.

[8] N. Gude, T. Koponen, J. Pettit *et al.*, "Nox: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[9] D. Erickson, "The beacon openflow controller," in *ACM SIGCOMM Workshop on HotSDN*, 2013.

[10] P. Lin, J. Bi, and H. Hu, "Asic: an architecture for scalable intra-domain control in openflow," in *Proceedings of the 7th International Conference on Future Internet Technologies*. ACM, 2012, pp. 21–26.

[11] V. Jacobson, D. K. Smetters, J. D. Thornton *et al.*, "Networking named content," in *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, 2009.

[12] M. J. Freedman, "Experiences with coralcdn: A five-year operational view," in *USENIX NSDI*. USENIX Association, 2010, pp. 7–7.

[13] "Openflow switch specification, version 1.4.0," *Open Networking Foundation, http://www.opennetworking.org*, 2013.

[14] A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnsim: Ndn simulator for ns-3," *Univ. of California, Los Angeles, Tech. Rep.*, 2012.

[15] B. Chun, D. Culler, T. Roscoe *et al.*, "Planetlab: an overlay testbed for broad-coverage services," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 3–12, 2003.

[16] L. Lamport, "Byzantizing paxos by refinement," in *Distributed Computing*. Springer, 2011, pp. 211–224.

[17] I. Stoica, R. Morris, D. Karger *et al.*, "Chord: A scalable peer-to-peer lookup service for internet applications," in *ACM SIGCOMM*, 2001.

[18] A. Greenberg, G. Hjalmtysson, D. Maltz *et al.*, "A clean slate 4d approach to network control and management," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 41–54, 2005.

[19] V. Gurbani, V. Hilt, I. Rimac, M. Tomsu, and E. Marocco, "A survey of research on the application-layer traffic optimization problem and the need for layer cooperation," *IEEE Communication Magazine*, vol. 47, no. 8, pp. 107–112, 2009.

[20] A. Medina, A. Lakhina, I. Matta *et al.*, "Brite: An approach to universal topology generation," in *MASCOTS 2001*. IEEE, 2001, pp. 346–353.

[21] C. Monsanto, N. Foster, R. Harrison, and D. Walker, "A compiler and run-time system for network programming languages," in *SIGPLAN*, vol. 47, no. 1. ACM, 2012, pp. 217–230.

[22] N. P. Katta, J. Rexford, and D. Walker, "Incremental consistent updates," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 49–54.

[23] P. Francois, M. Shand, and O. Bonaventure, "Disruption free topology reconfiguration in ospf networks," in *INFOCOM*. IEEE, 2007, pp. 89–97.

[24] S. Raza, Y. Zhu, and C.-N. Chuah, "Graceful network state migrations," *IEEE/ACM Transactions on Networking (TON)*, vol. 19, no. 4, pp. 1097–1110, 2011.

[25] L. Vanbever, S. Vissicchio, C. Pelsser, P. Francois, and O. Bonaventure, "Seamless network-wide igp migrations," in *SIGCOMM*, 2011.

[26] D. Levin, A. Wundsam, B. Heller *et al.*, "Logically centralized?: state distribution trade-offs in software defined networks," in *SIGCOMM*, 2012.