Algorithms for Distributed Programmable Controllers

Boyang Zhou and Chunming Wu
Institute of Computer System Architecture
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 Software and Intelligent Technology
Hangzhou Dianzi University
Hangzhou 310027, China
jmzju@163.com

***Abstract*—A few works on SDN (Software-Defined Networking) like those in Onix improve programmability of the distributed network control. The asynchronism and Byzantine issues of the control challenge the re-configurability of the service that is to safely program the control in atomic so as to avoid the transient control issues like the routing loops and black holes. We propose two important algorithms of the distributed control to enable the programmability: (1) the reconfiguration primitive allows the network control of the services being able to safely react to an external event; and (2) the reuse primitive allows the control states of a service being accessible for all services. We give concepts and algorithms of two primitives. In addition, we provide the concrete cases of the current approaches for ICN (Information-Centric Networking) and CDN (Content Distribution Networks) for quests of the reconfigurability and programmability. Then, we evaluate the performance of ICN in both simulation and the PlanetLab testbed. The evaluation results show that the layer improves the lowers 19.6% of the Interest delays in the ICN that is heavily congested and lowers 97% delays in the PlanetLab with 9 nodes on usual case. In addition, the evaluation of CDN on the PlanetLab shows that it reduces 81% request delay on usual case.**

***Index Terms*—Software-Defined Networking, Distributed Network Control, Algorithms**

I. Introduction

OpenFlow (OF) improves the network programmability by isolating the control plane from the data plane and by pushing the network control complexity to the controller soft-ware. It uses a centralized controller for programing a specific service and control a set of switches in a domain to act on data forwarding. The simplicity of OpenFlow via a centralized programming and configuration makes OpenFlow it a high favorable for SDN since its first proposal in 2008[1]. Recent work such as NetCore[2] and Lithium[3] further improves the programmability of the centralized OF controller by abstracting data plane functions and by providing safer runtime.

The OpenFlow/Software-Defined Networking (OF/SDN) paradigm is currently a popular programmable architecture. It allows a programmer or an operator to program their network and to deploy innovative network service in control plane. It improves the network programmability by isolating the control plane from the data plane and by pushing the network control complexity to the controller software. It uses a centralized controller for programing a specific service logic and control a set of switches in a domain to act on data forwarding. For example, Figure 1 shows four domains to form a distributed control plane to support the services. NOX[4] / Beacon[5] Cores shown are the network operating systems (NOSs, also named as the SDN controllers) that abstract, control and manage resources of both data plane and controllers hardware for the services. In SDN, the control plane consists of a high-level service logics layer and a low-level network control layer. The service logics are realized though communications between the controllers using the data plane switches they control. The network control actions configure the switches to execute the forwarding logics of the data plane according to the service logic.

Distributed control plane in SDN (DCP) has caught great attentions recently. The work of Onix compared the network control plane problem as a distributed system problem, giving network designers freedom and simplicity to refactor the control plane like that it were taken in a centralized device, rather than a distributed system. Researches on Internet architecture such as RCP[6] and consensus routing[7] also show that solving the network control problem from the distributed system point of view can improve flexibility and configurability.

Programming with DCP is the process of innovating and maintaining services in DCP. Especially, the low-level network control of services is hardest to be designed and maintained. Because it can be significantly complicated by asynchrony and instability of the control states at the controllers, as the controllers can undergo physical device or link failures, software or configuration malfunctions and highly unbalanced traffic load. They lead to a critical issue, i.e, the states of control become uncontrollable when the transient states are generate during a series of configurations at different controllers at run time. Such an issue can trigger further issues such as routing unavailability, flow interruption and security holes. These latter problems have challenged the current Internet routers, e.g. research has shown that the transient unavailability of ASes in BGP contributes to 90% packet loss.

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, which limits the programmability of DCP in supporting novel network services.

In this paper, we propose and analyze two properties that the distributed control states must hold for programmability. They are state variability and reusability. The state variability states that the states of the distributed network controls should be safely reconfigurable (i.e. to avoid the transient control states during a reconfiguration, referred as reconfigurability of the control). The steps of programming the control logics are interpreted as series of sequential configurations on the controllers. And to realize the programmability, the transient control states should be coped with in a process of enforcing those steps and the process should be controllable. The programming process is able to be controlled and realized by enforcing the configurations over the controls . For example, a step of setting up filters in firewalls is safely enforced by reconfiguring its underlying firewall control system.

We then provide a brief description on an abstract supervision layer. The layer sits between the controller (i.e, the network operation system) and network service (see Fig.1). It implements mechanisms to handle the dynamics states due to physical network changes or operational changes and the dynamics of the network control. And it which offers the two features for the services, namely, re-configurability of the control state to achieve high availability and reusability of the control state to leverage the system complexity of the control.

The core interfaces that the supervision layer provides are primitives to achieve consistency of control states and safety of control state variations of a service. In this paper, we introduce two algorithms of reconfiguration and reuse primitives to achieve variability and consistency of the control states, respectively. Specifically, the reconfiguration algorithm is to realize the safe variation process of control states, i.e. to eliminate the transient states in the process. The algorithm is executed in the distributed way by all service nodes, making the variation process is transactional and atomic. We implement the algorithm based on Paxos[8] and DHT[9]. The reuse algorithm enables the control states of a service to be accessible by for network controls of all services. We implement the algorithm based on DHT. The two algorithms are implemented on the popular SDN controllers, namely, NOX and Beacon, and also NS-3[10].

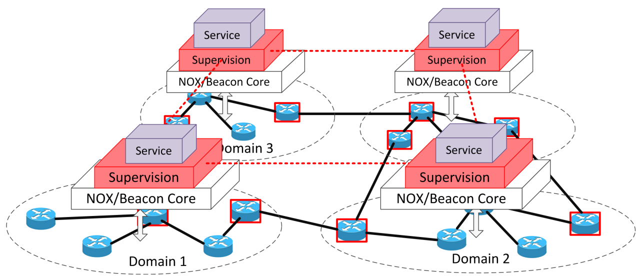

Fig. 1. Distributed Programmable Network Control in SDN

Further, the paper shows how to program new network services using distributed control in SDN with the support of and the supervision layer. Three novel network services are used to illustrate the programmability of the distributed network control in SDN. They are Information-Centric Networking (ICN), Content Distribution Networks (CDN) and Inter-Domain Routing (IDR). Supervised ICN shows that its control states (PIT table ?) can be reconfigured safely in the presence of link congestions, lowering 19.6% Interest delays in the simulation and 97% in the PlanetLab. Supervised CDN shows that path decisions are made better for the content requests, reducing 81% request delay in the PlanetLab. In addition, we illustrate that the ICN and CDN function of updating routing information reuses the function module of the IDR, a routing protocol we design and implement for the distributed control routers. These cases show that the steps of a re-configuration can be safely enforced so as to overcome the performance bottleneck, with its unique and efficient primitives. The three cases are evaluated in both NS3 ndnSim simulator and PlanetLab to show its advantages. In both evaluation platforms, the supervisions layer is implemented.

These contents are organized as follows. Two problems of DCP are discussed in Section II. The reconfiguration and reuse algorithms of DCP are introduced in Section III. Section IV gives implementation, examples and evaluation of the algorithms. Section V concludes the paper.

II. Problem Statements

In this section, we specifically discuss the control state variation and reuse problems.

A. Control State Variation Problem

The control states of the distributed control plane (DCP) can face the same problem as BGP does, i.e. the states are very hard to be reconfigured because the resulting transient states can disturb network operation and decrease performance. With BGP, research has shown that the transient unavailability of ASes in BGP contributes to 90% packet loss. Such hardness increases the complexities in developing a safe distributed control plane, and deteriorates the flexibility in its programmability. Thus variability is one of the design goals for DCP in SDN. It describes the issue of safely transiting from one view of the distributed states to the next view. Better variability should safely eliminate the transient packet loss, routing black hole and routing loops.

The reconfigurability of the distributed control plane in SDN enables the variability. Specifically, the SDN controller provides an underlying mechanism to ensure controllability for the state varying process of the control and to deal with the transient control states. So the configuration for a control system can be easily realized to remove the hardness of the state variation process for the distributed control.

The technical way to solve the variability issue is to deal with the uncontrollability nature of the state update process of the distributed control. Its caused by asynchronism and instability issues of the execution of the network control, i.e. the transient states between two logical times of an execution of the network control cause packet lost, thus degrading the availability of service. Formally, the transient control issues can be modeled as Eq.1. Given two logical times of a network control of a service denoted as t and $t+1$, and a update for the control causes that two state sequences of the control must be changed between the two times, thus the transient states are defined as the intermediate state sequences for epochs between the two times. In addition, the transient states are generated only if there are at least two states changed between the two sequences.

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

Consider the example shown in Fig.1, the failure state changes of the border routers for different domains result in the routing changes. Such routing changes require the each controller achieves the state consistency of their domain based on DHT and then compute a new routing based on the current states. Such transition causes the routing control experiences the transient states between two logical times of old and new routing. Further to say, it results in the possibility of the loss of in-flight packets. In addition, the malicious packets are possible to be forwarded during the transient state, causing the security issues. Our approach to solve this issue is to conceal the complexity of the distributed update logics that deal with the transient state and use the reconfiguration primitive of the supervision layer to perform the update. Specifically, the concept of the primitive is to control the service during the transient state and operate the transient state in another control system (see Section IV.B).

B. Control State Reuse Problem

The application layer traffic engineering (ALTO) techniques in the current Internet like those in P4P[11] requires the application to cooperate with the network layer. However, as the routing controls are dynamically computed and their routings are invisible to the applications. Thus, its hard to accurately predict the forwarding costs.

The control states of DCP can face the same problem as ALTO does. The distributed control states of a service are hard to be consistently captured by other services. As such, allowing a service to reuse the control state of a service should be necessarily support to improve the programmability. And exposing the network control state helps the performance of the service easier to be optimized by the control dynamics for the traffic.

III. Algorithms of Distributed Programmable Controllers

In this section, we first give efficient data structures for representing the control states in SDN controllers. Then, we give concepts and algorithms of reconfiguration and reuse primitives.

A. Data Structures of Control States

The control states are maintained as a state object in the supervision layer. Specifically, the state object is a generalized data structure to hold a set of states and further realized by a hash table with versioning, which is composed of tuples of the element of ¡key, value¿ and a version. In detail, the key and value are the state name and value, respectively, e.g. latency and 200ms. Formally, the model is denoted as the Eq.2. where t is an epoch, and ℓ, Θ and λ are the set of state object identifier, key-value pairs and version, respectively, and in addition it satisfy the criteria: for an epoch t and $t+1$, $\lambda^{t+1} = \lambda^t + 1$ if and only if $\Theta^t \neq \Theta^{t+1}$.

$$\omega_\ell^t \equiv \left(\ell, \Theta_\ell^t = \left\{\left(k_1^t, v_1^t\right), \left(k_2^t, v_2^t\right), ..., \left(k_{|\Theta^t|}^t, v_{|\Theta^t|}^t\right)\right\}, \lambda_\ell^t\right) \quad (2)$$

And given a service, a quorum is a set of membership of the distributed state objects, which is defined as the set of the state object references that refer to their state objects in the service. Formally, we denote a quorum as the Eq.3, where $u_1, u_2, ..., u_m$ is a sequence of the node identifiers with which the controllers in the quorum identify each of state object in the quorum by its own object reference, n_k is the node identifier with which the core identify where the quorum is currently stored and ℓ is the quorum identifier.

$$\Omega_\ell^{t,n_k} = \{\omega^{t,u_1}, \omega^{t,u_2}, ..., \omega^{t,u_m}\} \quad (3)$$

Overall, to reflect the dynamics variations of the distributed states of a service when the service is executing, the core treats the variations in the way that the service are realized and performed in the I/O automata model. In addition, such model is used to capture the distributed states and collaboration actions of distributed processes[12]. In detail, to leverage the model complexities, the states and actions of a service are defined as the snapshot of states of a quorum and the actions of callings of the dynamics primitives for the nodes in the quorum for the service. In addition, the execution sequence from epoch 0 to h of the service is seen as a distributed process in which the states and actions are intertwined with each other. Thus, the logical time of a quorum is a sequence of versions of all ordered state objects in the quorum, which is represented in the WelchTime model[13]. Formally, to further discuss the dynamics primitives, a dynamics process is denoted as the Eq.4, where S_ℓ^t and β_ℓ^t are the states and actions of all state objects in a quorum at epoch t, respectively, and version of the quorum is denoted as $\mathrm{K}_\ell^t\left(\tilde{\omega}_\ell^t\right)$ in Eq.5.

$$\alpha_\ell\left(0, h\right) \equiv \left(S_\ell^0, \beta_\ell^1, S_\ell^1, \beta_\ell^2, ..., \beta_\ell^h, S_\ell^h\right) \quad (4)$$

$$\mathrm{K}_\ell^t\left(\omega_\ell^t\right) \equiv \left(\lambda_\ell^{t,n_{u_1}}, \lambda_\ell^{t,n_{u_2}}, ..., \lambda_\ell^{t,n_{u_{|\Omega|}}}\right) \quad (5)$$

B. Control State Reconfiguration Primitive

The variability ensures safety of state variation process to avoid the transient states in varying. And the process is seen as a sequence of reconfigurations. The technical way to solve the variability issue is to deal with the uncontrollability nature of the state update process of the distributed control.

The control is reconfigured if it needs to distributed update its states by an external event. Such event has two types: (1) the event generated by the control itself like timeout for routing convergence, and (2) the event generated by other controls, e.g. a link congestion event from a routing control can trigger the control of ICN to update the content routing.

The procedure of the reconfiguration primitive is shown in Alg.1. The algorithm is executed in a node of a service by which an external event is triggered (named as the origin node). The execution is distributed taken on all nodes of a service and takes the following steps: (1) The origin node updates the global state to 1 that indicate a reconfiguration is requested for all nodes in the service; (2) all nodes hang up the current program and then perform the transient control; (3) the reconfiguration is taken after all nodes have finished their transient controls; and (4) all nodes restore the transient control and then resume their program. Formally, we provide the algorithm in Alg.1, and consider to use the spanning tree for multicasting, the time and message complexities of the algorithm are $O(logN)$ and $O(N)$, respectively.

Algorithm 1: Pseudo Code of Reconfiguration Primitive

```
input : $ReqRecfState$: 0 or 1 to indicate whether a
        reconfiguration is running for the current service
        $RecfState$: the global consensus state to indicate
the process of a reconfiguration

//Program below distributed executes over the
supervision layer for each controller
$lock \leftarrow 0$
Upon receiving $ReqRecfState$:
Mutex.lock($lock$)
if $ReqRecfState = 1$ and $InExecuting = 0$ then
    $InExecuting \leftarrow 1$
    Hangup program of current service node
    Send and wait transient control event of service
    Paxos.propose($RecfState$, $entering_recf$)
    if Paxos.accept($RecfState$) != $entering_recf$ then
        Resume the program and rollback
        Mutex.unlock($lock$)
        return
    Send and wait performing recf. event of service
    Paxos.propose($RecfState$, $leaving_recf$)
    if Paxos.accept($RecfState$) = $leaving_recf$ then
        Send and wait finish. recf. event of service
        Paxos.propose($RecfState$, $leaving_recf_final$)
        Paxos.accept($RecfState$)
    Resume the program
    Mutex.unlock($lock$)
```

There are three stages for the service to realize their reconfigurability which are (see Fig.2): (1) the transient control entering stage to realize the temporary resource control for the reconfiguration; (2) the reconfiguration stage to realize the changes of the control; and (3) the transient control leaving stage to realize the resource control that restores the resources for the reconfiguration.

C. Control State Reuse Primitive

Reusability realized by the reuse primitive enables the control states accessible for all network controls. Specifically, the algorithm of the reuse primitive works in the two modes to promote responsiveness of the control: (1) The active mode allows a quorum of nodes of a control (i.e. a set of node) consistently retrieving states of a control; and (2) The passive mode allows a control consistently listening on events of state changes of a control. The reuse primitive is realized based on DHT. Its used to store the control states and allows the states being stored and retrieved[9]. And the event triggers in DHT are realized to allow the control registering state criteria for states of a quorum of nodes and receive states update of the quorum if the states satisfy the criteria[14].

Further, the primitive deals with the two factors of networks: (1) The asynchrony causes time differences of state retrieval processes, which can ruin the consistency. To deal with those issues, the Chandy-Lamport algorithm[15] to snapshot the consistent states of the requested quorum is executed before getting a quorum of DHT values. In detail, the algorithm is executed on all nodes that send a marker to a quorum of nodes to snapshot their local state and broadcast the marker to its un-broadcasted nodes. For efficiency, routing of marker is decided on the finger table of DHT for a specific DHT key; and (2) The instability of networks can cause execution failures of DHT or ChandyLamport. The issue is pushed to Paxos[8] that is finally called so as to make consensus on the requesting quorum for the states.

IV. Implementation, Examples and Evaluation of Algorithms

In this section, we first discuss implementation of the reconfiguration and reuse primitives. Then, we provide and evaluate a set of concrete examples of DCP to showcase the performance gains of the primitives.

A. Implementation of Algorithms

The supervision layer sits between the distributed service and the SDN controllers (see Fig.3). It enables the reconfigurability of the services to avoid the transient control issue and also enables the reusability of the network controls so as to improve the control performance according to states of other controls. In detail, the layer is constituted with two parts:

- The I/O automation based programming model is provided for the upper distributed service. In the model, the parallel tasks running in a single machine all make the transition of their program state triggered by an input to the tasks where the distributed tasks run in a network as a distributed system. Each task is encapsulated as a component. Network states of the service are distributed controlled by the network control of the service to achieve a control objective.
- The supervision core realizes the primitives and supervises the resources allocated for the services. The core separates the network control of the services from the physical networks. To support the services, the core realizes and provides three types of APIs for the network control: (1) The consistency primitive achieves the distributed state consistency for all nodes of the service. States of

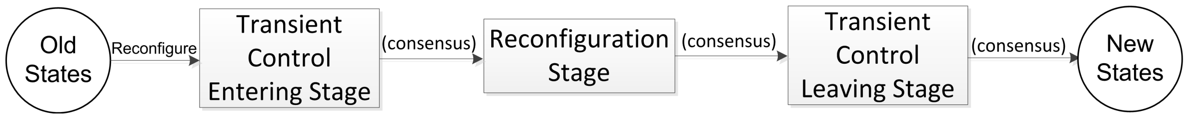

Fig. 2. Stages of Control State Reconfiguration

the controls are stored in the distributed hashing table (DHT) of the layer that allows the states to be stored and retrieved in consistency[9]. In detail, the DHT in the core provides the common get/put APIs and also provides an event triggers. The trigger allows the control registering a state criteria and a callback to a quorum of nodes and the control receive states update of the quorum if the states satisfy the criteria; (2) The consensus primitive makes the distributed value that are consented by all nodes in a quorum of nodes. In detail, its realized based on the Paxos algorithm[8]; (3) The reconfiguration primitive safely performs a configuration of the service, e.g. configuring the flow table of switches ; and (4) the reuse primitive allows the current service to be able to access the states of the other services. To integrate with SDN controller, the core provides the standardized APIs that are allowed to program the flow table of OpenFlow switches and configure the states of the switches.

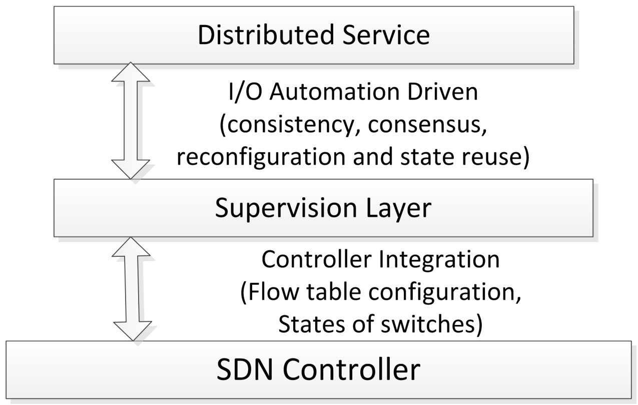

Fig. 3. Design of Supervision Layer

B. Example of Inter-Domain Routing

The Inter-Domain Routing service (IDR) is developed for communication among distributed controllers of in SDN, where each controller controls the switches in its domain. It adopts the idea of consistency-based network routing, which has shown better flexibility than the common dynamic Internet routing approach[7][16]. IDR builds on the consistency primitives of the supervision 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.

IDR takes three steps to converge to new routes in the distributed manner: (1) Each controller obtains and aggregates the route states of switches in its domain, and determines the IDR updates for propagation; (2) Each controller uses Dijkstra algorithm to compute the domain level routing path. While updating the controllers, IDRuses Paxos to obtain consensus on the condition of whether or not the routing computation is finished for all controllers. As a result, all the controllers routing states are consistent; and (3) Each controller calls the reconfiguration primitive to replace the routing states of its switches. Thus, this step can be safely executed without the transient phase.

During the reconfiguration, the in-packet packets that are injected into the border switches are maintained in the temporary buffers before enforcing the new routing states and then switched out of the buffers according to the new routing after the enforcement.

In detail, the execution of the routing takes the steps as Alg.1. It has five tasks: (1) The first task aggregates the updates of the switches in its domain to minimize the consistency overhead and maintains the topology consistency; (2) The second task distributed computes the consistent domain level routing based on the network topology by adopting the Bellman-Ford algorithm. Further, the Paxos algorithm is used to ensure the resilience against the Byzantine issues[17]; (3) The third tasks perform the transient control before and after the reconfiguration, to ensure safety of the reconfiguration; and (4) The last task to realize the reconfiguration logic that is to safely configure the border routers.

In the Alg. 2, the routing states are updated by three steps: (1) Recent updates of switch states are locally aggregated into the topology for each controller; (2) Each controller consents on the states of topology and compute its own routing; and (3) The leader node call the reconfiguration primitive to reconfigure the routing states for all nodes (the reconfigurability of IDR is ensured to avoid the transient control).

C. Examples and Evaluation of Reconfiguration and Reuse

To optimize performance of other controls, IDR exposes the topology states that are reusable for the controls including ICN and CDN (see Fig.4). Changes of the states trigger ICN and CDN so as to react to the event to perform their control adjustments. Specifically, we discuss them correspondingly.

1) Information-Centric Networking: ICN enables the information providers to deliver contents at the subscribers requests by receiving senders Interest packets. Our ICN is realized based on CCNx[18]. The current works realize the mechanisms in the global fashion: (1) The ICN networks globally choose a node to periodically compute the best paths to the prefix for each origin content name of all nodes by the Bellman-Ford algorithm; and (2) The naming in ICN is performed by a centralized node that is used to periodically probe the content in the networks to ensure the freshness and

Algorithm 2: Pseudo Code of the IDR Service

```
input : $topology$: states in the consistency table
//Program below distributed executes over the
supervision layer for each controller
Upon receiving switch update $m$:
$updateCache \leftarrow updateCache \cap m$
if $updateCache$ is full or timer reaches MRAI then
    Consistency.put($topology$, $updateCache$)
Upon topology update:
  Mutex.lock($lock$)
  $[DP] \leftarrow$ Bellman-Ford($topology.domains$)
  Paxos.propose($routing$, $[topology.domainsDP]$)
  $[topology.domainsDP] = Paxos.accept(routing)$
  if the node is leader then
      Reconfigure(IDR, [topology.domains, D, P])
  Mutex.unlock($lock$)
Upon entering into the transient control:
  Hang up the switching buffers for border switches and
  receive the in-flight packets into the temporary buffer
Upon finishing reconfiguration:
  Resume the switching buffers and release the in-flight
  packets into the switches
Upon performing reconfiguration:
  Configure the border routers with
  $[topology.domains\ D\ P]$
```

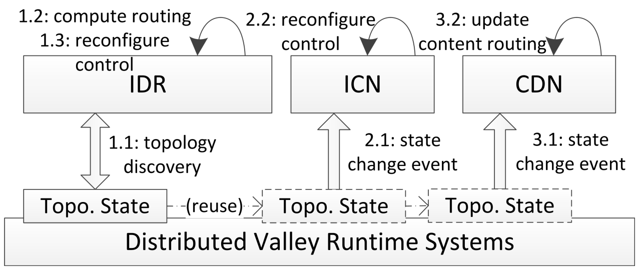

Fig. 4. Control State Reuse in IDR, ICN and CDN

staleness of the content.

However, the heavy content requests make the states of underlying link gone into the link congestions, while the slow routing is hard to capture such highly dynamics of the congestions, decreasing the control performance. Meanwhile, changing the routing causes the ICN control gone into the transient states in which decrease availability of the ICN service.

To deal with such issue, ICN reconfigures its control if the states reveal a link is congested. The content items placement for the switches that is realized by the ICN control should be able to safely and optimally replace the items to satisfy the congestion changes, so as to improve performance by exploiting the distributed nature of ICN. Supervised ICN shows that its control states can be reconfigured safely in the presence of link congestions, lowering 19.6% Interest delays in the simulation and 97% in the PlanetLab (see Fig.5 and 6).

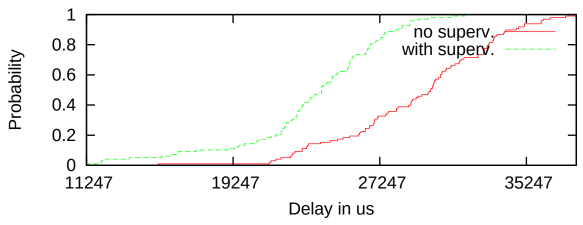

Fig. 5. Performance Comparison of ICN in ECDF of Mean Values of Content Request Delay (simulated by ndnSim[10] with 200 nodes generated by BRITE[19])

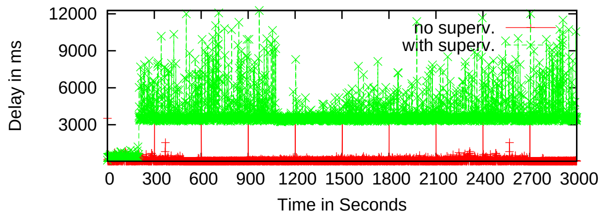

Fig. 6. Content Delays for ICN Nodes on PlanetLab[20] Impacted by Heavy Link Congestion

2) *Content Distribution Networks:* CDN is consisted of a centralized CDN content publisher to manage the content and many CDN content distributors to distribute the content[21]. It realizes the data accessing with better scalability by delivering replica of the content distributors close 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 like constraining upper bound of delay is difficult because it requires a correct choice of content distributors for the requesting user. To deal with such issue, CDN reuse the states of IDR, so that it can update its content routing if the states reveal topology is changed. Supervised CDN shows that path decisions are made better for the content requests, reducing 81% request delay in the PlanetLab (see Fig.7).

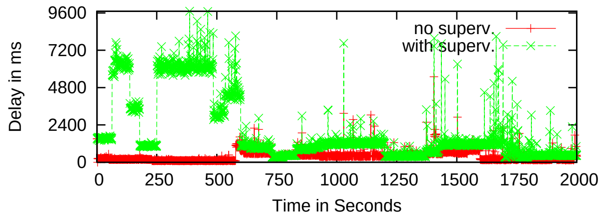

Fig. 7. Content Delays for CDN Nodes on PlanetLab[20] Impacted by Highly Flutuated Congestion States of Links

V. Conclusion

The network control is distributed in nature in requesting to achieve the better scalability, reliability and responsiveness.

In this paper, we introduce reconfigurability and reusability as two critical factors to enable the programmability of the distributed control plane in SDN. We give concept, algorithm and example of reconfiguration and reuse primitives. They allow the network control of the services to be safely configured by the external events like the topology congestions in order to avoid the transient control issues and to improve the network performance.

This work has made a significant contribution to the distributed network control in SDN by introducing for the first time the re-configurability of the distributed services.

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), 863 Program of China (2011AA01A107) and the National Natural Science Foundation of China (61070157). This work was also supported in part by the BBN/NSF Project 1783.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: enabling innovation in campus networks,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[2] 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.

[3] H. Kim, A. Voellmy, S. Burnett, N. Feamster, and R. Clark, “Lithium: Event-driven network control,” 2012.

[4] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, “Nox: towards an operating system for networks,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[5] “Beacon project.” [Online]. Available: https://openflow.stanford.edu/display/Beacon/Home

[6] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. Van Der Merwe, “The case for separating routing from routers,” in *SIGCOMM*, 2004.

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

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

[9] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, “Handling churn in a dht,” in *USENIX*, 2004.

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

[11] 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.

[12] N. Lynch and M. Tuttle, “An introduction to input/output automata,” *Centrum voor Wiskundeen Informatica*, vol. 2, no. 3, pp. 219–246, 1989.

[13] J. Welch, “Simulating synchronous processors,” *Information and Computation*, vol. 74, no. 2, pp. 159–171, 1987.

[14] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, “Dynamo: amazon’s highly available key-value store,” in *ACM Symposium on Operating Systems Principles: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, vol. 14, no. 17, 2007, pp. 205–220.

[15] K. M. Chandy and L. Lamport, “Distributed snapshots: determining global states of distributed systems,” *ACM Transactions on Computer Systems (TOCS)*, vol. 3, no. 1, pp. 63–75, 1985.

[16] N. Kushman, D. Katabi, and J. Wroclawski, “A consistency management layer for inter-domain routing,” 2006.

[17] M. Castro and B. Liskov, “Practical byzantine fault tolerance and proactive recovery,” *ACM Transaction on Computer System*, vol. 20, no. 4, pp. 398–461, 2002.

[18] “Ccnx project.” [Online]. Available: http://www.ccnx.org

[19] A. Medina, A. Lakhina, I. Matta, and J. Byers, “Brite: An approach to universal topology generation,” in *MASCOTS 2001*. IEEE, 2001, pp. 346–353.

[20] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, “Planetlab: an overlay testbed for broad-coverage services,” *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 3–12, 2003.

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