Automatic Life Cycle Management of Network Configurations

Hongqiang Harry Liu, Xin Wu, Wei Zhou, Weiguo Chen, Tao Wang,
Hui Xu, Lei Zhou, Qing Ma, Ming Zhang
Alibaba Group

ABSTRACT

Managing the life cycle of network configurations, including the generation, update, transition and diagnosis of the configurations, is the primary task of network operators and a critical process for the reliability and efficiency of the networks. This paper presents NetCraft, a framework which automates the life cycle management of network configurations with a unified network model. Designed for life cycle automation, NetCraft's network model can expressively encode all parts and protocols in the network; It can be converted to or constructed from configurations with interoperability; It is able to perform fine-grained configurations with flexibility to deactivate or undo any configurations for safe configuration updates; And it can work without cooperations from device vendors. We have built and deployed an initial version of NetCraft in Alibaba's global WAN. Evaluations in real environments show that NetCraft can reduce the network incidents caused by configurations by 95% and cut the average time to plan and execute a network update by up to 93%.

CCS CONCEPTS

• **Networks** → **Network management**; **Network reliability**;

KEYWORDS

Self-driving Network, Configuration, Lifecycle, Automation

ACM Reference Format:
Hongqiang Harry Liu, Xin Wu, Wei Zhou, Weiguo Chen, Tao Wang, Hui Xu, Lei Zhou, Qing Ma, Ming Zhang. 2018. Automatic Life Cycle Management of Network Configurations. In *SelfDN '18: Afternoon Workshop on Self-Driving Networks, August 24, 2018, Budapest, Hungary.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3229584.3229585

1 INTRODUCTION

Managing the configurations on network devices is the core of network management. The properties of a network, such as connectivity, performance, cost efficiency, reliability, and security *etc.*, all depend on the device behaviors defined by the configurations.

Network configurations have a life cycle with the developments of a network. The configurations of a network are initially generated when the network is built and get updated frequently due to network changes, incident mitigation or new application requirements. A configuration update is usually risky to the network, so network operators also need to make a plan to cautiously make the configuration changes step by step without disturbing the network, namely "a smooth transition". In addition, throughout the entire life cycle, network operators also need to diagnosis configurations for purposes like validation, debugging, configuration translation between vendors' languages, configuration diff and so forth.

Unfortunately, managing the life cycle of network configurations, including the *generation*, *update*, *transition* and *diagnosis* of configurations, is notoriously challenging for network operators. On one hand, since production networks are large and heterogeneous, it is hard for human beings to fully understand and trace the effects of network configurations on different devices and protocols; On the other hand, network configurations are frequently updated, and each update is a risk to the network. Operators also need to design a smooth transition without disturbing the network, which is difficult and time consuming. Furthermore, configuration languages of network devices are low level, and their semantics varies in different device vendors, so that operators need tremendous expertise to understand and manipulate them to correctly realize a high level intent, *e.g.,* moving user A's traffic from backbone 1 to backbone 2, *etc.*.

There are numerous efforts recently to automate the configuration management, while they merely focus on some specific part(s) of the whole life cycle. For instance, Propane [5], Propane/AT[6], and Robotron [18] automatically generate network configurations with high level intents input by network operators, while it is not clear how these systems generate incremental changes and transition plans for network update, or how they learn the high level intents by diagnosing existing configurations. CPR [11] automatically proposes incremental configuration changes for network updates or repairs, while the changes are not guaranteed to be transitioned to smoothly. In addition, the configuration changes it generates can be hard for human to understand or reason about. Configuration verification tools, *e.g.,* [4, 9, 10, 12, 15], focus on how to justify the correctness of a snapshot of configurations, but they do not generate configurations, configuration updates or transition plans.

Network operators need the automation of the whole life cycle management of network configurations. On one hand, automating a particular step in the life cycle does not necessarily ensure the network reliability. For instance, even though we can leverage configuration generation tools to create a correct snapshot of network configurations, the risk of network incidents is still significant if we cannot find proper incremental configuration changes or smooth transition plans for network updates (see more discussions in §6.1). On the other hand, any human involvements along the life cycle

SelfDN '18, August 24, 2018, Budapest, Hungary

ACM ISBN 978-1-4503-5914-6/18/08…$15.00
https://doi.org/10.1145/3229584.3229585

will slow down the speed of the network to adapt to the applications' requirements to the network, which is also a critical metric to the efficiency of network management.

We believe the fundamental reason why people cannot fully automate the life cycle of the network configuration management nowadays is the lack of a unified network model which abstracts network configurations. Such a network model should have four essential features for serving the entire configuration life cycle: *(i) Expressiveness*: The model can describe all parts of network configurations, including physical connectivity, IP connectivity, and various of routing protocols and route policies. *(ii) Interoperability*: The model can freely be translated into configurations, or be constructed from existing configurations. This feature is critical for automation. *(iii) Flexibility*: The model can describe fine-grained operations (*e.g.,* creating a peering session, updating a routing policy, *etc.*) in network configurations and can also easily deactivate, activate or undo any configuration module. This feature is needed for performing network updates and smooth transitions. *(iv) Independence*: The model should not require any cooperation from device vendors. This feature is important for the adaptation speed of the model, since standardization among vendors are usually slow.

In this paper, we present NetCraft, a pioneer framework towards an automatic network configuration life cycle management with a unified network model. Network operators using NetCraft only express their network design or update intents via the network model, and the generation, update, transition and diagnosis of the configurations are automatically done by software.

The network model of NetCraft is a multi-layered graph. At the bottom, a physical graph defines the physical connectivity among network devices, and on top of the physical graph there is a IP graph which defines the IP interfaces and the links among them. Each individual routing protocols, *e.g.,* BGP, OSPF, ISIS, MPLS, *etc.*, has its own dedicated graph layer in which nodes are the protocol agents inside each device and edges are the peering among protocol agents. Different layers are connected by their shared entities, *e.g.,* BGP graph and OSPG graph share the interfaces on the IP graph. Furthermore, policies are encoded as properties of nodes, interfaces or edges on certain graph layers.

This network model achieves the four required features for network configuration life cycle automation. First, it distinguishes different network layers and protocols in the network for *expressiveness*; Second, it breaks down configurations into standard and reusable modular templates and associate each of them to a part (*e.g.,* node, interface, edge, or property) on the network model. Therefore, the *interoperability* is achieved naturally since each part of the network model can be easily converted back and forth with a configuration module; Third, during network updates, the scope of each configuration module is only on simple and basic operations, and each module also has several shadow modules for deactivating, activating or undoing its effects to the network (*flexibility*); Finally, it maintains *independence* by directly dealing with vendor-specific configuration semantics without depending on any other configuration standardization (*e.g.,* OpenConfig [2]), data models (*e.g.,* YANG [7]), or configuration protocols (*e.g.,* NetConf [8]).

Surrounding the network model, we have built and deployed an initial version of NetCraft in Alibaba's global scale wide area network (WAN). Preliminary evaluations from real usage shows that NetCraft has helped Alibaba to reduce 95% network incidents caused by configurations, and 93% in average processing time for network updates. We also discuss our on-going effort to improve NetCraft and future research directions related with NetCraft in §6.

Root Cause	**Fraction** (%)
Configuration Update Error	56%
Configuration Generation Error	10%
Device Firmware Bug	10%
Human Error	10%
Transient Error	8%
Unknown	6%

Table 1: Categories of network incidents in Alibaba in 2016 and 2017.

2 BACKGROUND AND MOTIVATION

In this section, we first briefly introduce the global scale network of Alibaba, and then we present some key observations from the daily managements in this network to motivate NetCraft.

2.1 Alibaba's network infrastructure

Alibaba has a global scale infrastructure to support its various types of online services, including online retail, cloud, mobile payment, FinTech, *etc.*, which serves billions of users. This infrastructure has data centers and edge sites across multiple continents [1], and it contains metropolitan area networks (MANs) connecting data centers in the same city and a single wide area network (WAN) connecting MANs in different cities, edge sites and peering with external ISPs. Networks inside data centers are Clos fabrics running OSPF or eBGP for internal routing; Data center networks, MANs, edge sites and the WAN are connected by eBGP; WAN is a single AS that uses iBGP on top of IS-IS to distribute routes. For some particular applications, DC-to-DC traffic is also carried by a SDN controller with Segment Routing.

With the fast growth of the application demands, the size of the whole infrastructure network is almost doubled each year, while the requirements on availability, performance and security of the networks are also rapidly increasing. Therefore, Alibaba is fully motivated to adopt new technologies which potentially enhance the *reliability* and the *agility* to adopt to new application requirements of its network.

In this section, we take the network of Alibaba as a concrete example to uncover the problems in the state-of-art network configuration management, while we believe the problems also apply to most of large online service providers.

2.2 Challenges to manage network configurations

Applications always require the under-layer network to be always reliable and agile to adapt their new requirements. However, these two requirements, reliability and agility, are both hard to achieve.

Reliability: Large scale production networks are usually complicated and hard to configure correctly. Taking Alibaba's network as an example, it is complex and error-prone due to three specific reasons. First, Alibaba has a number of independent businesses that are

sharing a single infrastructure but need to be isolated in networking as required by policies. Thus, there are numerous route-isolated VPNs in the WAN and a large amount of ACL rules between data centers and MANs; Second, for multiplexing the expensive WAN resources (*e.g.*,optical cable), Alibaba decided to carry internal traffic among its own data centers and edge sites and external traffic with Internet on a single WAN and implemented specific security and routing strategies for the two types of traffic; Finally, Alibaba has multiple vendors due to historical and businesses reasons, and each vendor has specific configuration interfaces and features.

Unsurprisingly, such a complex network is error-prone. Table 1 shows a summary of the root causes of customer impacting network incidents of Alibaba in recent two years (2016 and 2017) before we deploy NetCraft. The incidents caused by configurations in total is about 66%, including the mistakes in configuration updates (56%) and configuration bugs triggered by hardware failures (10%).

Obviously, configuration updates are in the highest risk for several reasons. The first reason is that a configuration update is derived from a high-level update intent (*e.g.,* disallowing data center A and B to communicate). There can be mistakes within this translation due to a misunderstanding on the intent, the current network status or the configuration features of some devices. The second reason is that a configuration update often involves numerous devices and multiple protocols, so that there is a need for a smooth transition plan to make the changes step by step without disturbing the network. Transition plans are mainly made by operators from experience, which is not reliable. The third reason is the speed to detect problems and roll back during configuration updates. Operators need time to check numerous statuses to confirm a problem, and, furthermore, planning and executing a smooth roll back is as challenging as a smooth update.

Agility: As we can see, operators need to spend a long time to perform a configuration update for reliability, while sacrificing agility. For instance, in Alibaba's WAN, it used to take about a few days to update all ACL rules for an application and a few hours to assign or recall IP blocks to or from an application. Making things worse, in recent years, the number of configuration updates has doubled annually. It is not scalable anymore to depend on human operators to handle such a huge number of configuration updates and still require both reliability and agility.

Hence, we are fully motivated to explore how to automate the configuration management. Ideally, human operators should only design a network or define an update intent with a high level network abstraction, and the low level details of the configuration generation, incremental configuration update, transition design and configuration diagnosis should be handled well by software. Hence, we design and build NetCraft to realize this vision.

3 DESIGN

This section first overviews the system architecture of NetCraft, and then it elaborates the design of NetCraft's network model and how to leverage the network model for configuration generation, update, transition and diagnosis.

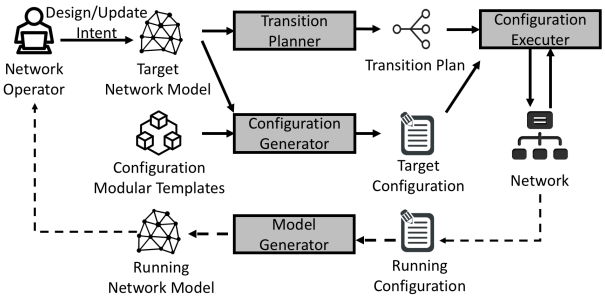


Figure 1: The architecture of NetCraft. Gray boxes are the main function blocks of NetCraft.

3.1 Overview

NetCraft is a software system sitting between network operators and networks and automating the whole life cycle of network configuration management.

As shown in Figure 1, for configuring a network, network operators can make a target network model, which can either describe the whole network or changes to some parts of the network, and push the model to NetCraft. The configuration generator parses the target model and selectively combines pre-defined configuration modular templates to generate the target configuration. At the same time, if the operator is updating the network, the transition planner also computes a dependency graph among all configuration modules, so that executing the modules according to the dependencies will guarantee a smooth transition. Finally, the concrete transition plan will be executed by the configuration executor. The executor performs the configuration modules step by step, and checks the status of the network. It will roll back to the configurations before the transition if it detects any anomalies. Additionally, it is also important to convert configurations back to network models for diagnosing purposes, and the model generator is in charge of it.

The core of NetCraft is the design of the network model and how to design the functional blocks (the gray boxes in Figure 1) to leverage the network model. We will present the details with examples next.

3.2 Network model

The network model of a network is a multi-layered graph. Taking the case shown in Figure 2 as an example, the devices, WAN, DCA and DCB, are physically connected as illustrated in Figure 2(a). This graph shows the connectivity among Ethernet interfaces of the three devices. Also, each device, interface or link can have multiple properties. For instance, `eth1.mtu` explicitly setup the MTU on `eth1`, which will be converted into relevant configuration lines.

On top of the physical graph, Figure 2(b) shows the IP graph. It shows the IP interfaces, which are either port channels (*e.g.,* `pc1` on either WAN or DCA) or ethernet interfaces, and the interface IP assignments as a property of the IP interfaces. Note that device DCA and DCB also has a property `net` which means the IP prefixes they own.

Based on the IP graph, we can design a BGP graph as presented by Figure 2(c). This graph only has two edges which means there are only two BGP sessions, WAN-DCA and WAN-DCB. The AS number of the devices is a device property. The IN and OUT policies are also attached to each BGP peering interface, demonstrating the

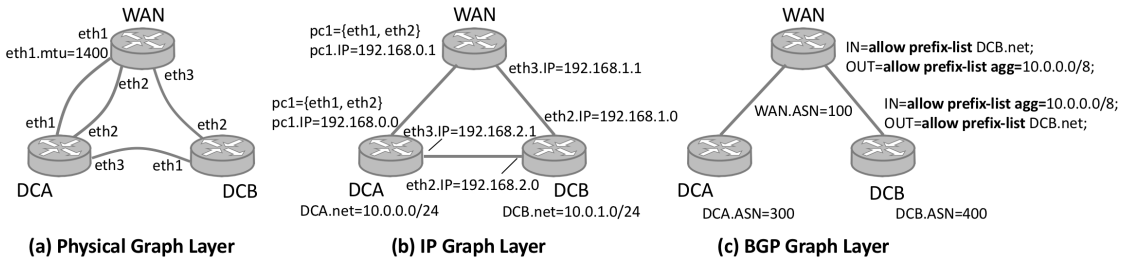


Figure 2: An example of NetCraft's layered network model.

accepted IP lists or actions on some IP lists. For instance, for its BGP peer DCB, device WAN in Figure 2(c) only accepts DCB's IP lists in IN policy, and it allows to announce IP prefix 10.0.0.0/8 with name "agg" in OUT policy.

As we can see, the network model encodes the configurations on physical network topology, IP layer links, IP assignments, BGP peering and BGP policies into the layered graph and the properties within the nodes, interfaces and edges of the graph. It is not only straightforward for operators to understand and manipulate, but also sufficient to generate configurations, as we will see in §3.4.

3.3 Modular configuration template

NetCraft stores a collection of modular configuration templates which are pre-defined by network operators (with some configuration analyzing tools we develop) for various vendor-specific configuration languages. As shown in the example (in Cisco's semantics) in Figure 3(a), each modular only targets a fine-grained operation, such as making a BGP peer, creating a routing policy or define a prefix list. It takes variables from the network model as input, *e.g.,* `PeerIP` in *BGP Peering*.

The modular templates in Figure 3(a) will take effect immediately after being executed on corresponding devices. It is mainly used in configuration generation for newly built networks. However, for configuration updates, it is not safe to do so. Network operators often need to first insert the configurations without activating them, and activate/deactivate as needed. Also, there is also a need to entirely undo the changes or check whether the configuration is successful or not. Therefore, one modular template also has four *shadow modules*. For example, Figure 4 shows the shadow modules of *BGP Peering*: `Deactivate`, `Activate`, `Undo` and `Check`. Shadow modules are useful for constructing smooth transition, as we will see in §3.6.

3.4 Configuration generation

The generation of configurations is a process which associates modular templates with proper entity on the network model. For instance, Figure 3(b) shows the configuration piece of WAN when the configuration generator of NetCraft sees the peering edge between WAN and DCB in the BGP graph shown in Figure 2(c).

At the beginning, the generator calls the BGP peering module and feeds the variables needed by the *BGP Peering* module with concrete values available from the BGP graph. Then, it will find the two policies `DCB_IN` and `DCB_OUT` are not defined yet. It will finds the polices on WAN's peering interface with DCB and calls the *Routing Policy* module to generate the two routing policy configurations, and it will find the prefix lists are not defined yet. This process will continue if the generator still has anything undefined.

Note that the BGP peering example shown in Figure 3 is a simple case. In reality, the dependency among the modular templates might be complicated. The configuration generator needs to use a depth first searching algorithm to enumerate all definitions that are needed.

3.5 Incremental update

NetCraft accepts new network models that are different from the running version in the network to perform a network update. The transition planner of NetCraft takes the two snapshots of network models and fully computes the differences on each layer. There are four potential differences: new nodes or edges are added, nodes or edges are deleted, nodes or edges are updated in properties, policies are changed. For each of the case, the transition planner prepares a configuration module and its shadow modules accordingly.

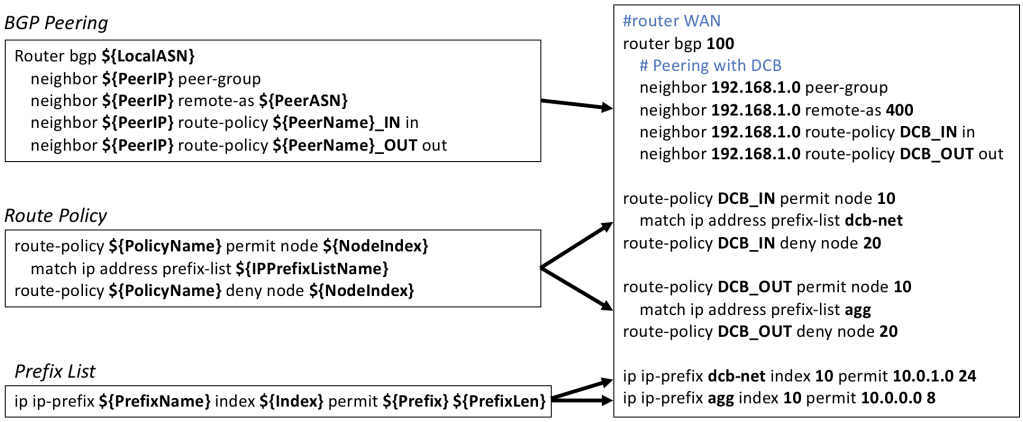
For instance, Figure 5 shows an update from the running model in which DCA and DCB are directly peering with WAN to a target model in which DCA and DCB are peering with MAN, and only MAN peers with WAN. The transition planner generates basically 5 modules for the BGP peering updates: installs peering MAN-WAN, MAN-DCA and MAN-DCB, and turn down peering WAN-DCA and WAN-DCB.

Figure 5 only shows the changes in BGP peering, but in practice there should also be configurations for setting up IP connectivity and route policy changes. NetCraft is able to figure them out from the physical layer at the bottom to the top.

3.6 Smooth transition

NetCraft currently takes three high-level strategies to construct a smooth transition. First, only one update is performed on the network at a given time, and configuration modules within an update are executed one by one without parallelization; Second, the configuration executer checks the status of network until a module is successfully executed before continuing to the next one; Finally, if there is an abnormality in the network during a transition, NetCraft immediately rolls back with the *Undo* shadow modules.

NetCraft uses a heuristic to decide the order of the execution of the configuration modules within an update. First, there are always four stages for each update. At the first stage, all `Deactivate` shadow modules are executed; Then, a customized configuration module is executed to make the configurations in the first stage less preferred; At the third stage, all `Activate` modules are executed; and finally, executing `Undo` of the customized configuration module at the second stage to prefer the newly configured rules and



(a) Modular Templates (b) A piece of WAN's configuration for peering with DCB

Figure 3: Examples of configuration modular templates and how they assemble a configuration.

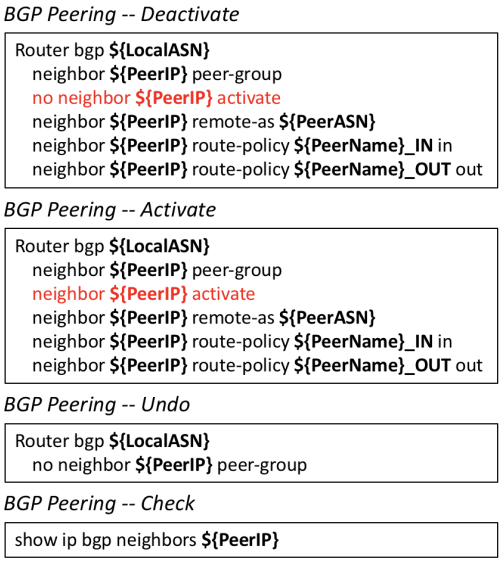


Figure 4: The shadow modules of BGP Peering.

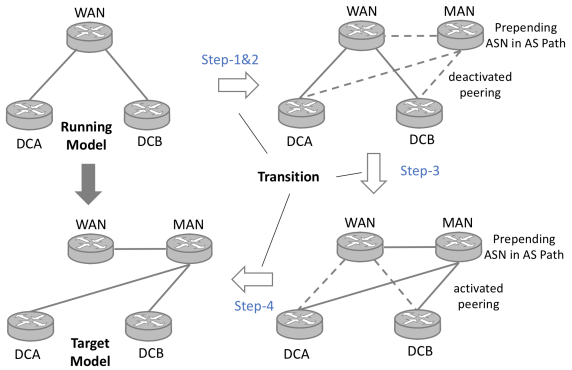


Figure 5: A example of network update and smooth transition. All of the four graphs are BGP graphs.

removing old configurations. For example, the transition shown in Figure 5 exactly follows the four stages. Note that the "Prepending ASN in AS Path" in MAN is the customized configuration module, which cannot be derived from the network models but relies on the input of network operators. Currently we have not found automatic methods to define such customized modules, but some common cases, like the BGP peering updates in Figure 5, can be pre-defined, so that the whole process does not involve Human's interaction.

Within each stage, the configuration modules for lower layer graphs are executed ahead. This is because the lower layer is the foundation of the upper layer, so that the latter can fail without ensuring the former first (see discussions in §6.2).

3.7 Model generation

NetCraft is able to generate a network model from a configuration snapshot even if the configurations are not generated from the network model. NetCraft has a build-in configuration parser which learns the connectivity and peering in routing protocols from configurations. The whole process is similar to the control plane construction in BatFish [10].

The model generator is a useful feature in configuration verification, configuration diff, configuration translation from one language to another, and so on.

4 IMPLEMENTATION AND DEPLOYMENT

An initial version of NetCraft has been implemented and deployed in Alibaba's WAN for 1 year. The whole system contains O(100k) lines of Java code, including the four functional blocks in Figure 1 and tools for building modular templates. Next, we show some key challenges we solve in implementations.

Multi-layered graph model : There are two primary challenges in the implementation of NetCraft's network model. First, despite the conceptual simplicity of the layered graph, pushing standardization among all of our vendors ended up with a massive vendor extensions which requires a complex cross-referencing implementation. We eventually manually analyzed all the vendors' configurations and defined Alibaba's data model that satisfies our requirements. The cost is a translation layer sitting between our data model and vendors' data models. Second, version control is required for various operations mentioned in Section 3, *e.g.*,, configuration diff and roll back. As a result, we implemented multi-version concurrency control underneath the data model, with which we can do time travel across the life cycle of a configuration.

Modular template : Breaking configuration files into modular templates is challenging because it heavily depends on different vendors' semantic designs. This part of the system is not automated. Instead, we have implemented a set of tools to help operators modularize vendor specific configurations. The most important tool is a simple template language, which can use loop and conditional statement to simplify the representation of a collection of configuration modules. This tool helps us significantly decrease both the number of modular templates and the length of a modular template.

5 EVALUATION IN PRODUCTION

Due to space limit, we only highlight some key improvements on network reliability and agility after we deployed NetCraft on Alibaba's WAN.

Coverage : The network updates NetCraft can handle was only a small fraction when it was just deployed. However, after enhancing the modular configuration templates and considering more vendors, most of the network updates have been covered by automation by March, 2018.

Reliability : The metric we use is the probability of network incidents when performing a network update. In the recent year with NetCraft, the number of network updates has doubled, but the probability of network incidents caused by network updates was cut by 18 times compared with the previous year without NetCraft.

Agility : Here are two case studies. *Case-1*: The average time spent to on-board a device onto the WAN was cut by 83% (from hours to minutes) after using NetCraft. *Case-2:* Due to the shortage of IPv4 addresses, there are usually updates to recycle the unused IP addresses, including cleaning all relevant configurations in all devices. The average processing time was cut by 92% after we deployed NetCraft (from hours to minutes).

6 DISCUSSION

In this section, we discuss some practical considerations that lead us to the design of NetCraft, the limitations of current version of NetCraft, and the future work to enhance NetCraft.

6.1 Practical considerations

Hot Restart v.s. incremental update : Ideally, a configuration update can be done by hot restarting all devices with a brand new configuration snapshot, without the need of incremental updates. However, in reality, incremental updates are indispensable because high end routers, especially WAN routers, are designed for "running forever" and could take tens of minutes to reboot with a risk of reboot failures. Even for low end switches, like in data centers, incremental updates are still preferred [18] because they have less impacts to the network.

Relying on CLI : One critical design decision is to make the network model independent with vendors by merely using the devices' command line interfaces for configuring the devices. Despite protocols like NetConf [8] offers better interfaces for configuration updates (*e.g.,*), only a small number of vendors have fully deployed the features after the protocol has been released for years.

6.2 Current limitations and future work

Transition planning : The heuristic for transition planning in Section 3.6 only works for simple cases, and the majority of transition plans are not entirely automatically generated at this stage. Instead, operators need to write a state machine for guiding the transition in complex updates. In addition, the strategy to always configure from lower layer might be too aggressive in updates for failure mitigation, such as turning down a link.

In the future, we will study how to make the transition planning more automatic. We need to have a deeper understanding about the dependency in the configurations of different protocols and network topology, and develop a theoretical guarantee on the smoothness of the transitions.

Multi-update scheulding : The current multi-update scheduling only allows one update at a time, which might be too conservative. We will develop a better strategy for scheduling multiple updates on a network, including how to make an update atomic, how to coordinate different updates and how to judge the dependencies and inferences between two network configuration updates.

Model & configuration simplification : Currently, the multi-graph layered graph model can be huge and hard for human to trace and understand in large networks. We are looking for methods to "zoom-out" the model by summarizing the model in different granularity. In addition, we will try to study how to make equivalent changes on the model, which improves the simplicity of network configurations without changing the intents of the configurations.

7 RELATED WORK

Configuration generation : We have discussed Robotron [18], Propane[5] and Propane/AT[6] in §1. In addition, PGA [16] and Janus [3] use abstract graph models to generate policy and QoS configurations. The key difference between these works and NetCraft is that the latter's model can be used for planning smooth transitions (with roll back) and has interoperability for configuration diagnosis. Compared with Propane, Propane/AT, PGA and Janus, NetCraft targets all network configurations rather than a special domain.

Smooth transition in network updates : zUpdate [14] and Dionysus [13] proposes abstractions and algorithms for planning smooth transitions on data plane rule changes in a software-defined network (SDN) context. Statesman [17] is a management system for data plane states in data center networks. It schedules concurrent data plane updates and solves the dependencies and conflicts of them. NetCraft focuses on networks running traditional distributed network protocols and the smooth transitions on control plane configurations.

8 CONCLUSIONS

We present NetCraft, a pioneer effort to build a unified network model for automating the life cycle of network configuration management. NetCraft's multi-layered graph network model can automate the configuration life cycle with expressiveness, interoperability, flexibility and independence. Real-life valuations show that NetCraft can significantly reduce network incidents and the processing time of network updates. NetCraft is still at an early stage, and we propose several research directions to enhance it.

REFERENCES

[1] AliCloud Global Locations. https://www.alibabacloud.com/global-locations?spm=a2c63.p38356.a3.1.6dac4020d9ARAW.

[2] OpenConfig. http://www.openconfig.net/.

[3] A. Abhashkumar, J.-M. Kang, S. Banerjee, A. Akella, Y. Zhang, and W. Wu. Supporting diverse dynamic intent-based policies using janus. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 296–309. ACM, 2017.

[4] R. Beckett, A. Gupta, R. Mahajan, and D. Walker. A general approach to network configuration verification. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 155–168, New York, NY, USA, 2017. ACM.

[5] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker. Don't mind the gap: Bridging network-wide objectives and device-level configurations. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 328–341, New York, NY, USA, 2016. ACM.

[6] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker. Network configuration synthesis with abstract topologies. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2017, pages 437–451, New York, NY, USA, 2017. ACM.

[7] M. Bjorklund. YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020.

[8] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman. Network Configuration Protocol (NETCONF). RFC 6241.

[9] S. K. Fayaz, T. Sharma, A. Fogel, R. Mahajan, T. Millstein, V. Sekar, and G. Varghese. Efficient network reachability analysis using a succinct control plane representation. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 217–232, Berkeley, CA, USA, 2016. USENIX Association.

[10] A. Fogel, S. Fung, L. Pedrosa, M. Walraed-Sullivan, R. Govindan, R. Mahajan, and T. Millstein. A general approach to network configuration analysis. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI'15, pages 469–483, Berkeley, CA, USA, 2015. USENIX Association.

[11] A. Gember-Jacobson, A. Akella, R. Mahajan, and H. H. Liu. Automatically repairing network control planes using an abstract representation. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 359–373. ACM, 2017.

[12] A. Gember-Jacobson, R. Viswanathan, A. Akella, and R. Mahajan. Fast control plane analysis using an abstract representation. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 300–313, New York, NY, USA, 2016. ACM.

[13] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer. Dynamic scheduling of network updates. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 539–550. ACM, 2014.

[14] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, and D. Maltz. zupdate: Updating data center networks with zero loss. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 411–422. ACM, 2013.

[15] H. H. Liu, Y. Zhu, J. Padhye, J. Cao, S. Tallapragada, N. P. Lopes, A. Rybalchenko, G. Lu, and L. Yuan. Crystalnet: Faithfully emulating large production networks. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 599–613, New York, NY, USA, 2017. ACM.

[16] C. Prakash, J. Lee, Y. Turner, J.-M. Kang, A. Akella, S. Banerjee, C. Clark, Y. Ma, P. Sharma, and Y. Zhang. Pga: Using graphs to express and automatically reconcile network policies. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 29–42. ACM, 2015.

[17] P. Sun, R. Mahajan, J. Rexford, L. Yuan, M. Zhang, and A. Arefin. A network-state management service. *ACM SIGCOMM Computer Communication Review*, 44(4):563–574, 2015.

[18] Y.-W. E. Sung, X. Tie, S. H. Wong, and H. Zeng. Robotron: Top-down network management at facebook scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 426–439. ACM, 2016.