CLICK-UP: Towards Software Upgrades of Click-driven Stateful Network Elements

Junxiao Wang, Yuchen Huang, Heng Qi, Keqiu Li
Dalian University of Technology

Steve Uhlig*
Queen Mary University of London

ACM Reference Format:
Junxiao Wang, Yuchen Huang, Heng Qi, Keqiu Li and Steve Uhlig. 2018. CLICK-UP: Towards Software Upgrades of Click-driven Stateful Network Elements. In *Proceedings of SIGCOMM Posters and Demos '18*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

In production scenarios of Network Function Virtualisation (NFV), active network elements typically come with software upgrades, e.g., when off-the-shelf become inappropriate in terms of their function or simply when evolving components. Therefore, some network elements have to be re-formulated and adapted to better fit their new purpose. With the component modularity inherent to VNF comes the opportunity to replace traditional (partly) hardware-based upgrades with software ones, increasing cost-efficiency by allowing technological and software improvements to be included faster.

The Click modular router [3], thanks to its design, has been one of the most popular NFV platforms, allowing flexible composition of packet-processing functions. It has been used in numerous applications, such as a routers for future Internet architectures, redundant traffic elimination systems, scalable middlebox platforms, and as a cluster-based high performance software router, just to name a few.

The key strength of Click is its inherent extensibility for functional components: a new feature can be easily implemented by module integration. While Click's flexible design has satisfied the demand for rapid prototyping, its internal architecture has unfortunately not kept up with the specific problem of software upgrades. Traditional Click-driven upgrades for network elements have the following ***drawbacks***:

D1: Integrating new modules with upgraded network elements is a time-consuming process. During this process, however, the packet-processing functionalities are out-of-work. This brings several issues including the inability to elastically scale out network functions on-demand and to quickly recover from downtime.

D2: With no mechanism to reconstruct lost states for network elements, stateful functionalities are unable to correctly handle packets after upgrade, leading to service disruption. This may involve states such as connection information in a stateful firewall, substring matches in an intrusion detection system (IDS), address mappings in a network address translator (NAT), or server mappings in a stateful load balancer.

D3: The development of new modules requires intimate knowledge of complex library implementations, let alone correctly deal with states. The current visibility of libraries is poor, and the learning curve to understand them is rather steep, making upgrades generally frustrating.

To address these problems and satisfy practical requirements about stateful network element upgrades, we present CLICK-UP [2], which is, to the best of our knowledge, the first research effort towards software upgrades of Click-based stateful network elements.

The reason behind drawback ***D1*** is that software integration of upgrades in Click does not stick well enough to the service context. Currently, a series of functionality-neutral modules are redundantly shipped with essential modules. We argue that explicitly integrating essential modules *in a service context-aware manner* can cut down upgrade overheads, and could be a solution for ***D1***. Second, the current Click modularity still poses significant challenges in maintaining network states during software upgrades. Instead of expecting the operators who originate network elements to manage any problem, we argue that a state synchronization scheme could be forcefully integrated into modules, and would address ***D2***. Finally, we argue that a lightweight runtime library of software upgrades is also necessary to clarify service semantics, simplifying the tedious orchestration and unifying interfaces. This would address ***D3***.

Besides handling software upgrades, CLICK-UP has also been designed with compatibility in mind, for example with ClickOS [5], CliMB [4] and FastClick [1].

2 DESIGN OF CLICK-UP

Design Overview: As shown in Figure 1, the framework of CLICK-UP is divided into the top three layers.

In contrast with native Click processing, in CLICK-UP, operators directly interact with the DAG orchestrator (top layer of Figure 1) and formulate their required atom-based pipeline (See Figure 2) with a view of the runtime library. During the new processing, all dependencies of essential modules are fully explored according to the determined service context coming from the DAG pipeline. This cuts down unnecessary overheads of module integration, and speeds up the recovery from downtime as much as possible. The exploration is implemented by maintaining the required module list for each module. A specified module can be integrated only after all its required modules have been integrated.

*Junxiao Wang, Yuchen Huang, Heng Qi, and Keqiu Li are with the School of Computer Science and Technology, Dalian University of Technology. Steve Uhlig is with the School of Electronic Engineering and Computer Science, Queen Mary University of London.

SIGCOMM Posters and Demos '18, Budapest, Hungary
2018. ACM ISBN 123-4567-24-567/08/06...$15.00
https://doi.org/10.475/123_4

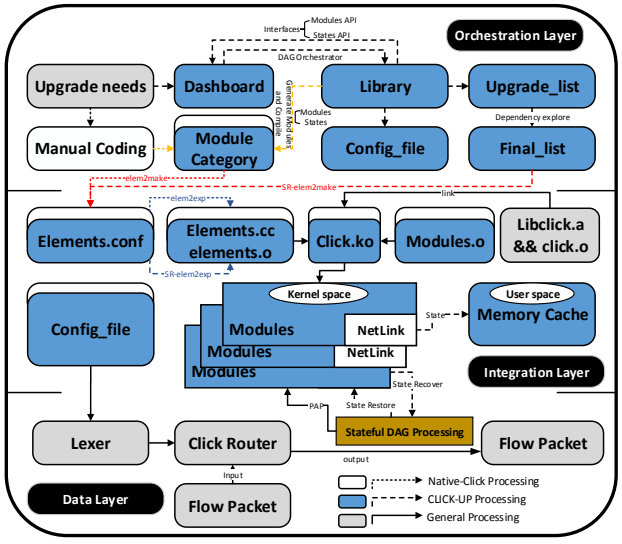

Figure 1: CLICK-UP framework overview.

The reconstruction scheme for lost states is implemented through periodic synchronization between modules and persistent storage, located in kernel and user space respectively. With recovery bootstrap integrated inside all modules, the states can be recaptured after upgrade. We claim this is a new contribution to the upgrade of stateful network elements.

Workflow: The procedure for CLICK-UP to handle a software upgrade are shown as directed arrows in Figure 1: (1) The dashboard exposes the DAG orchestrator to operators, allowing operators to define their upgrade needs as a DAG. The DAG is based on well-known semantics and consists of a series of pipeline processing related atom functionalities, including their required service states. (2) The DAG should be parsed to a set of Click modules (called elements), and its new state collection is integrated into the corresponding modules. All modules have a state synchronization mechanism and a state reconstruction bootstrap. (3) The modules are compiled, built into kernel space, and the persistent storage in user space is initialized with a new version number. At the same time, former version related states are sent back to the module. (4) The new configuration is created and the upgraded network element is rebooted with its former service states fulfilled by a recovery bootstrap.

Atom-based Orchestration: As shown in Figure 2, the DAG orchestration of CLICK-UP is based on a series of core functionalities called atom functionalities, e.g., packet parsing, payload modification, and the like, each of which is easy to follow. Operators can thus represent their upgrade needs as a DAG with stateful or stateless atoms linked. By exploring all the dependencies of essential modules, the runtime library will parse the atom-based graph into a module-based graph that has identical service contexts and pipeline functionalities. This provides a unified interface for operators. At the same time, it provides compatibility between CLICK-UP and other Click-driven techniques based on module-style graphs. Moreover, since the orchestration comes with a service context, a series of functionality-neutral modules won't be shipped

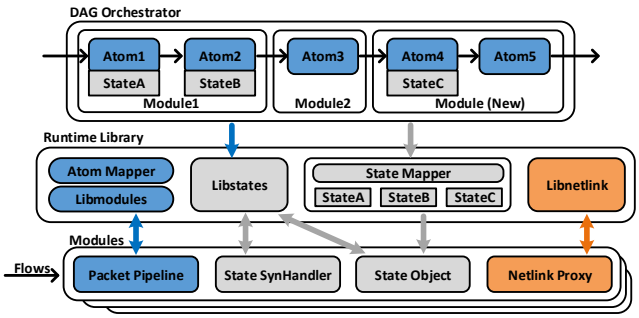

Figure 2: Atom-based upgrade orchestration.

with essential modules any more, significantly reducing the integration time of future modules.

Runtime Library: As shown in Figure 2, the runtime library of CLICK-UP consists of a set of interfaces exposed to the orchestration. They specify available atom functionalities as well as the mapping between atoms and module category. These interfaces also provide a state management toolset to deal with the state synchronization and the reconstruction bootstrap. Through top-down integration, all the modules of upgraded network elements can recover from downtime and take back their service states correctly. Developers can also enlarge the current library stack by implementing new interfaces for new atoms or states, when atom-based graphs do not match the existing upgrade needs.

3 DEMO

To further demonstrate the effectiveness and convenience of CLICK-UP[1], we offer a software upgrade walk-through of a firewall network element. We hope to demonstrate module integration by upgrading a certain number of new modules during this demo, and checking its downtime compared to the original version. We also plan to demonstrate the state persistency by implementing TCP session states in the demo, and observing the negative influence on state loss compared to our state reconstruction case.

During demonstration, we will provide more details compared to here, by using a laptop (with dashboard and HTTP benchmark on it) and a raspberry Pi (with a whitelist firewall and CLICK-UP server on it). On site, we will showcase at length how to apply CLICK-UP to the software upgrades of Click-driven stateful network elements.

REFERENCES

[1] Tom Barbette, Cyril Soldani, and Laurent Mathy. 2015. Fast Userspace Packet Processing. In *Proc. of ACM/IEEE ANCS*.
[2] CLICK-UP. 2018. Project Website. (2018). https://click-up.github.io.
[3] Eddie Kohler, Robert Morris, Benjie Chen, et al. 2000. The click modular router. *ACM Transactions on Computer Systems* 18, 3 (2000), 263–297.
[4] Rafael Laufer, Massimo Gallo, Diego Perino, et al. 2016. CliMB: Enabling Network Function Composition with Click Middleboxes. In *Proc. of ACM HotMiddlebox*.
[5] Joao Martins, Mohamed Ahmed, Costin Raiciu, et al. 2014. ClickOS and the art of network function virtualization. In *Proc. of USENIX NSDI*.

[1] Source codes are now available at https://github.com/click-up

Demo Requirement

- **Equipment to be used for the demo**:
  A raspberry Pi 3B,
  A wireless router,
  A laptop.
- **Space needed**:
  Table space by default.
- **Setup time required**:
  Half an hour.
- **Additional facilities needed, including power and any Internet access requirements**:
  Power by default,
  Wireless Internet access by default.