Scalable and Resilient Network Traffic Engineering Using Erlang-based Path Computation Element

Sébastien Merle*, Juan-Pedro Fernández-Palacios†,
Oscar González-de-Dios†, Lluís Gifre‡, Ricard Vilalta‡, Peer Stritzinger*

*Peer Stritzinger GmbH, Germany
†Telefónica I+D, Madrid, Spain
‡Centre Tecnològic de Telecomunicacions de Catalunya (CTTC/CERCA), Castelldefels, Spain

***Abstract*—Scalability and resiliency are two significant requirements for upcoming beyond 5G networks. As network resource usage is expected to increase, scalability of the control plane handling the required amount of network resources is also necessary. Moreover, resiliency of cloudified control plane software mechanisms is also a requirement. This demonstration presents an Erlang-based Path Computation Element (PCE) able to perform traffic engineering for Segment Routing. Specific scalability and resiliency properties are provided in order to support transport connectivity in beyond 5G networks.**

I. Introduction

Current cloudified beyond 5G networks will support cloud-scale number of connectivity requests, including high variability of network requirements to be supported on top of the same infrastructure. To this end, network control and management plane software is also proposed to be deployed using cloud infrastructure to cope with these stringent requirements.

As one of the objectives of the TeraFlow OS, a transport SDN controller [1], is scaling to support the large number of concurrent flows required in a beyond 5G world, using a language designed for telecommunications and offering native tools for scalability and resilience is a big advantage. Resilience mechanisms are also necessary in terms of allowing the deployed control and management plane services to survive to unknown states due to the inclusion of whiteboxes and network disaggregation.

Erlang is a general-purpose functional language initially developed by Ericsson to build high-performance telecom switch systems. These systems are expected to scale horizontally, handle a heavy load with very low latency, and run practically forever without service interruption [2]. These specific requirements guided the design of the language: a) First-class processes with strong isolation; b) Integrated support for transparent distribution; c) High-level distributed coordination mechanisms; d) Sophisticated fault tolerance abstractions; e) Automatic memory management; f) Soft real-time support; and g) Hot-loading software updates.

To summarise, Erlang and the Open Telecom Platform (OTP) [3] are the pinnacles of thirty years of correct design choices that directly apply to modern networking applications. It takes direct advantage of the improvements in hardware parallelism of the last decades. This makes Erlang an ideal choice for the TeraFlow OS, and therefore adding native support for Path Computation Element Protocol (PCEP) is a natural step toward supporting scalable Traffic Engineering.

The objective of this demonstration is to present the current state of our scalable and resilient Traffic Engineering PCE implementation in Erlang, and how it can already be used for steering traffic in a simulated network.

II. Demonstration and relevance

The Erlang PCE spawns two processes per PCEP connection. The first one is handling the TCP/IP connection and the encoding and decoding of the PCEP protocol into internal data structures. The decoded PCEP messages are sent to the second process that manages the protocol state machine and the cached context data. This second process presents a high-level API to the rest of the service that is mostly PCEP-independent. The PCE itself is formed by a pool of processes each ones owning a shard of the flow space. The PCE processes are aware of the session processes with flows defined in their managed space.

With the Erlang virtual machine, more than a million processes can be spawned on modern hardware, the main limitation being the memory and the IO, offering good vertical scaling. In addition, Erlang provides transparent distribution and inter-process communication, making it possible for all these processes to run on different physical machines, and therefore scaling the system horizontally.

Erlang is a functional language where all the processes are isolated and communicate between each other with messages, there is no shared memory. This makes Erlang applications reliable and fault-tolerant, as no state corruption is ever propagated to the rest of the system. In addition, the native tools for process monitoring and supervision allow the system to heal itself automatically by restarting any failing process from a known working state.

For this demonstration, we will simulate a network composed of 6 interconnected routers, where both head-ends routers are connected to a PCE through PCEP. The routers are instances of Free Range Routing (FRR) daemons running on a single desktop machine and the network is virtualized using Linux namespaces (see Fig. 1). The Erlang PCE service is responsible for creating packet flows between the head-ends routers of the network, and modify them dynamically for Traffic Engineering purposes.

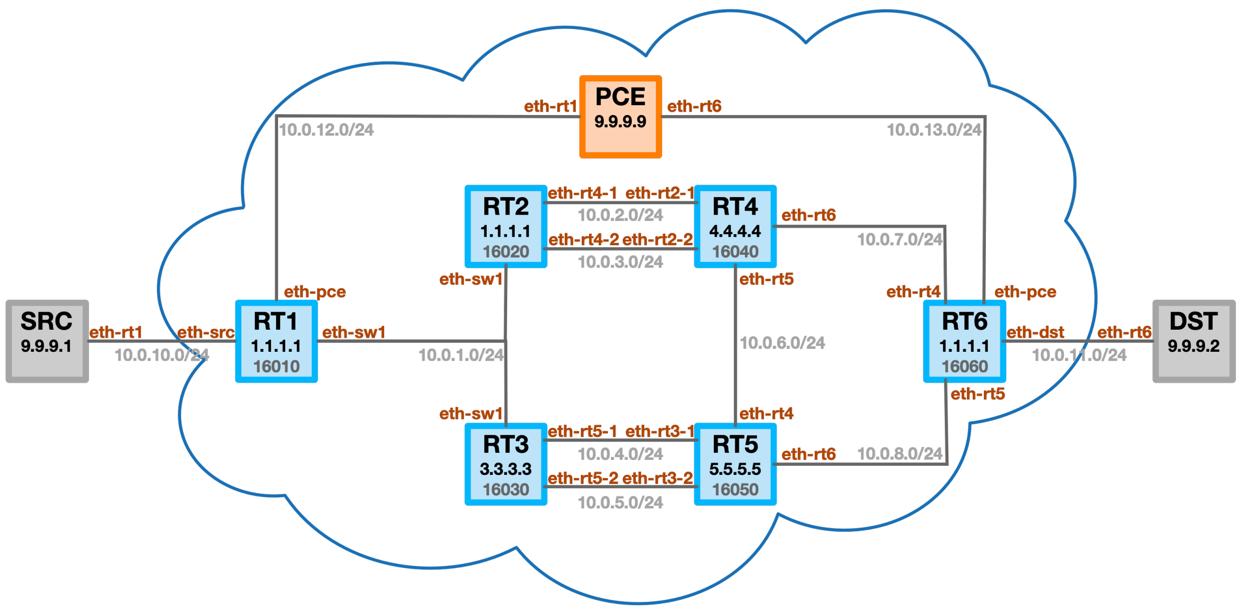


Fig. 1: Demonstration topology

By being a native Erlang application, the PCE can take advantage of the concurrency and resiliency offered by the language and can use native process communication with the rest of the TeraFlow OS services implemented in Erlang. The result is a scalable PCE with really low latency that might integrate perfectly in an micro-service based SDN controller.

The PCE node is selected and the Erlang PCE is started in console mode. Both headend routers can be seen connecting to the PCE and receiving the default route for their requested path. From the source node, the destination node is reachable, meaning that a bidirectional packet flow has been established between them. Using tcpdump on router 4's second interface, both the ICMP request and reply packets can be seen, confirming the path of the flow in the network has been established.

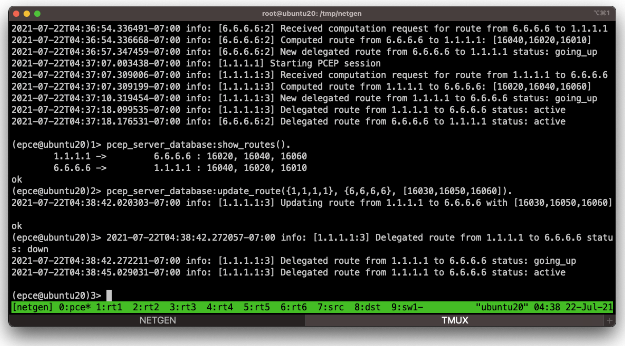
Going back on the PCE node, the Erlang console is used to manually update the flow from router 1 to router 6 to pass through router 5 (see Fig. 2). Inspecting the packets on router 5 shows the ICMP response has been steered through the new path. For completeness, the Erlang console on the PCE node is used to update the second flow from router 6 to router 1 to pass through router 5 too (see Fig. 3). It can finally be seen on router 5 that both ICMP requests and replies are following the new established path.

III. Conclusion

This demonstration has presented a native Erlang application performing Traffic engineering in a virtualized network, allowing the TeraFlow OS to take full advantage of the scalability and resiliency offered by the Erlang programming language and the OTP framework.

Acknowledgment

Work partially supported by the EC H2020 TeraFlow (101015857) and Spanish AURORAS (RTI2018-099178-I00).



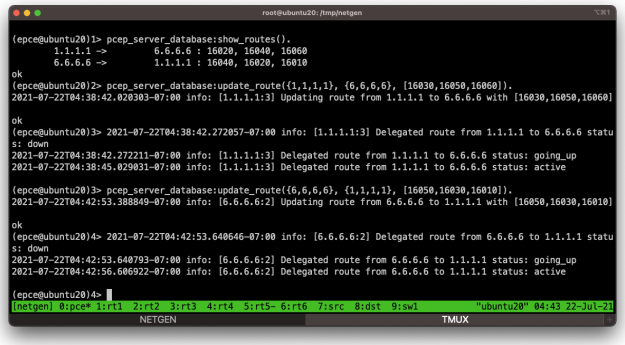
```
2021-07-22T04:36:54.336491-07:00 info: [6.6.6.6:2] Received computation request for route from 6.6.6.6 to 1.1.1.1
2021-07-22T04:36:54.336668-07:00 info: [6.6.6.6:2] Computed route from 6.6.6.6 to 1.1.1.1: [16040,16020,16010]
2021-07-22T04:36:57.347459-07:00 info: [6.6.6.6:2] New delegated route from 6.6.6.6 to 1.1.1.1 status: going_up
2021-07-22T04:37:07.003438-07:00 info: [1.1.1.1] Starting PCEP session
2021-07-22T04:37:07.309006-07:00 info: [1.1.1.1:3] Received computation request for route from 1.1.1.1 to 6.6.6.6
2021-07-22T04:37:07.309199-07:00 info: [1.1.1.1:3] Computed route from 1.1.1.1 to 6.6.6.6: [16020,16040,16060]
2021-07-22T04:37:10.319454-07:00 info: [1.1.1.1:3] New delegated route from 1.1.1.1 to 6.6.6.6 status: going_up
2021-07-22T04:37:18.099535-07:00 info: [1.1.1.1:3] Delegated route from 1.1.1.1 to 6.6.6.6 status: active
2021-07-22T04:37:18.176531-07:00 info: [6.6.6.6:2] Delegated route from 6.6.6.6 to 1.1.1.1 status: active

(epce@ubuntu20)1> pcep_server_database:show_routes().
        1.1.1.1 ->         6.6.6.6 : 16020, 16040, 16060
        6.6.6.6 ->         1.1.1.1 : 16040, 16020, 16010
ok
(epce@ubuntu20)2> pcep_server_database:update_route({1,1,1,1}, {6,6,6,6}, [16030,16050,16060]).
2021-07-22T04:38:42.020303-07:00 info: [1.1.1.1:3] Updating route from 1.1.1.1 to 6.6.6.6 with [16030,16050,16060]

ok
(epce@ubuntu20)3> 2021-07-22T04:38:42.272057-07:00 info: [1.1.1.1:3] Delegated route from 1.1.1.1 to 6.6.6.6 statu
s: down
2021-07-22T04:38:42.272211-07:00 info: [1.1.1.1:3] Delegated route from 1.1.1.1 to 6.6.6.6 status: going_up
2021-07-22T04:38:45.029031-07:00 info: [1.1.1.1:3] Delegated route from 1.1.1.1 to 6.6.6.6 status: active

(epce@ubuntu20)3> 
[netgen] 0:pce* 1:rt1  2:rt2  3:rt3  4:rt4  5:rt5  6:rt6  7:src  8:dst  9:sw1-        "ubuntu20" 04:38 22-Jul-21
```

Fig. 2: Update packet flow from router 1 to router 6



```
(epce@ubuntu20)1> pcep_server_database:show_routes().
        1.1.1.1 ->         6.6.6.6 : 16020, 16040, 16060
        6.6.6.6 ->         1.1.1.1 : 16040, 16020, 16010
ok
(epce@ubuntu20)2> pcep_server_database:update_route({1,1,1,1}, {6,6,6,6}, [16030,16050,16060]).
2021-07-22T04:38:42.020303-07:00 info: [1.1.1.1:3] Updating route from 1.1.1.1 to 6.6.6.6 with [16030,16050,16060]

ok
(epce@ubuntu20)3> 2021-07-22T04:38:42.272057-07:00 info: [1.1.1.1:3] Delegated route from 1.1.1.1 to 6.6.6.6 statu
s: down
2021-07-22T04:38:42.272211-07:00 info: [1.1.1.1:3] Delegated route from 1.1.1.1 to 6.6.6.6 status: going_up
2021-07-22T04:38:45.029031-07:00 info: [1.1.1.1:3] Delegated route from 1.1.1.1 to 6.6.6.6 status: active

(epce@ubuntu20)3> pcep_server_database:update_route({6,6,6,6}, {1,1,1,1}, [16050,16030,16010]).
2021-07-22T04:42:53.388849-07:00 info: [6.6.6.6:2] Updating route from 6.6.6.6 to 1.1.1.1 with [16050,16030,16010]

ok
(epce@ubuntu20)4> 2021-07-22T04:42:53.640646-07:00 info: [6.6.6.6:2] Delegated route from 6.6.6.6 to 1.1.1.1 statu
s: down
2021-07-22T04:42:53.640793-07:00 info: [6.6.6.6:2] Delegated route from 6.6.6.6 to 1.1.1.1 status: going_up
2021-07-22T04:42:56.606922-07:00 info: [6.6.6.6:2] Delegated route from 6.6.6.6 to 1.1.1.1 status: active

(epce@ubuntu20)4> 
[netgen] 0:pce* 1:rt1  2:rt2  3:rt3  4:rt4  5:rt5- 6:rt6  7:src  8:dst  9:sw1         "ubuntu20" 04:43 22-Jul-21
```

Fig. 3: Update packet flow from router 6 to router 1

References

[1] R. Vilalta *et al.*, “Teraflow: Secured autonomic traffic management for a tera of sdn flows,” in *EUCNC*, 2021.

[2] A. Lindberg, S. Merle, and P. Stritzinger, “Scaling erlang distribution: going beyond the fully connected mesh,” in *Proceedings of the 18th ACM SIGPLAN International Workshop on Erlang*, 2019, pp. 48–55.

[3] “Erlang and otp,” accessed: 2021-07-22. [Online]. Available: http://erlang.org/doc/system_architecture_intro/sys_arch_intro.html