Evaluation of Network Function Virtualization Frameworks on Telco Workloads

András Gulyás, Gábor Rétvári (BME)

Contents

1 Abstract

Network function virtualization is a key to move traditional telco workloads from costly on-site black-box devices to centralized, easy-to-manage cloud-like data centers. In order to evaluate how a virtualized network function performs in such a system, we started the implementation of a professional benchmarking system called TIPSY and we implemented and evaluated different telco workloads using TIPSY.

In this document first we report on the initial progress we achieved with TIPSY, then we describe an exemplary telco pipeline, the Mobile Gateway Pipeline, we implemented on top of TIPSY, and finally we briefly overview some results from an extensive evaluation of this pipeline over the Berkeley Extensible Software Switch (BESS) and we present the conclusions.

2 The Telco pIPeline benchmarking SYstem (TIPSY)

TIPSY is a benchmark suite to evaluate and compare the performance of programmable data plane technologies and network-function virtualization frameworks over a set of standard scenarios rooted in telecommunications practice. Apart from simple L2 and L3 pipelines, currently there is a rather complex BNG (Broadband Network Gateway) and 5G MGW (Mobile Gateway) pipeline defined and implemented in TIPSY, with further pipelines and implementations to follow soon.

The aim of TIPSY is to provide the networking community a set of **standardized telco-oriented scenarios** on top of which different **programmable data plane technologies can be fairly and**

comprehensibly evaluated. The target audience is network operators who want to test new data-plane equipment, network engineers evaluating the scalability of a programmable switch in terms of increasingly complex configurations, or researchers who want to compare a new data plane algorithm with existing and established technology.

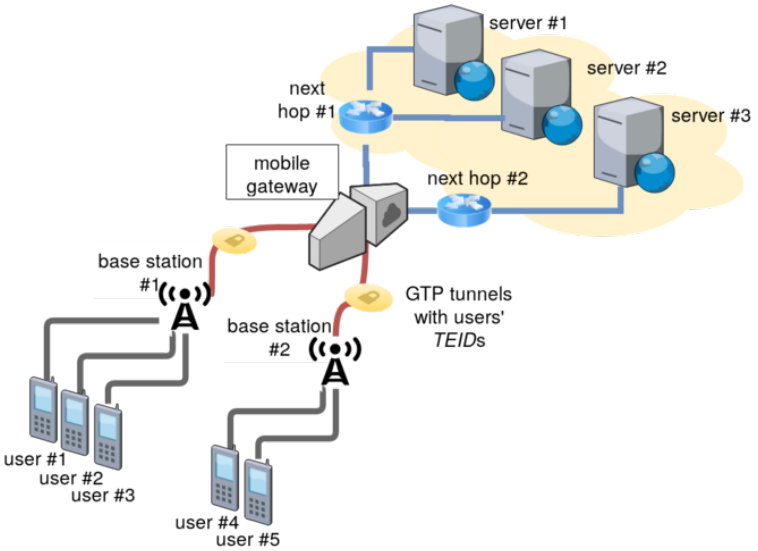
TIPSY comprises 6 elemental parts, currently existing at varying levels of maturity:

- a set of telco pipelines with working implementations (currently OpenFlow and BESS are supported, contributions are welcome),
- a test suite to validate the implementations (ongoing),
- a configuration system that allows to fine-tune general parameters of the pipelines (supported),
- a tunable trace generator to produce deterministic traffic traces for repeatable experiments and reproducible results (supported),
- a distributed measurement infrastructure to feed the traffic traces to the system-under-test, controller code to drive dynamic benchmarks, and an evaluation framework that visualizes the results (rudimentary),
- an evaluation and visualization framework to generate production-quality reports from the benchmark results (planned).

3 The Mobile Gateway Pipeline

The mobile gateway pipeline (name: `mgw`) represents a simplified 5G gateway that connects a set of mobile user equipments (UEs), located behind base stations (BSTs), to a set of public servers available on the Internet.

The general setup for this pipeline is given below.



3.1 Static pipeline

In the uplink direction (UE/BST -> server) the MGW receives GTP-encapsulated packets from the base stations, identified by the source IP address in the GTP header, and forwards the decapsulated packets to public servers in the Internet. After decapsulation, the source IP address identifies the user and the GTP TEID identifies the bearer, the the destination IP designates the public server to forward the packet to. The uplink pipeline performs various checks, decapsulates the UE's packet from the GTP tunnel, identifies the user and polices the UE (rate limiting), and then routes the decapculated packet to the Internet.

In the downlink direction (server -> user/bst) the MGW receives normal packets from the Internet and the pipeline is basically the reverse of the uplink one: identify the UE/bearer based on the packet destination IP address (we now assume there is only one bearer per user), rate limit the UE flow, and encapsulate and send the packet to the BST behind which the UE is currently located.

In particular, the MGW performs the following processing steps per uplink/downlink packet:

Uplink:

- L2, L3 and L4 check (gateway MAC/IP and UDP port destination 2152)
- GTP decap, save TEID
- rate limit per bearer (TEID)
- L3 routing towards the Internet + L2 fwd

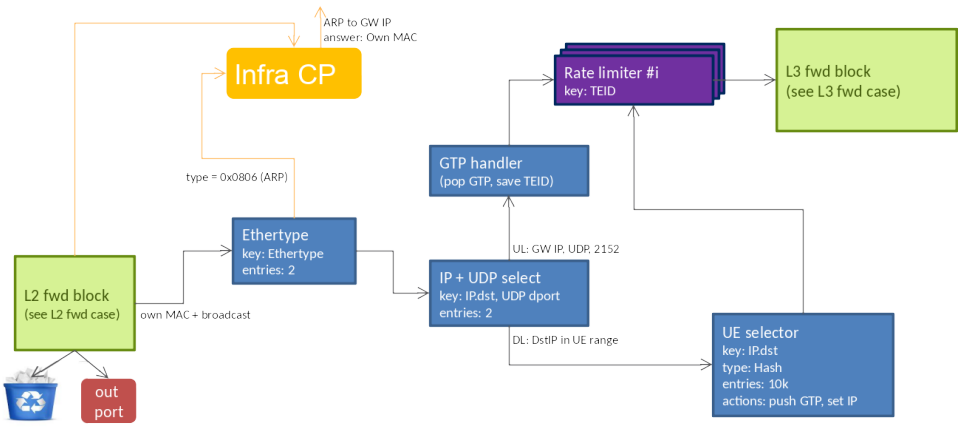
Downlink:

- L2 and L3 check (check if destination IP is in the UE range)
- per user rate limiting
- GTP encap (set bearer in TEID)
- set destination IP of the base station of the UE
- L3 routing towards BSTs + L2 fwd

This roughly maps to the below modules:

- Ingress:
  - `l2_fwd`: identify local packets
  - `ether_type`: identify ARP/IP packets
  - `dir_selector`: distinguish uplink/downlink packets
- Egress:
  - `rate_limiter`: per user traffic policing
  - `l3_lookup`: route to servers (uplink) and BSTs (downlink)
  - `group_table`: next-hop setting per server and BST
- Uplink: Ingress -> GTP decap -> Egress
- Downlink: Ingress -> UE selector -> GTP encap -> Egress
  - `ue_selector`: find the GTP TEID for the user's IP address

The next figure details the MGW pipeline.



3.2 Dynamic scenarios

TIPSY defines the below update scenarios for the MGW pipeline.

- `user-update`: model the arrival/departure of a user; a UE arrives/departs to/from a BST, involving the following updates to the pipeline:
  - an entry is added/removed to/from the `ue_selector` module
  - the queue for the user in the `rate_limiter` is updated
- `handover`: models user mobility (handover); a user's attachment point (i.e., BST) changes, with the below changes to the pipeline:
  - the `ue_selector` table is updated
- `server-update`: addition/removal of a server: a destination in the public Internet changes
  - the `l3_lookup` and the `group_table` are modified accordingly

3.3 Pipeline configuration

A sample TIPSY configuration for the MGW pipeline is shown below:

```
{
    "pipeline": {
         "name": "mgw",
         "user": [1, 2, 3],
         "bst": [9, 4, 10],
         "server": 1,
         "rate-limit": 10000,
         "nhop": 4,
         "fakedrop": false,
         "fluct-user": [17, 200],
         "handover": 0,
         "fluct-server": 4,
    }
}
```

The parameters specific to the MGW pipeline are as follows:

- `name`: name of the pipeline, must be set to `mgw` for the MGW pipeline
- `user`: number of UEs
- `bst`: number of BSTs
- `server`: number of public servers
- `rate-limit`: rate limit threshold [byte/sec]
- `nhop`: number of next-hops in the L3 table towards the public Internet
- `fakedrop`: whether to actually drop unmatched packets (`false`) or send them immediately to the output port (`true`) for correct rate measurements
- `fluct-user`: number of user arrival/departure events (`user-update`) per sec
- `handover`: number of handover events (`handover`) per sec
- `fluct-server`: number of server update events (`server-update`) per sec
- `fakedrop`: whether to actually drop unmatched packets (`false`) or send them immediately to the output port (`false`) for correct rate measurements

4 Evaluations on the Berkeley Extensible Software Switch (BESS)

4.1 BESS Implementation

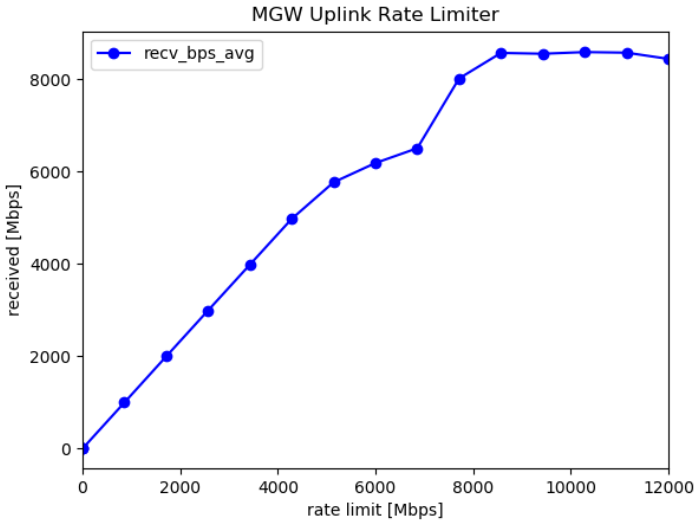
BESS is a programmable software switch implemented on top of the Intel DPDK. BESS currently contains a number of limitations we had to take into account when implementing the Mobile Gateway pipeline.

- No GTP in BESS: substituted with VxLAN

- VXLAN encap bug in BESS: multiple VxLAN encaps do not work, no error handling when headroom is too small (reported but upstream has no fix yet)
- Rate limiters in BESS: no rate limiter per se, emulated with a Queue + Scheduler
- IPLookup module bug: did not handle partial batches, worked around by adding a Buffer in front (reported & fixed)
- Splitter module bug: metadata-based split had an endianness bug (reported + fixed upstream)
- IPLookup module: memory leakage (reported + fixed)
- BESS service graph scalability: BESS reconstructs the service graph from scratch upon any updates, like module or link insertions, rewrote BESS core to incremental updates (reported, proposed fix, fixed upstream)
- Traffic management: BESS WFQ scheduler enters a deadlock when certain queues are drained (reported, no fix yet, problem seems deeper), worked around by round-robin or using the experimental scheduler

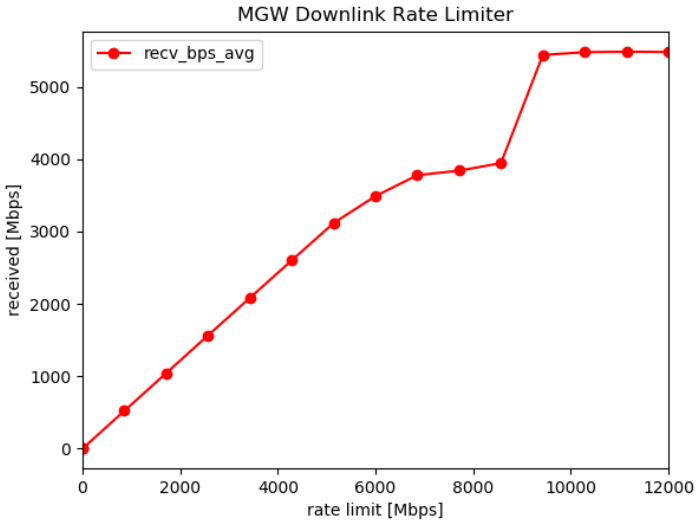
4.2 Results

4.2.1 Rate limiter: Upstream



- Rate limiter hack seems to work in upstream dir
- We hit BESS performance peak at ~8.5Gbps

4.2.2 Rate limiter: Downstream



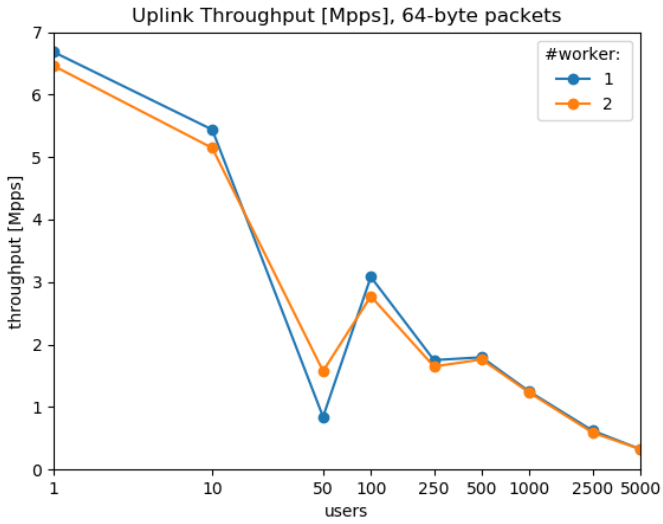
- Rate limiter not so precise in downlink direction
- BESS would need a built-in rate limiter module

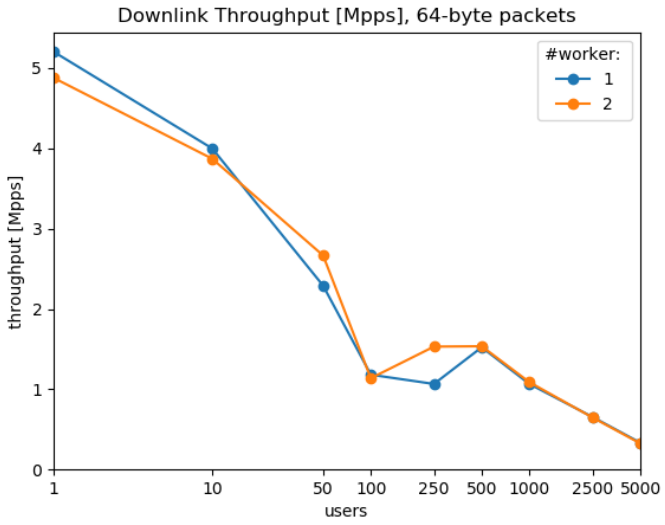
4.2.3 Scalability

- Basic config: #users=50, #servers=5, #bsts=2
- Base scalability: take base and proportionally increase each of #users, #servers, #bsts
- Number of CPU cores/workers: 1, 2, 4
- Measured: throughput [Mbps] and packet rate [Mpps], memory usage (resident set size, [Gbyte]), and init time [sec]

1. Throughput

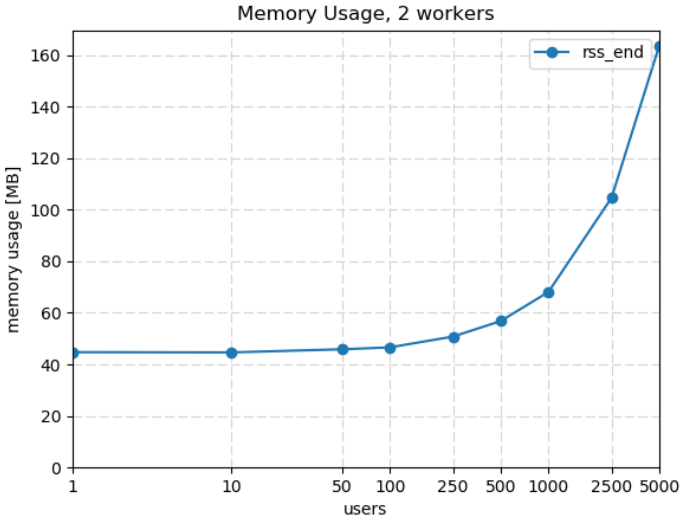
    - #users, #servers, #bsts grow simultaneously





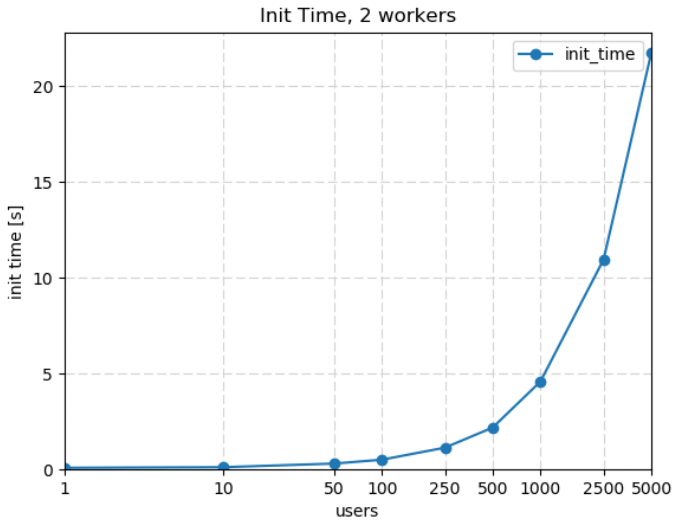
2. Memory

   - #users, #servers, #bsts grow simultaneously
   - 2 worker threads



3. Init time

   - #users, #servers, #bsts grow simultaneously
   - 2 worker threads



5 Conclusions

During the measurement studies, we made the following observations:

- BESS scales poorly when #users>1000
- Most probably due to the rate limiter hack (should write a rate limiter module!)
- Apart, it scales nicely with #srvs and #bsts
- For a modest number of users, we get packet rate in the millions per second range at reasonable memory usage
- In production, we should not admit more than 500 users into a single BESS datapath: virtual mobile gateway would allow us to scale the number of BESS datapaths freely
- Code seems stable (we needed to report/fix some bugs), no random segfaults seen