A low-level dive into building a high-speed NFV dataplane for service chaining

Tom Barbette[†] [*], Cyril Soldani[*], Romain Gaillard and Laurent Mathy
University of Liege
firstname.lastname@uliege.be

To cope with the growing performance needs of security appliances in datacenters or the network edge, current middlebox functionalities such as firewalls, NAT, DPI, content-aware optimizer or load-balancer are self-contained software. They avoid OS services as those are not tailored for NFV and use most of the time RAW sockets, or specific I/O frameworks (DPDK, Netmap, ...) to receive raw packets.

Therefore VNFs implement their needed functionalities by themselve. In consequence, we observed that usual middlebox service chains are subject to the following issues:

1. Packets are partially or completely re-classified in each middlebox component, *i.e.* packet headers are inspected to **classify** the packets according to known values such as “destination port 80” to decide that packets are HTTP packets.
2. A dictionary data structure is present in all stateful middleboxes to assign a memory-space for each group of packets belonging to the same **session**. The most widely used concept of session is the TCP 4-tuple, shared by all packets of the same TCP session.
3. They operate on different networking layers and therefore ship with their **own specialized stack**. Across the chain, most of those stacks actually execute redundant tasks.

In this work we present a system specifically designed to run a pipeline of VNFs. VNFs are developed as independent applications, but when piped together, the system combines the classification and sessions needs of the VNFs. The design is based on a common subsystem that implements shared features and makes sure no features are applied that aren’t absolutely needed by one of the VNFs (*e.g.* no or partial TCP reconstruction according to the VNFs needs). This brings together the advantage of re-using software components with the performance provided by state-of-the-art high-speed NFV frameworks that force reimplementing protocol specifics in each application. Even if the VNFs ship with their own full or specialized stacks, they may benefit from the unified session space provided by the flow manager table, avoiding the need for the identical classification and session mappings inside each of the VNFs.

While some ideas presented here are already known, we offer a more practical, low-level dive into building a high-speed NFV dataplane. We show unique considerations about factorizing session management and multi-protocol support for high-speed in-the-middle inspection and modification of flows. The system also offers automatic, session-aware parallelism to handle a large amount of flows.

The framework also provides a zero-copy **stream abstraction**, allowing to modify packets of the same session without the need for any knowledge of the underlying protocols. The abstraction provides seamless inspection and modification of the content of any flows (such as TCP or HTTP), automatically reflecting a consistent view, across layers, of flows modified on-the-fly. When an HTTP payload is modified, the content-length must be corrected. A layered approach allows to back-propagate the effect of stream modification across lower layers. Following this approach, we provide a TCP in-the-middle stack avoiding the overhead of a full TCP stack and greatly simplifies high-level middleboxes development. The stack can modify on-the-fly sequence and acknowledgement numbers on both sides of the stream when the upper layer makes changes.

The system finally provides support for a mechanism to “wait for more data” when a middlebox needs to buffer packets, unable to make a decision while data is still missing. Our TCP in-the-middle implementation supports pro-active ACKing to avoid stalling a flow while waiting for more data, and enables handling of large amounts of flow using a run to completion-or-buffer model.

Our prototype, codenamed MiddleClick, has been implemented on top of FastClick, an enhanced version of the Click Modular Router. This gives rise to a user-space software NFV dataplane enabling easy implementation of middlebox functionalities, as well as the deployment of complex scenarios.

[†] Presenter

[*] PhD Student

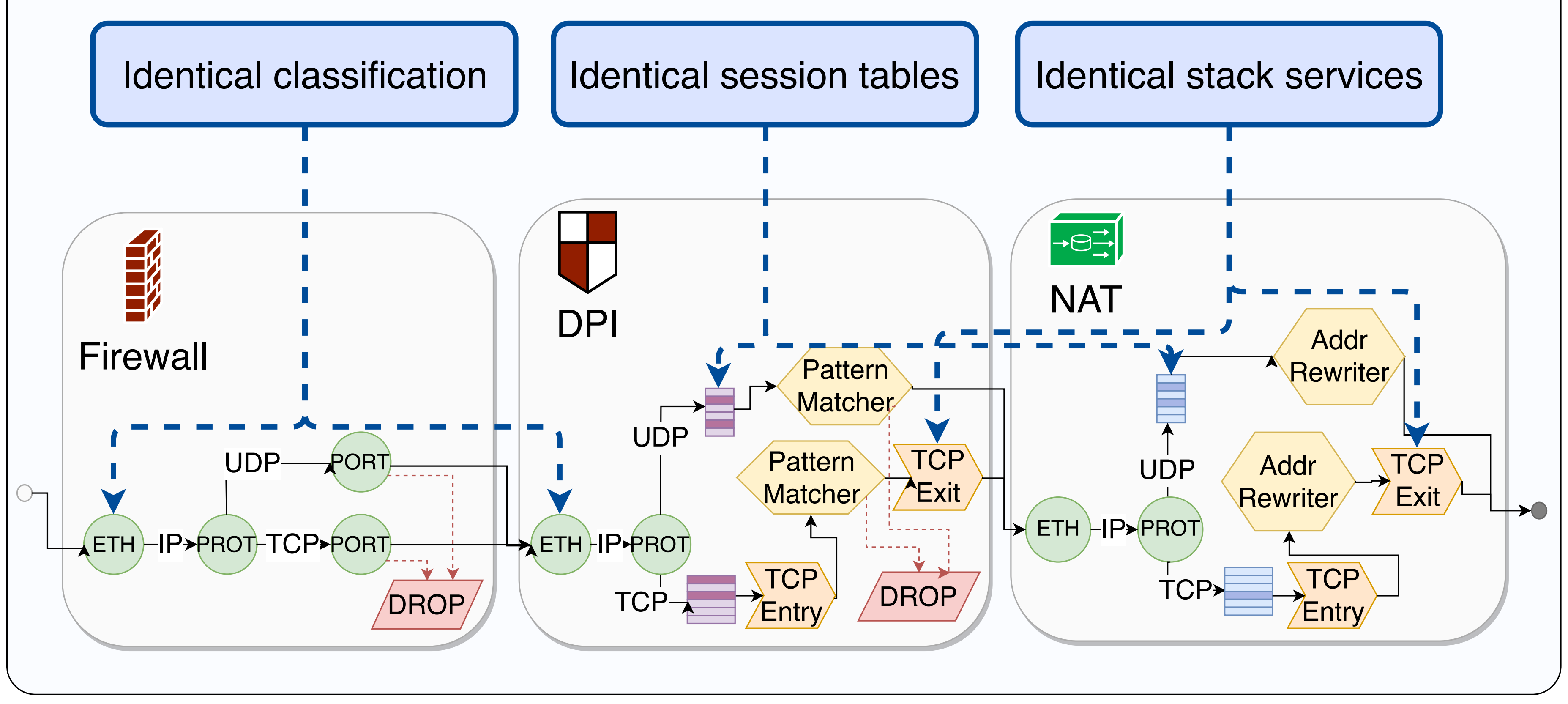
A low-level dive into building a high-speed NFV dataplane for service chaining

Tom Barbette, Cyril Soldani, Romain Gaillard and Laurent Mathy
University of Liege

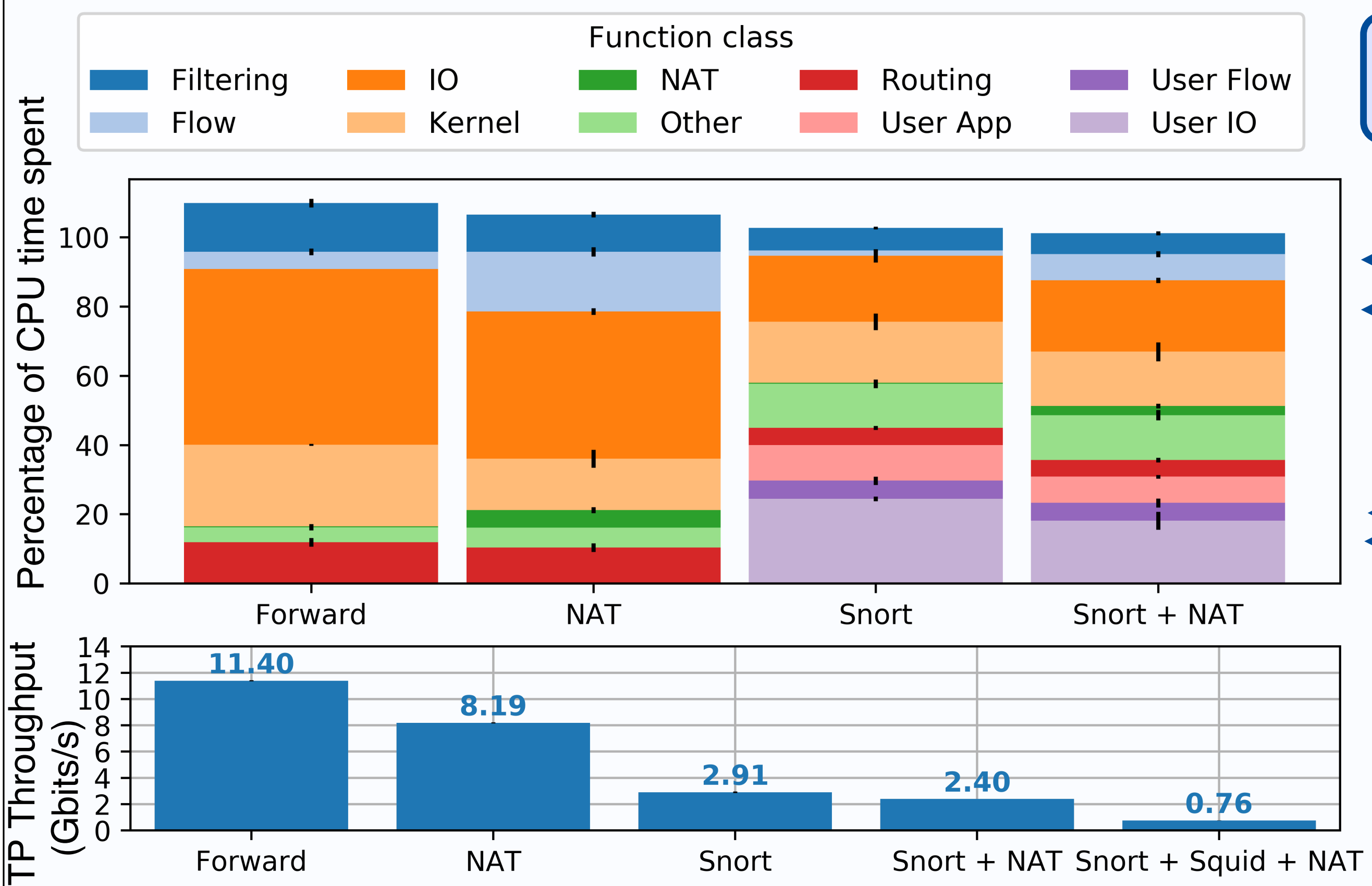
SUPERFLUIDITY EU H2020 PROJECT

The problem

Example service chain, in which middleboxes perform redundant classification and session reconstruction operations



Observation of its consequences



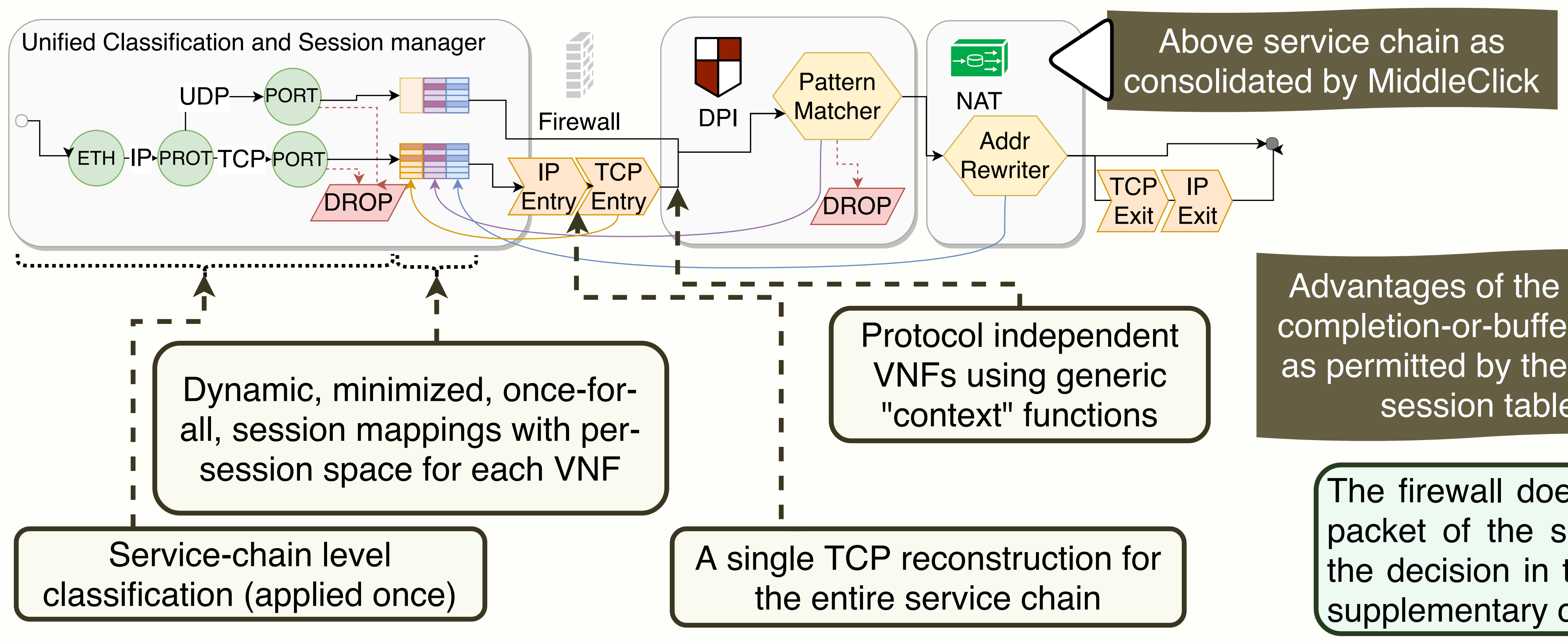
Profiling of multiple service chains running on a unique CPU core acting on 8K HTTP requests

2 identical session mappings

Time spent on I/O and classification

4 session mappings (+1 for TCP in-kernel + 1 for buckets in Squid)

MiddleClick architecture



Above service chain as consolidated by MiddleClick

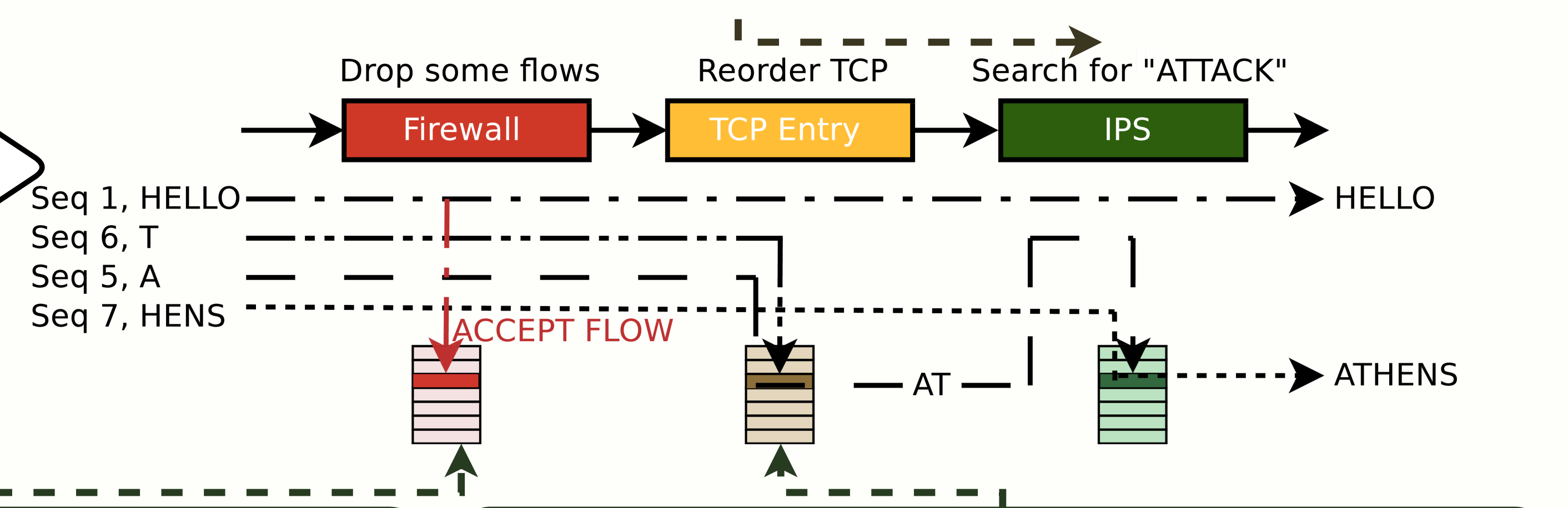
Dynamic, minimized, once-for-all, session mappings with per-session space for each VNF

Protocol independent VNFs using generic "context" functions

Service-chain level classification (applied once)

A single TCP reconstruction for the entire service chain

IPS can remember its DFA state while waiting for more data to make a decision. MiddleClick can send pro-active ACKs

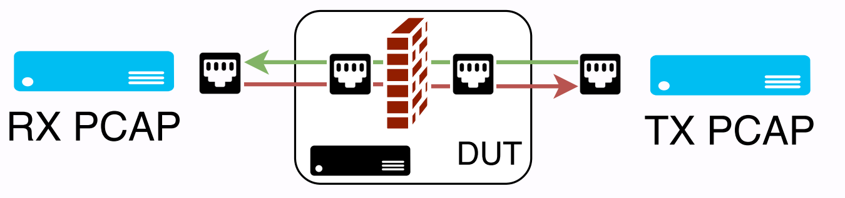


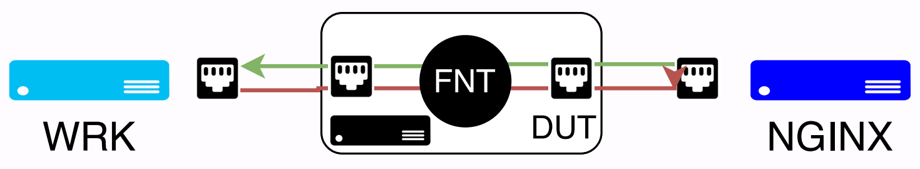
Advantages of the run-to-completion-or-buffer model as permitted by the unified session table

The firewall does a lookup only for the first packet of the session as it can remember the decision in the per-session space at no supplementary cost

The TCP reorderer has its space ready to put reference to out-of-order packets while waiting for the missing ones. No need for a flow table

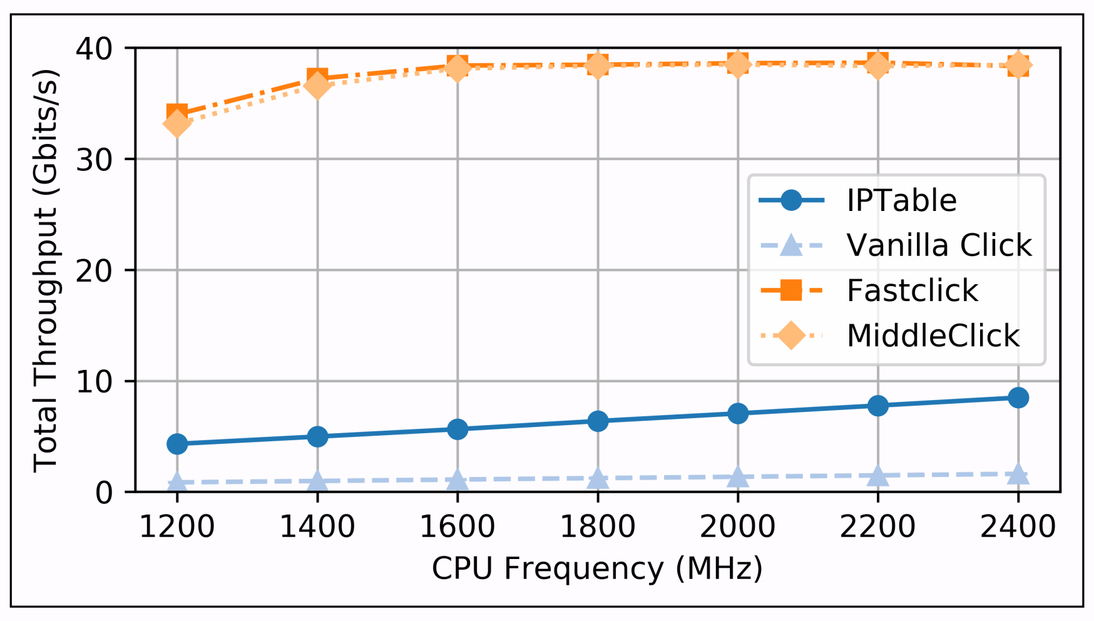
Evaluation



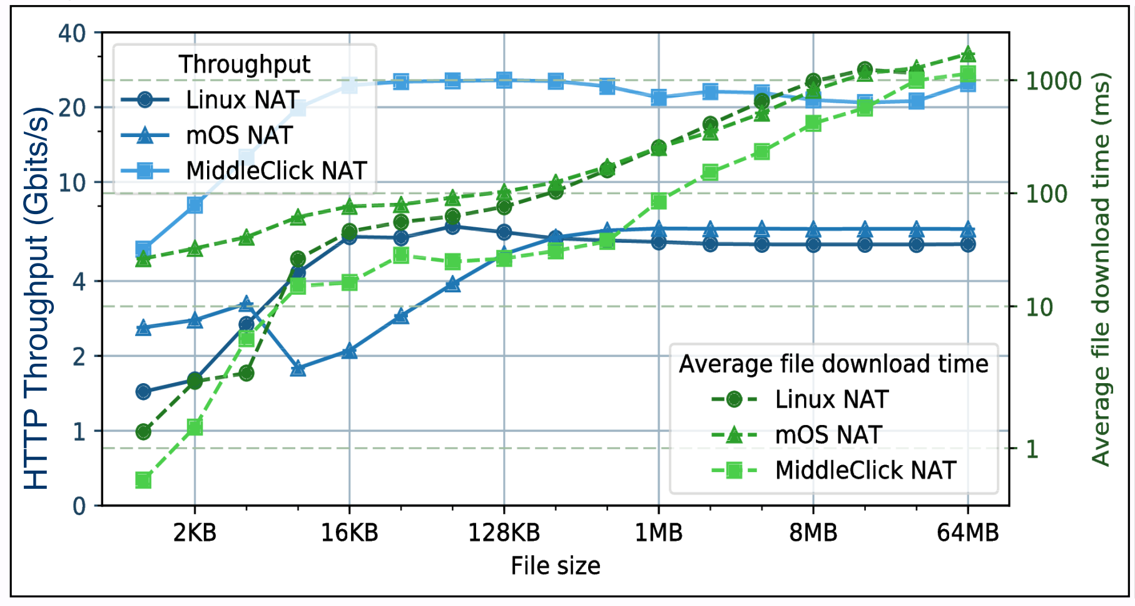


▬	Machine A (Xeon E5-2630 v3 8*2.4GHz)
▬	DUT (Xeon E5-2630 v3 8*2.4GHz)
▬	Machine C (Xeon E5-2683 v4 16* 2.1GHz)

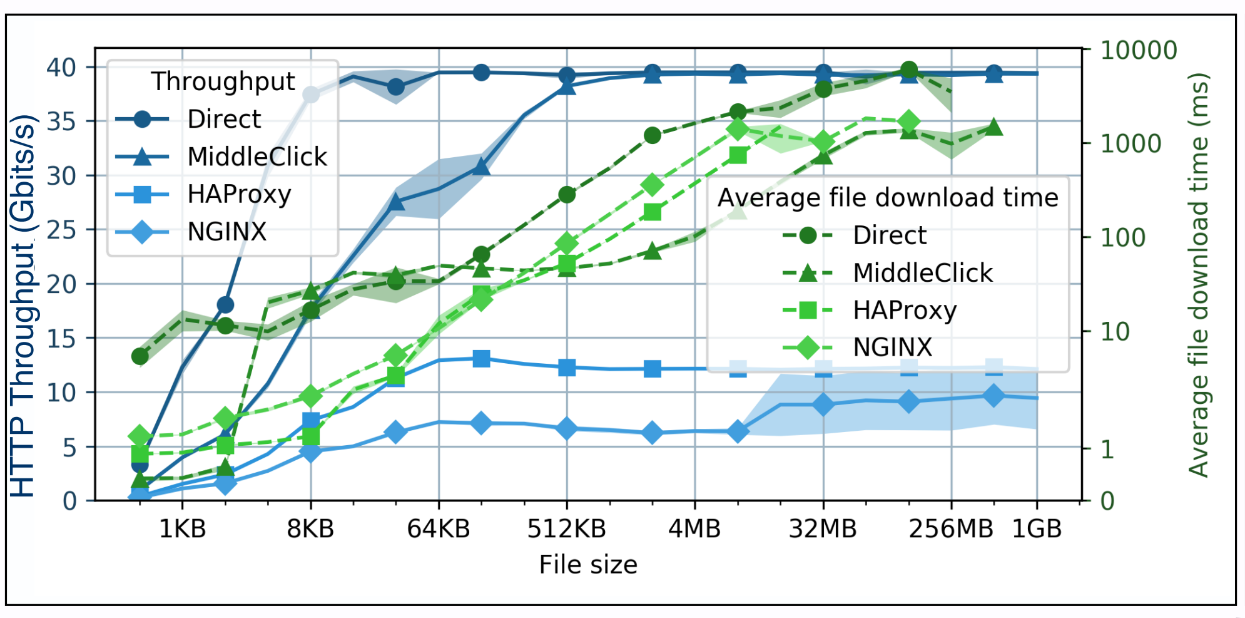
One 40G port of an Intel XL710 QDA2 NIC
DUT has two different cards



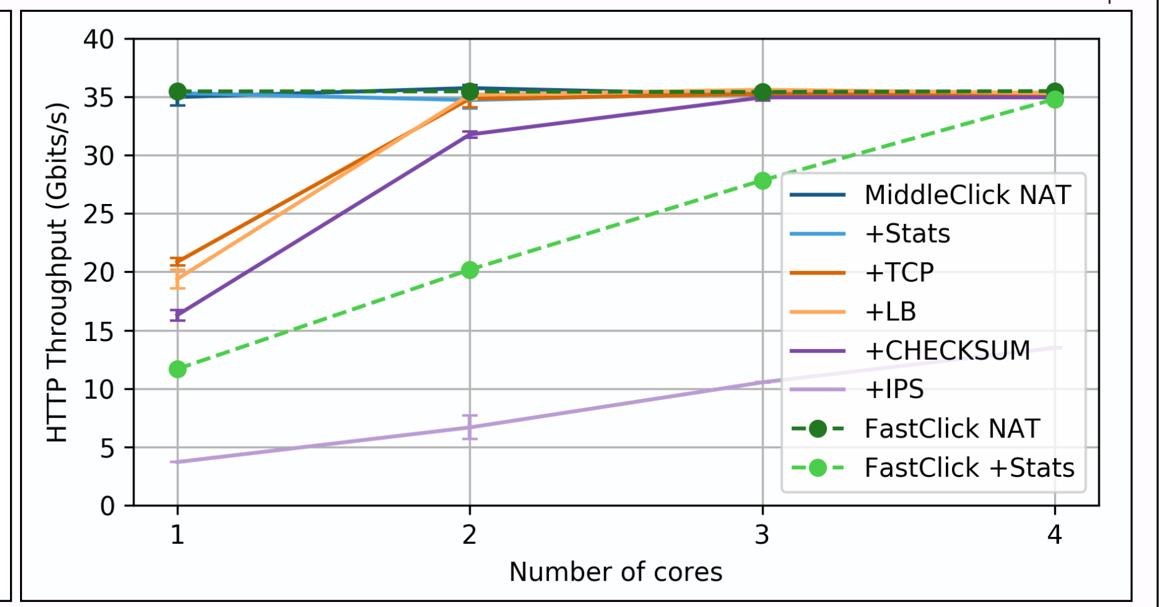
Total throughput for both sides of campus traces replayed through a **firewall** running on 2 CPU cores
Impact of the flow system is minimal



Throughput and latency of downloading files of varying size through a **NAT** running on 1 CPU core
MiddleClick has better throughput and latency

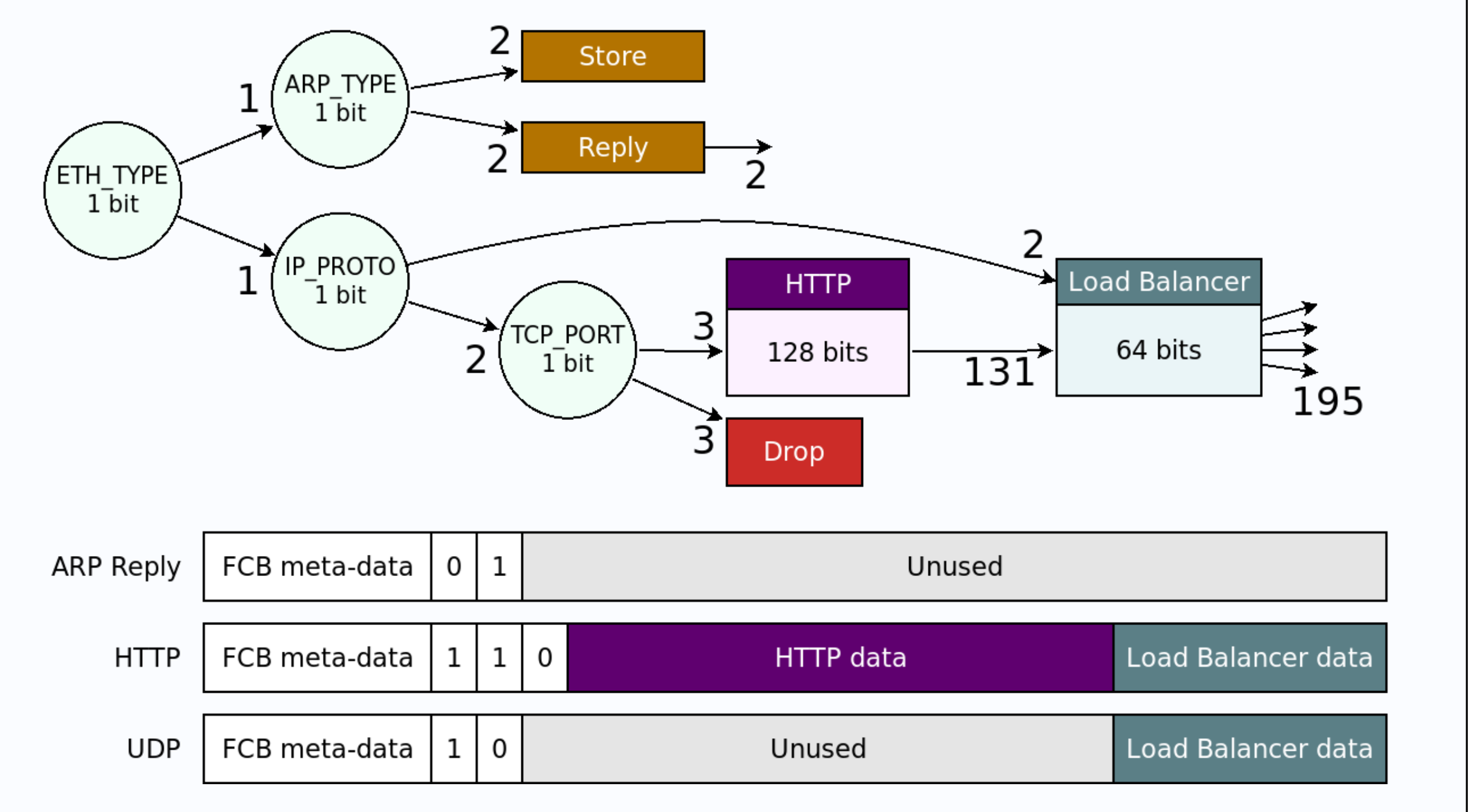


TCP load balancing reverse proxy running on 1 CPU core
MiddleClick achieves 1M requests/s of 1KB files



VNFs acting on 8K requests given an increasing number of cores
MiddleClick exhibits minimal performance impact with increasing number of flow-based middleboxes

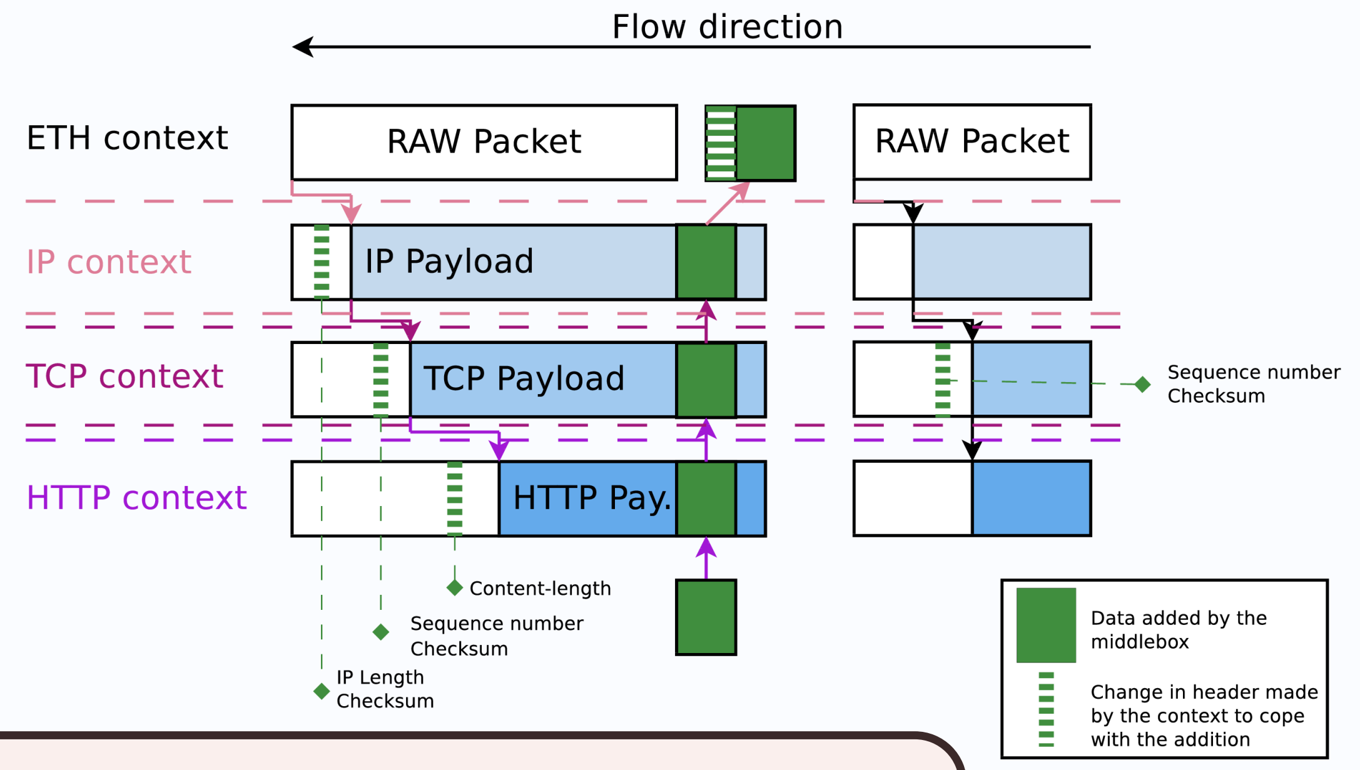
Unified sessions



MiddleClick maintains only the necessary number of sessions by associating each flow with a flow control block (FCB), instanciated per session. The amount of needed session space is collected from all the VNFs at start-up.

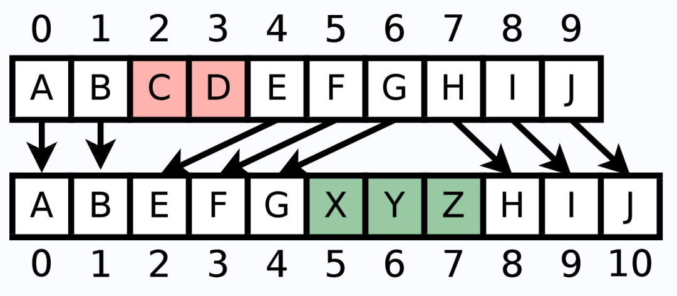
Flow abstraction and tempering

Upon entry in a protocol context, the payload offset is moved forward according to the protocol. When the stream is modified, lower layers of context take care of the implications.



Each VNF can make requests to its context, such as determine if a packet is the last useful one of the flow, or close the connection. They can also add and remove bytes.

In an HTTP flow, the Content-Length may be changed, and then the request passed to the lower layer, likely TCP. *I.e.* the TCP context will change SEQ numbers and ACK numbers of the other side of the connection. Then, the request will pass to the lower layer, etc...



List of modifications
2 → -2
6 → +3

A glimpse into state of the art solutions

I/O Frameworks

Other works focus on providing a fast I/O platform with different layers of isolation that could replace FastClick[3]. NetBricks (Compiler-based isolation)[1], Mirage (unikernel), ClickOS (Xen-based), NetVM (KVM), OpenNetVM (container-based).

Service chain consolidation

OpenBox[7], SNF. xOMB[2] and CoMb[5] optimize the graph to reuse some basic components and optimize the software classification. None of those work implements factorization of session management, or support flow tempering.

NFV Dataplanes

E2[6], OpenBox[1], OpenMB and xOMB[2] focus less on low-level. They provide controllers and discuss more optimal placement, a problem not tackled in this work. While E2 provides "bytestream vports" their design is not mentioned.

On-the-fly protocol tempering

NetBricks[4] and mOS[8] both implement a TCP stack with some similar monitoring abstractions but no support for modifications. They do not provide a generic non-TCP specific stream abstraction nor the session scratchpad facilies unified for all middleboxes.

[1] A. Bremler-Barr, Y. Harchol, and D. Hay. Openbox: a software-defined framework for developing, deploying, and managing network functions
[2] J. W. Anderson, R. Braud, R. Kapoor, G. Porter, and A. Vahdat. xomb: extensible open middleboxes with commodity servers
[3] T. Barbette, C. Soldani, and L. Mathy. Fast userspace packet processing
[4] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker. Netbricks: Taking the v out of nfv
[5] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi. Design and implementation of a consolidated middlebox architecture
[6] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker. E2: a framework for nfv applications
[7] A. Bremler-Barr, Y. Harchol, and D. Hay. Openbox: software-defined framework for developing, deploying, and managing network functions.
[8] M. A. Jamshed, Y. Moon, D. Kim, D. Han, and K. Park. mos: A reusable networking stack for flow monitoring middleboxes