A Scalable and Flexible Packet Forwarding Method for Future Internet Networks

Andrzej Beben, Piotr Wiśniewski
Warsaw University of Technology
Warsaw, Poland
{abeben, p.wisniewski}tele.pw.edu.pl

Jordi Mongay Batalla
National Institute of Telecommunications
Warsaw, Poland
jordim@interfree.it

George Xilouris
NCSR Demokritos
Athens, Greece
xilouris@iit.demokritos.gr

***Abstract*—The paper proposes a novel flexible packet forwarding (FPF) method designed for Future Internet networks. It follows the source routing principle at an inter-domain level and applies the list of domain customized identifiers to forward packets on the end-to-end path. The main features are capability to introduce flexible routing path selection, native support for multipath and multicast packet transfer, ability for exploitation of advanced in-network packet processing as, e.g., content recoding and caching. The performance and scalability of FPF approach were evaluated by experimentation on developed prototype as well as by scalability studies assuming Internet-scale network scenario.**

Future Internet, source routing, packet forwarding, SDN

I. INTRODUCTION

The severe limitations of the Internet architecture motivate research towards Future Internet (FI), also called New Generation Network (NWGN) or Internet 3.0 [1, 2]. The main constraints stem from the ossification of TCP/IP architecture that prevents innovations at the network level. This ossification comes from: (1) the global scope and location dependency of the IP addresses, (2) destination sink tree routing (3) lack of multi-path transfer, (4) hardly scalable multicast in large-scale, multi-domain networks, and (5) closed routers ("walled gardens" created by vendors) that render the implementation of new packet processing impossible. Current research approaches to obviate these limitations focus on Software Define Networking (SDN) [3] that enables implementation of novel management and control mechanisms by providing clear separation of control and data planes.

In this paper we propose and evaluate a Flexible Packet Forwarding (FPF) method that follows the source routing principle (which offers high flexibility in routing path selection) and is open for innovative in-network packet processing functions required by FI applications in the data plane. Its features go beyond the State of the Art on forwarding mechanisms, as shown in Section II. Besides the specification of FPF method (Section III), the main achievements of the presented research are the development of both software and hardware prototypes (Section IV) and the performed experiments proving that FPF performance is slightly better than IP router (Section V). Moreover, in Section VI we present scalability analysis showing that FPF is suitable for Internet-scale networks. We believe that FPF is a step forward towards the extension of the SDN concept by instilling new capabilities in the data plane while improving SDN scalability thanks to the reduction of state information in SDN forwarders [4].

This work was undertaken under the Pollux II IDSECOM project supported by the National Research Fund Luxembourg (FNR) and the National Centre for Research and Development (NCBiR) in Poland.

II. ANALYSIS OF RELATED WORKS

One of the main FI challenges is to design effective routing and forwarding methods that overcame limitations of the IP protocol. The key objectives are providing more flexible routing path selection and enabling innovative in-network packet processing, while assuring scalability of the solution. Many of recently proposed approaches are based on the source routing principle, which is not really a new idea. The original studies on exploiting source routing in IP networks are presented in [5]. Authors proposed to extend packet header with *route sequence*, which includes addresses of intermediate nodes on the path towards destination. This solution introduced several advances, e.g., enlarged flexibility and support for multicast, but it was not widely accepted due to significant overhead of packet header and security threats coming from possible attacks by replacing the addresses of intermediate nodes in the packet header. Similar constraints have MPLS stacking approach [6]. In order to overcome these limitations, the *pathlet routing* proposal [7] uses node identifiers (vnode) instead of IP addresses of intermediate nodes. The end-to-end routing path is created as a concatenation of path segments, called pathlets. A pathlet may be defined locally inside one Autonomous System (AS) or may span several ASs. In the latter case, node identifiers must be globally unique, so the node identifier is extended to the pair (AS number, vnode). From the development point of view, the *pathlet routing* requires deployment of new forwarding entities.

The original approach for source routing multicast was proposed by LIPSIN [8]. It exploits Bloom Filters (BF) for creating multicast tree identifiers, which are used by intermediate nodes to forward packet copies to selected output ports. The BF uses hash functions that significantly reduce overhead but suffer from false positive outcomes.

The *segment routing* [9] is a new approach for source routing that has been recently proposed by IETF SPRING WG. It assumes that nodes forward packets on the basis of a list of segments included in the packet header. Each segment is a 32-bit-long identifier that can be used either for packet forwarding or packet processing (e.g., multicast). The *segment routing* is flexible and open to innovations, but constant segment size leads to scalability constrain and security threats.

The proposed FPF method has been designed and developed in parallel with *pathlet* and *segment routing* proposals, resulting in some similarities with them. Nevertheless, the FPF main advantages over *pathlet* and *segment routing* correspond to: (i) local (defined in node) scope of LIDs, which significantly reduces LID size, (ii) variable size of LID, which makes feasible adaptation to domain size and (iii) openness for innovative in-network packet processing. As we show later in Section VI, the first two features are of the great importance on FPF scalability.

III. Flexible Packet Forwarding Method

The proposed Flexible Packet Forwarding method assumes that packets are forwarded based on the domain-specific, Local IDentifiers (LIDs) included in the packet header. The vector of identifiers determines the unidirectional end-to-end routing path through a sequence of domains towards destination. Each LID is defined in local scope, so its structure and semantics is understandable only by Forwarding Entities (FE) located within a given domain. The FPF method enables FEs to maintain only the neighborhood information, i.e., how to forward packet to the peering FEs. This feature significantly reduces the LID size and consequently the size of forwarding tables. Due to the open nature of LIDs each domain may define, aside from basic packet forwarding function, other specific packet processing functions as, e.g., enforcing QoS handling, sending packet copies to multicast tree, storing content in the cache. The FPF header, defining the routing path, is attached and removed by the edge FEs located close to the source and the destination. The ingress edge FE assigns packets to routing paths using packet filters. They are defined by the network control plane in an explicit or implicit manner, as explained below in Section III B.

The FPF method assumes that the network control plane is in charge of calculating the end-to-end routing paths by concatenation of LIDs defined by consecutive domain on the routing path. It can use of any multipath routing protocols, e.g. as proposed by *pathlet routing* [7], *Routing Awareness Entity* [10], or it can be set even manually imposing the end-to-end routing inside of one network domain. Therefore, the establishment and selection of end-to-end routing paths are not within the scope of this paper and we only assume that such paths do exist.

Fig. 1 presents the concept of FPF. Let us consider an exemplary packet flow, delivered from domain C to domain A using routing path C-B-A, which differs from IP routing going through domain D. When FPF is being used, the edge FE located in domain C intercepts packets matching the packet

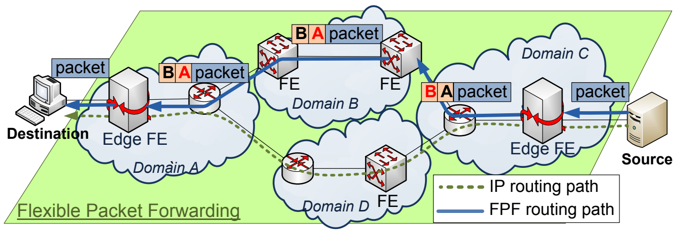

Fig. 1. The concept of FPF method

filter and encapsulates them with the FPF header. This header includes the vector of LIDs, [LID_B, LID_A], which determines successive FEs on the path towards destination. Each FE uses its LID from the FPF header. In our example, the LID_B defines how to forward packet from the edge FE located in domain C towards the first FE located in domain B. Then, the first FE in domain B forwards the packet to the second FE in domain B (see Fig.1) on the basis of the information provided by LID_A. LID_A is also used by the second FE to forward the packet to domain A (destination edge FE). Note, that LID_A is used by both FEs located inside domain B and both FEs know where and how the packet must be forwarded. Finally, the destination edge FE in domain A removes the FPF header and sends packets directly to the destination device.

FPF has been designed to coexist with different network technologies; i.e., it may exploit different underlying packet transfer technologies between peering FEs and it even allows for delivering any type of data units between the source and destination domains. In this case, a correct protocol specific filter should be defined at the ingress edge FE, which enables packet classification and en/de-capsulation of FPF header.

Another important feature is flexibility in the inter-domain routing selection thanks to the fact that inter-domain routing is based uniquely on the information included in the packet header (LID vector). For example, the network control plane could decide special routing for any flow set, ranging from micro flow (or even one single packet), through any level of aggregated flows, up to all flows going towards the same destination (destination sink tree). FPF approach natively supports multipath packet transfer allowing a given flow to be delivered on multiple routing paths. Moreover, the FPF makes the dynamic changes of routing rules feasible just by updating the vector of LIDs inserted by the edge FE.

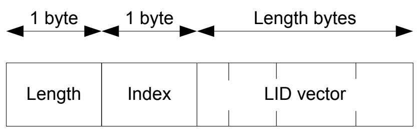

Fig. 2. The FPF packet header

The FPF header, depicted in Fig. 2, covers the following fields: (1) *Length* (1 byte) indicates the number of bytes required for storing the LID vector in the header; (2) *Index* (1 byte) indicates the offset to the first byte of the LID used in the current domain. The *Index* value is increased by the border FE when the packet leaves its domain. The ingress edge FE sets *Index* equal to 0 after classification and it increases *Index* in the case when the ingress edge FE is also the last FE in this domain; and (3) *LID Vector* includes the sequence of LIDs corresponding to the inter-domain routing path. We assume that LIDs used inside a domain should be of constant size in order to simplify matching process in FE forwarding table; but different domains may use LIDs of different sizes. Anyway, the LID size should be minimized in order to reduce the overhead (this will be discussed in Section VI).

Note that FPF header slightly increases the original packet size. Therefore, the underlying transmission systems may need segmentation/reassembly function for transferring FPF packets.

A. FPF forwarding process

The forwarding process is performed by each FE when it receives a packet with FPF header. Each FE keeps forwarding table (Input interface, LID) → (forwarding rule), as presented in Table I. This table shows exemplary forwarding rules corresponding to the packet transfer on Ethernet and multicast tree. All LIDs inside a given domain have constant size, so, the FE uses constant length matching LID in the forwarding table. The LID identifies the forwarding rule that describes "how to forward packet to the next FE". Note that this information has a long-term validity and it is updated by the control plane according to routing changes.

TABLE I. EXEMPLARY FORWARDING TABLE IN FE

Input interface	***LID***	***Forwarding rule***
#1	{0xa1}	Send packets to queue Premium on the interface #9 using Ethernet frame with destination address a1:01:02:ff:01:d9
#3	{0xf1}	Create packet copies for the following multicast leafs: 1) update LID vector to 0xa1e3d3 and handle packet by forwarding engine 2) update LID vector to 0xaabce8 and handle packet by forwarding engine

Upon receiving a packet with FPF header, the FE performs the following steps: (1) it registers the input interface *if_in* where the packet arrived; (2) it performs sanity check of *Index* filed: if *Index* value is greater than *Length*, then FE drops the packet; if *Index* is equal to *Length*, then FE removes the FPF header and sends the packet directly to the destination. Note that this is the case of last FE in the path; (3) it performs lookup in the forwarding table using the *if_in* and the LID pointed by *Index* to find the appropriate forwarding rule, (4) if the packet goes to the next domain, the FE increases the value of *Index* to point out at the next LID and (5) it processes the frame accordingly with forwarding rule and sends the packet to the output interface.

B. Packet encapsulation in edge FE

Packet encapsulation with FPF header is performed by the ingress edge FE based on the packet encapsulation (rules) table, which includes: (1) the multi-field packet filter to identify packet flow, (2) the LID vector used for packet forwarding, (3) timeout to remove expired filters and, optionally, (4) other flow specific information as, e.g., session identifier, traffic profile, etc. This optional information is used for traffic policing, shaping, multicasting, etc. For TCP/IP traffic, multi-field packet filtering is performed on the basis of 5-tuple including: *Src.* and *Dest. addresses* IPv4 or IPv6; *Protocol number* (IPv4) or *next header* (IPv6); and *Src.* and *Dest. port numbers* (optional). In case of non-IP protocols, packet filters specific for such protocols should be provided. The edge FE intercepts incoming packets and performs lookup in the encapsulation table. If a packet matches one of the packet filters, the FE encapsulates the packet following the encapsulation rule, resets the timeout and passes the packet to the FPF forwarding process. If a packet does not match any filter, then it is forwarded by using standard forwarding (e.g., IP) or it is dropped. The rules in encapsulation table are managed by higher layer (network management and control) and may be set in two ways: explicit and implicit.

1) Explicit approach

In this approach, packet encapsulation rules are configured per flow in advance before the beginning of flow transmission. The packet filter configuration is done explicitly by the network management and control system, e.g., SDN controller, and it is based on the characteristics of the expected flows. For each new flow request, the controller selects the transfer path (from the set of available established off-line paths) and configures appropriate packet filter in the edge FE. The explicit approach is suitable for the system with distinct flow set-up phase such as Next Generation Networks (NGN) or some Information Centric Networks (ICN) [11].

2) Implicit approach

In this approach, the packet encapsulation rules are configured dynamically on the basis of Deep Packet Inspection (DPI) [12]. When the first packet of the new flow arrives to the edge FE and there is no matching rule in the encapsulation table, then a copy of the packet is passed to the so-called Flow Aware Classifier (FAC). The FAC performs DPI on the copy of the packet, while the original packet is forwarded by standard routing rules. Some protocols may require more than one packet in order to univocally classify the flow; for example, video flow using RTP protocol can be identified by checking the increment of sequence number in the RTP header between two successive packets. Therefore, a packet counter controlling the packets passed to DPI may be required for some protocols. Following the higher layers information assessed during the DPI (e.g., the flow is VoIP stream) and the management instructions for such a kind of flows, a new packet encapsulation rule is configured in the encapsulation.

The DPI algorithms may inspect protocol/service/content related fields that are retrieved by information above L4. This approach facilitates flexibility in the creation of the filters and has been proposed in order to cope with the emergence of multimedia services, terminal mobility as well as the evolution of End User model from single direction service consumer to content creator and distributor (i.e., prosumer) [13].

IV. FPF PROTOTYPES

We developed the FE prototypes to prove theirs feasibility and evaluate performance of FPF method. They were implemented on software and hardware platforms, i.e., Linux based server and specialized network processor board - EZchip NP-3 EZappliance (EZappliance for short) [14].

Prototypes utilize the commonly used protocols: Ethernet 802.3 at link layer, IPv4/IPv6 at network layer, and TCP/UDP for transport. We implemented FPF at the 2.5 layer, so that IP packets are encapsulated with the FPF header and inserted into Ethernet frames. We use experimental value of ether type (0xcccc) of Ethernet header to distinguish FPF packets. Moreover, FPF coexists with IPv4/IPv6 forwarding.

Fig. 3 depicts the functional block diagram of FE prototype. FE is composed of forwarding and encapsulation engines, FAC, and forwarding and encapsulation configuration agents. The forwarding engine executes LID-based forwarding following the rules handled by the forwarding agent. The encapsulation engine intercepts packets received from an access network, classifies them based on 5-tuple packet filter and encapsulates them with the FPF header according to the encapsulation table. The FAC keeps track of running flows and manages entries in the encapsulation table (via encapsulation agent). In particular, FAC sniffs incoming IP traffic, detects new flows, and finally classifies them (using DPI). Sniffing is performed to decide if a packet belongs to a new flow enabling DPI to be executed only for a few packets per new flow.

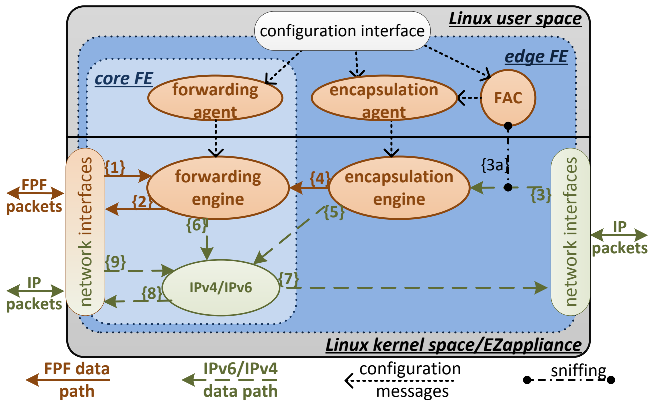

Fig. 3. FE prototype diagram

Edge FE has all modules, while core FE is simplified and uses only forwarding engine and configuration agent.

The FPF packets traversing through FE are processed exclusively by the forwarding engine, arrows {1},{2} in Fig. 3. The FPF packets exiting the core network are decapsulated by forwarding engine {1}, and forwarded by IP logic {6},{7}. The IP packets received by core FE are processed according to the standard IP stack {9},{8}.

All IP packets coming to edge FE are intercepted and classified {3}. If a packet matches a rule in encapsulation table, it is encapsulated and then switched by the forwarding engine {4},{2}. Otherwise, it is passed to IP stack {5},{7}/{8}.

A. Linux based prototype

The basic FE prototype was developed for Linux platform. The source code is available at tnt.tele.pw.edu.pl/software_fpf.php. The encapsulation and forwarding modules are implemented as loadable Linux kernel modules to assure high efficiency. The configuration agents are developed as user space programs in *Python* and communicate with kernel modules through *netlink* inter-process communication provided by *libnl* library.

The forwarding engine is implemented in the Linux kernel as a new protocol handler (called FPF protocol handler), which is called-back when a FPF packet distinguished by 0xcccc ether type value arrives to the kernel. Then the forwarding engine: performs sanity check, and *i*) decapsulates the IP packet and injects it back to the bottom of protocol stack with *netif_receive_skb* function (if index field is equal to length field), or *ii*) performs LID-based forwarding and transfers the packet to the chosen network device driver (with dev_queue_xmit function) or drops the packet if LID value is not found in the forwarding table. For implementation simplicity, we have assumed that each LID is one byte long. The FE Linux prototype supports the following link layer technologies: Ethernet, VLAN Ethernet and GRE tunnels.

The kernel encapsulation module is based on *netfilter* framework. It exploits two "hooks" for each IP version. The first hook is used to intercept incoming IP packets at pre-routing stage, whereas the second hook is used to intercept packets coming from local processes (we assume that FPF node can generate FPF packets). If a packet matches any encapsulation rule, then it is encapsulated with the appropriate FPF header and injected back to the bottom of protocol stack.

The FAC exploits *LibPCAP* library to capture packets and parse IP and transport headers. A hash-map based *Flow* class is introduced to keep track of running flows. Notice that the packet processing path through kernel is not affected. All the information required by FAC is copied, allowing packets to be forwarded unaltered, using the default forwarding paths, until the classification of the flow is done.

The classification is based on the heuristic and the finite state machines matching algorithms. They are capable of classifying flows based on the L7 protocol information like RTP header extensions or SEI messages (Supplemental Enhancement Information) contained in the payload for MPEG-4/SVC headers.

B. EZappliance based prototype

The core FE was also developed on EZappliance hardware platform. EZappliance contains NP-3 network processor [14] that provides flexible data packet processing, and build-in Linux based host CPU system that performs control plane functionalities [15]. The NP-3 processor itself is composed of a pipeline of heterogeneous Task-Optimized Processors (TOPs) tailored for different stages of packet processing (TOPparse, TOPsearchI, TOPresolve, TOPsearchII, TOPmodify) [15].

The core FE was implemented in dedicated NP-3 processor assembly language. For incoming Ethernet frames, the TOPparse code analyses ether type field, parses FPF header, performs sanity check and constructs search key for FPF forwarding table. TOPsearchI processor looks up LID values in forwarding table and, next, the TOPresolve makes forwarding decision and picks optional modifications. Finally, TOPmodify processor performs modifications (increase of index filed or decapsulation). Our implementation of FE supports Ethernet and VLAN Ethernet.

V. Performance Evaluation

This section focuses on evaluation of developed FPF prototype. We evaluate performance of three basic FPF modules, i.e., (1) FPF forwarding engine (throughput), (2) encapsulation engine (throughput for increasing number of running flows), and (3) FAC (throughput for increasing number of new flows). Moreover, we qualify the FE performance by comparing it with IPv6 software router.

A. The FPF forwarding engine

The FPF forwarding engine is used in both core and edge FEs, so its performance is crucial for the deployment of FPF method. Following the RFC 2544, the throughput metric is defined as the maximum rate of packet stream forwarded without any packet losses, expressed in frames per second (fps). We measure IPv6 throughput on the same machine as a point of reference.

The testbed consists of Device Under Test (DUT) and Automatic Test Equipment (ATE) interconnected by two 1 Gbps links in ring topology. The ATE generates the measurement stream (FPF or IPv6) and sends it to DUT by the first link. Then the stream is forwarded by preconfigured DUT and sent back to ATE by the second link. As ATE, we used Spirent TestCenter SPT-2000 chassis with hardware traffic generator/analyser card CM-1G-D4. The prototype of FPF forwarder (DUT) runs on either standard off-the-shelf Dell PowerEdge R620 server (Linux deployment of FE) or EZappliance. The server is equipped with Intel Xeon Processor *E5-2630L@2 GHz*, 12GB RAM and Broadcom NetXtreme BCM5719 network card and runs stack openSUSE 12.3.

To measure the throughput, we applied binary search method with resolution of 1Mbps. We performed measurements for at least the following frame sizes: 78B, 96B, 128B, 196B, 256B, 384B, 512B, 1024B and 1518B (includes Ethernet checksum). As recommended in RFC2544, each test iteration lasted 60 seconds (at least $4.8*10^6$ frames forwarded by DUT). We repeated each test several times to delimit the confidence intervals. As the 95% confidence intervals in all tests (with the exception of FAC performance tests, see Section V.C) are lower than 2% we do not present them for clarity.

In the first test scenario we evaluated performance of Linux based prototype. Fig. 4 presents FPF and IPv6 throughputs measured for different frame sizes and plotted against maximal theoretical throughput of 1Gbps Ethernet link. We observe that FPF throughput is noticeably higher than IPv6 throughput for frames sizes smaller than 256B and, above that size, both throughputs are equal to maximal theoretical value.

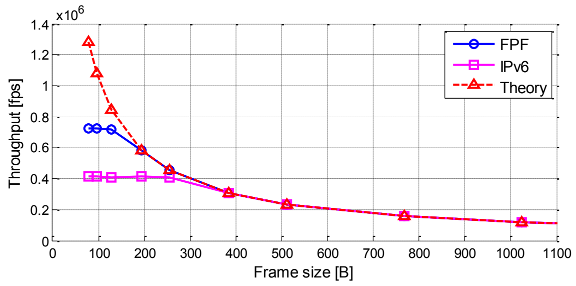

Fig. 4. Forwarding throughput of FE vs. IP router for software prototype

We conducted throughput evaluations in two other hardware devices and different Linux distributions. We observe that FPF and IPv6 throughputs are on a comparable level when network interface is a bottleneck, whereas FPF throughput is higher (up to twice) when the bottleneck is due to CPU limitations. This corresponds to the simpler packet processing in the FE in comparison to the IP stack. The FPF forwarding is less complex thanks to small and constant size LIDs. Such LIDs significantly simplify lookup process in forwarding table due to limited table size and simpler packet matching (constant size vs. the longest prefix matching).

In the second test scenario we evaluated performance of the network processor based prototype. As EZappliance is capable of forwarding several gigabits of traffic per second (around ten-odd) and the interfaces are limited to 1 Gbps, we interconnected a few interfaces creating local loops. Next, we prefetched FPF forwarding table with one LID per interface enabling the same FPF stream to be concurrently forwarded by DUT a number of times. As a result, thanks to the flexibility of FPF method, we have measured FPF throughput, which was not possible in case of IP routing.

Fig. 5 presents the FE throughput plotted against theoretical value. We calculated the theoretical value assuming the typical bandwidth of SPI4.2 interface (connecting NP-3 with Ethernet aggregator [14]), which is equal to 12.8 Gbps. We observe that the measured throughput differs slightly from the theoretical value only for small frame sizes (under 128B). We conclude that FPF engine is able to forward frames with the maximum theoretical throughput of EZappliance proving the feasibility of FPF method applied to a specialized network hardware.

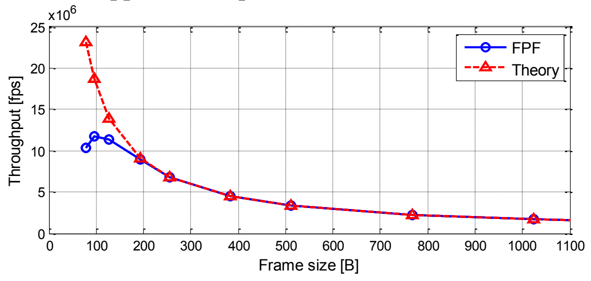

Fig. 5. Forwarding throughput of FE for network processor based prototype

B. The encapsulation module

The performance of the edge FE is determined by the number of concurrently classified flows (by encapsulation module) and their total input rate. We analyzed the ability of edge FE to intercept and handle large number of running flows up to 16 thousands (16K). Let us remark that the performance of core FE is independent from the number of running flows as it depends only on the number of paths which is intrinsically small (as shown in section VI).

The measurement method and scenario are the same as described in previous subsection, but in this case, the number of streams generated by the ATE ranges from 1K to 16K. Each stream, distinguished by unique IPv6 source address, has equal rate. We set the upper bound of number of streams to 16K since this is the upper limit of G711 VoIP (64 kbps) streams on 1 Gbps link. In order to measure the worst case scenario we configured DUT (Linux prototype) to intercept, encapsulate packets from all incoming streams.

The throughputs measured for different number of ingress streams are plotted in Fig. 6 against theoretical link throughput. We can observe that the throughput slightly decreases when the number of flows increases, which is caused by the growth of classification complexity of packets (look-up time in encapsulation table). Nevertheless this difference is small since the encapsulation and forwarding time is constant. Moreover, the throughput degradation relates only to relatively small frames (smaller than 384B), proving feasibility of edge FE.

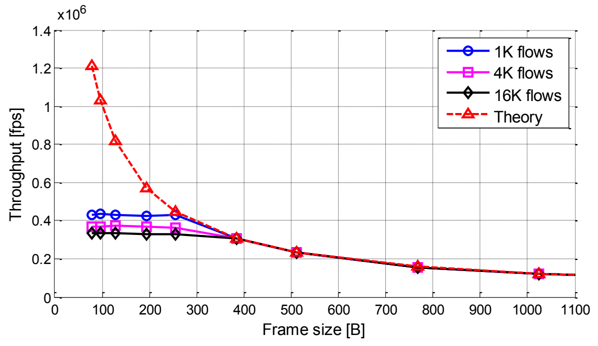

Fig. 6. Encapsulation throughput of Linux prototype

C. Flow Awareness Classifier

In order to evaluate the performance of FAC, we need to assess the correlation between the capability of capturing packet data and the arrival rate of new flows. In this context DUT (Linux-based edge FE) is tested under traffic load with variable flow arrival rate and two frame sizes: 128 and 1528B. In order to exclude problems caused by the network interface or the limitation of its hardware, the traffic is generated off-line by software generator, and captured in *LibPCAP* compatible trace files. Concretely, we created four trace files with 10, 100, 1K and 10K flows and the same number of packets in all the files. These files are read by the FAC module using the highest reading rate (allowed by the hardware and upper bounded by the hard disk transfer rate). The tests compare the time T_1 that FAC uses to read the file and to process the flows (sniffing all the packets and performing DPI for the first packet of each flow) and the time T_0 of reading the file without performing any other extra operation.

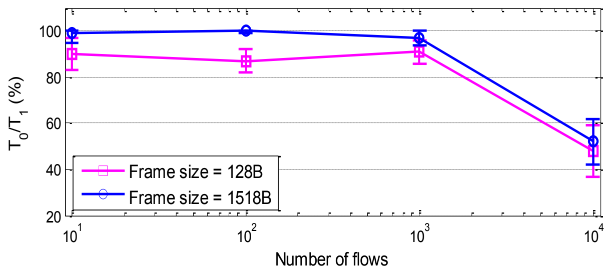

Fig. 7. FAC performance (percentile of T_0/T_1) vs. number of flows

Since the test results may depend on the CPU occupation and system clock, we repeated the tests several times in order to extract the confidence intervals. These intervals resulted in the range up to 18% of the mean values. As we may observe in Fig. 7, the processing of small packets is more demanding than the processing of large ones, which can be imputed to the sniffing operations performed by *LibPCAP* library. Moreover, the sniffing capacity is irrespective of the number of flows until a certain limit, which is sensible since the sniffing operations do not change for increasing number of flows (note that the table with the currently running flows has constant length equal to 20,000 flows in all the tests).

From the graph, it can be deduced that FAC is able to capture and process DPI over 1K simultaneous flows, but when the number of flows reaches 10K, t hen the edge FE needs twice more time for inspecting the flows. As a conclusion, we may affirm that the DPI mechanism limits the number of simultaneous flows that the edge FE can handle. Let us remark that the implemented code of DPI has not been optimized.

VI. SCALABILITY EVALUATION

In this section, we assess scalability of FPF method. The main questions about FPF scalability are related to the overhead introduced by FPF header as well as the performance of packet processing performed in FEs.

The FPF method follows the source routing principle, where each packet includes LID vector determining the inter-domain routing path towards destination. The FPF overhead depends on the number of LIDs in the header and the sizes of individual LIDs. The number of LIDs is correlated with the length of inter-domain path because, in principle, each domain "uses" one LID from the header. Therefore, the size of LID vector can be estimated from the length of inter-domain paths.

In order to calculate the length of inter-domain routing paths in the Internet, we use the CAIDA dataset [16], which contains data describing the Internet topology. We have analyzed the relations between the domains provided by RIPE [17] and introduced shortest path algorithm for obtaining the pdf of AS path length (over $1.3*10^9$ inter-domain paths connecting 36,878 domains), which are shown in Fig. 8.

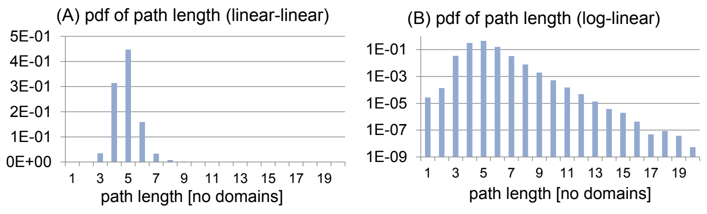

Fig. 8. The pdf of AS path length, where (a) linear scale, (b) logarithmic scale

From these plots, we can observe that the majority (99%) of paths cross less than 7 domains. The average length is 4.88 and the longest path crosses 20 domains. Briefly, the number of LIDs into the LID vector for the 99% of domains is 7 and the longest LID vector contains 20 LIDs.

In addition to the number of LIDs (into the LID vector), we must consider the length of individual LIDs. Basically, LIDs are defined in each domain to univocally identify the intra and inter-domain routes with associated per domain packet transfer behavior between two consecutive domain ingress points (see Fig. 1). In addition, an operator may define a number of specific LIDs to trigger special packet processing functions, e.g., multicasting.

The LID size depends on the number of: (1) routing paths towards neighboring domains, which is directly related to domain degree (DD), (2) per domain packet transfer behaviors offered on the paths (DB), and (3) special packet processing rules supported by the domain (SR). The upper bound of LID size (LS^+) expressed in bytes can be calculated by (1):

$$LS^{+} = \lceil log_2(DD * DB + SR)/8 \rceil, \qquad (1)$$

where $\lceil x \rceil$ denotes the minimum integer not lower than x.

We use the upper bound LS+ to estimate the number of bytes required in FPF header. For that purpose, we analyze degree of all domains available in CAIDA dataset. By way of example, we assume that each domain distinguishes three domain packet transfer behaviors for each intra-domain routing path and defines one special LID for each inter-domain link. The obtained results are presented in Table II.

TABLE II. PERCENTILE OF DOMAINS' DEGREE HISTOGRAM AND LID SIZE

Percentile [%]	Number of domains	Domain degree (max.)	Number of LIDs (max.)	LS^{+} [byte] (max.)
90%	33190	6	24	1
99%	36509	60	240	1
99.50%	36693	128	512	2
99.99%	36874	2000	8000	2
100%	36878	2972	11888	2

We can observe that 99% of domains need only 1 byte-long LID to identify all the paths, due to their small DD. Only backbone domains (tier 1) and large tier 2 domains may require 2 bytes for the LIDs.

Based on the above analyses and taking into consideration *Length* and *Index* fields we may conclude that, for majority of end-to-end paths, the FPF header is shorter than 10 bytes. In the worst case, i.e., for the longest routing path which would hypothetically cross only large transit domains, the FPF header does not exceed 42 bytes, which is as in IPv6. Since IPv6 scales properly in the Internet, we can affirm that also FPF is scalable to the Internet (as far as overhead is concerned).

The second scalability issue focuses on the complexity of FPF operations. For scalability reasons, we have designed the core FE to be as simple as possible and left any complex packet processing at the edge FE (in line with DiffServ approach).

As we have demonstrated in Section V, the core FE has slightly better forwarding performance than IPv6 router. On the other hand, the edge FE forwarding is more complex than edge IP router because it performs packet interception and encapsulation. Moreover, the edge FE handle information about all running flows in encapsulation table. Anyway, the performance tests confirmed that our simple (software-based) implementation of edge FE running on ordinary hardware can handle up to thousand active flows. If edge FE must handle more traffic, then more edge FEs could be easily deployed.

Based on the above discussion, we conclude that FPF forwarding approach is similar (as far as scalability concerns) to DiffServ approach, which is regarded as scalable. So, we may forecast not many scalability issues in FPF method.

VII. SUMMARY

This paper proposes novel Flexible Packet Forwarding method, where nodes forward packets based on a vector of Local Identifiers (LID) included in the packet header. Our method allows for flexible routing path selection, enables seamless multi-path and multicast routing at the inter-domain level, which offers new opportunities for traffic engineering at the inter-domain level. Moreover, the FPF method is open for implementation of innovative in-network packet processing. In the paper, we present developed FPF prototypes implemented on Linux based server and EZchip NP-3 network processor. We have made accessible the Linux based implementation to the research community. The code is open-source and can be further developed for specific purposes.

The performed experiments confirmed that FPF node achieves slightly higher throughput than corresponding IPv6 router due to less complex packet processing. The scalability studies showed that FPF method will fittingly scale in the Internet thanks to the limited size of LID vector because of limited length of the inter-domain paths and small degree of domains in the Internet. An important FPF feature is that FPF nodes use only local information, i.e. how to forward packets to the next domain. Any per flow information as packet filters are kept in the edge FE (configured by management and control plane).

ACKNOWLEDGMENT

The authors would like to thank Jarosław Śliwiński for fruitful discussions and support in FPF implementation.

REFERENCES

[1] S. Paul et al., “Architectures for future networks and the next generation Internet: A survey”, CompCom No. 34, 2011

[2] T. Aoyama, “A New Generation Network: Beyond the Internet and NGN”, IEEE ComMag, May 2009

[3] A. Lara et al., “Network Innovation using OpenFlow: A Survey”, IEEE Com. Surveys & Tutorials, 2013

[4] M. Soliman et al., “Source routed forwarding with software defined control, considerations and implications”, CoNEXT Student, 2012

[5] P. Francis and R. Govindan, "Flexible Routing and Addressing for a Next Generation IP," ACM Comp. Comm. Review 1994

[6] H. Gredler, “Supporting Source/Explicitly Routed Tunnels via Stacked LSPs”, IETF draft, draft-gredler-spring-mpls-06, 2014

[7] P. Godfrey et al., “Pathlet routing”, ACM SIGCOMM 2009

[8] P. Jokela et al., “LIPSIN: line speed publish/subscribe inter-networking”, SIGCOMM Comp. Com., 2009

[9] C. Filsfils et al., “Segment Routing Architecture”, IETF SPRING WG, draft-filsfils-rtgwg-segment-routing-01, 2014

[10] G. Garcia et al., “COMET: Content mediator architecture for Content-Aware Networks”, FNMS, 2011

[11] G. Xylomenos, et al., “A Survey of Information-Centric Networking Research”, IEEE Com. Surveys & Tutorials, 2013

[12] T. Nguyen et al., “A survey of techniques for internet traffic classification using machine learning” IEEE Com.Surveys, 2008

[13] TV and Media, “Identifying the need of tomorrow’s video consumers”, ERICSSON Consumer Insight Report, 2013

[14] NP-3 and EZappliance Product Briefs, at www.ezchip.com

[15] R. Giladi, “Network Processors - Architecture, Programming and Implementation”, Morgan Kaufman, 2008

[16] The Cooperative Association for Internet Data Analysis, http://www.caida.org/.

[17] RIPE Network Coord. Centre, “Routing Information Service”, http://www.ripe.net/data-tools/stats/ris/ris-peering-policy