Blind Packet Forwarding in a Hierarchical Architecture with Locator/Identifier Split

Irfan Simsek, Yves Igor Jerschow, Martin Becke, Erwin P. Rathgeb
Computer Networking Technology Group
University of Duisburg-Essen
Essen, Germany
{irfan.simsek, jerschow, martin.becke, erwin.rathgeb}@iem.uni-due.de

***Abstract*—In our previous work, we discussed the issue that with existing techniques we can either provide end-to-end information security or establish smart in-network services. Using the example of Blind Packet Forwarding (BPF), we demonstrated the feasibility to simultaneously realise both goals. However, the basic BPF design is not sufficiently scalable and lacks support for mobility and multihoming. This paper focuses on the issues of the basic BPF design and resolves them by applying the Locator/Identifier Split approach. We extend our BPF design and especially adopt the features of the Hierarchical Architecture for Internet Routing (HAIR). Additionally, we present our prototype implementation of the improved BPF, which benefits from the flow-based forwarding in OpenFlow, and evaluate it.**

***Keywords*—Future Network security, Blind Packet Forwarding, PEKS, Locator/Identifier Split, HAIR, OpenFlow.**

I. Introduction

Meanwhile, there exist a wealth of approaches for a *Future Network Architecture (FNA)*. Although these approaches differ in their orientation, they all suggest that a network should be service-oriented and flexibly orchestrated from atomic smart in-network services [1]. These in-network services require access to various control data signalled in different ways to utilise the complete functionality of the orchestrated network, and the diversity and amount of the processed control data rises progressively. Hence, the communication endpoints have to allow more and more access to information about themselves. However, the in-network services are able to sniff or to spoof information. Furthermore, even third parties can sniff or spoof information while it is transferred in cleartext. Beside these considerations for a FNA, the de facto method applied so far to provide information confidentiality and integrity for two communicating end points is end-to-end encryption. However, in-network services have then no longer access to the encrypted control data and they cannot accomplish their tasks anymore so that it becomes impossible to utilise the benefits of the FNA approaches. If we look back on today's Internet architecture, control data such as addresses for packet forwarding has been always transferred in cleartext so far.

These issues indicate that it is only possible to realise one of the two goals – information security and smart in-network services – at once. But by applying the *Public key Encryption with Keyword Search (PEKS)* algorithm [2], we demonstrated the feasibility to simultaneously establish smart in-network services and to provide information confidentiality and integrity in case of *Blind Packet Forwarding (BPF)* [3]. We chose this in-network service as an example because packet forwarding is one of the basic services required for most network architectures. BPF allows to correctly match masked packet addresses with masked routing table entries by means of PEKS so that confidentiality and integrity for packet addresses are provided during transmission as well as during processing by network nodes.

In the basic BPF design, we constructed a generic hierarchic address structure, where a host address in cleartext consists of two parts. The first part holds the ID of the network node responsible for the local network and the second part is the ID of the host in that local network. These two parts of a host address are masked by applying PEKS. In a masked host address, the first part is the ciphertext of the network node ID, which has been encrypted with the public key of the network node, and the second one is the ciphertext of the host ID, which has been encrypted with the public key of the host. When we extend this address structure to a more complex one like IPv4 with conventional address classes, the address parts of two communicating hosts have to be encrypted byte by byte according to the netmasks (i.e., without subnetting) used on the route between them. Additionally, a source host requires the public key of gateway nodes (i.e., nodes at the border of two neighbouring domains) on the route to the destination endpoint. Furthermore, since routing table entries are masked, they cannot be aggregated so that conventional super- and subnetting is not supported. Thus, while demonstrating the general feasibility, our first BPF design is not sufficiently scalable, and furthermore, it lacks support for mobility and multihoming.

In order to resolve the issues identified above, we extend our BPF design using the *Locator/Identifier (Loc/ID) Split* approach [4]. While an IP address identifies an endpoint and describes its network attachment point (locator) at once, the Loc/ID Split principle separates the locator functionality from the identifier. This principle supports scalability, mobility and multihoming by design and is regarded as the de facto addressing standard for FNA [5], [6]. In the extended BPF, we separately mask the identifier and locator of an endpoint, which gives us the ability to construct two BPF modes. In the *semi-blind mode*, only identifiers are masked, while we mask both identifiers and locators in the *fully blind mode*. Thus, each mode of the extended BPF provides a certain security level adapted to various use cases. The *Hierarchical Architecture for Internet Routing (HAIR)* [7] is one of the FNA approaches relying on the Loc/ID Split principle. This approach constructs a hierarchical mapping system to bind the locator to the identifier of an endpoint. Additionally, HAIR defines a hierarchical scheme supporting super- and subnetting by

design. Due to this scheme, HAIR provides better scalability, fine-granular mobility and multihoming. Therefore, we adopt the architectural features of HAIR and extend our BPF design. In the previous prototype implementation of BPF, we utilised OpenFlow [8] and NOX [9] as the controller. There, masked packets are handled by the controller hop-by-hop which causes a certain overhead. Therefore, another contribution of this paper is the reduction of this overhead by leveraging the *flow-based forwarding* in OpenFlow.

The rest of this paper is structured as follows. In Section II, we sketch the PEKS algorithm and HAIR, which we have applied for our design. Section III presents and Section IV analyses the extended design of BPF. Our prototype implementation and our evaluation results are discussed in Section V. Finally, we conclude our paper in Section VI.

II. Related Work

PEKS [2] is a cryptographic algorithm which enables to compare two ciphertexts with each other. Hsu *et al.* presented a study of PEKS and its extensions in [10]. In our approach, we adopt the PEKS construction which consists of the following methods:

- $\mathbf{KeyGen}(\mathbf{s}) \rightarrow (\mathbf{A_{pub}}, \mathbf{A_{priv}})$: For a given security parameter s it generates a public/private-key pair.
- $\mathbf{PEKS}(\mathbf{A_{pub}}, \mathbf{W}) \rightarrow \mathbf{E}(\mathbf{W})$: For a given public key A_{pub} and a word W it outputs a searchable encryption of the word.
- $\mathbf{Trapdoor}(\mathbf{A_{priv}}, \mathbf{V}) \rightarrow \mathbf{T}(\mathbf{V})$: For a given private key A_{priv} and a word V it generates a trapdoor for the word.
- $\mathbf{Test}(\mathbf{E}(\mathbf{W}), \mathbf{T}(\mathbf{V}))$: For a given searchable encryption $E(W)$ and a trapdoor $T(V)$ it outputs 1 if $W = V$, otherwise it outputs 0 ($W \neq V$).

We want to note here that this PEKS construction does not allow to decrypt ciphertexts, i.e. the encryption is not invertible.

In the basic BPF design, we defined a basic hierarchic address structure. There, the cleartext address of the host H connected to the network node N is $N.H$. After applying PEKS, the masked address of the host H is $E(N).E(H)$. For a masked routing table setup, we redesigned the Distance Vector Routing algorithm to a blind one. In our design, a masked routing table entry for the network node N contains the trapdoor $T(N)$. For an incoming masked packet addressed to the host H, a network node performs $Test(E(N), T(N_i))$ for each routing table entry i. The packet is forwarded via the port mapped in the entry for which $Test()$ returns 1.

Hoefling *et al.* [11] surveys mapping systems constructed in approaches relying on the Loc/ID Split principle. HAIR [7] is one of them and defines a n-level-based hierarchical scheme, which reflects the current Internet structure. There, local networks are placed at level n, called *Edge*. At levels $n-1$ to 2 we have the so called *Intermediate (INT)* networks which consist only of routers and ensure the reachability between the attached Edges and INTs of the next higher level. We can conceive the INTs as access providers or enterprise networks. The top level of the hierarchy is the so called *Core* providing routing reachability between the INTs at level 2. Each router in the Core belongs to one administrative domain and is under the control of a single ISP, just like in the today's Internet backbone. Levels 1 to n are connected via so called *Level Attachment Points (LAPs)* acting as gateway nodes. A routing domain at level k is connected to a routing domain at level $k+1$ via L_kAP. The organisation of hierarchy levels is not restricted to be globally symmetrical. Domain providers can independently organise their own domains in various hierarchy levels. In order to bind the actual locator to the identifier of an endpoint, HAIR defines a hierarchical mapping system consisting of a *Core Mapping Service (CMS)* and multiple *INT Mapping Services (IMSs)*. While an IMS holds the actual mappings for the endpoints belonging to that INT, the CMS keeps a pointer to the IMS maintaining the current mapping for each identifier. The address of the endpoint D is $Addr_D : LOC_D \mid ID_D$, where $LOC_D = L_1AP_D|...|L_{n-1}AP_D$ is D's actual locator sequence and ID_D is D's identifier. The locator sequence defines the LAPs to be traversed for forwarding a packet from the Core to the destination Edge, which is similar to loose source routing.

III. Construction

In the extended design of BPF we construct two modes. In the first mode, only identifiers are masked, while both locators and identifiers are masked in the second mode. The extended BPF performs the steps defined in HAIR (see Figure 1) for mapping distribution, lookup and packet delivery. We assume that all network nodes, communicating endpoints and mapping servers have already generated a key pair using PEKS and that all required public keys are already exchanged and bound to their owners, and furthermore, the communicating endpoints have already found their identifiers, e.g., via DNS.

A. Semi-blind Packet Forwarding

The semi-masked address of the endpoint D with the identifier ID_D is $smAddr_D : L_1AP_D|...|L_{n-1}AP_D \mid E(ID_D)$. Here, $E(ID_D)$ is D's masked identifier encrypted with D's public key by means of PEKS. LOC_D is D's actual locator sequence and defines the LAPs to be traversed for forwarding the packet from the Core to the $Edge_D$, just like in HAIR. The semi-masked address fields in a packet from the endpoint S to the endpoint D consist of $smAddr_D \mid smAddr_S \mid C(ID_S)$. Here, the last field is the ciphertext generated by conventionally encrypting (e.g., with RSA) ID_S with the corresponding public key of D. It is added, since the PEKS construction used for our design does not enable to decrypt ciphertexts. However, this field can be omitted in case that S's identifier can remain unknown for D. To send a packet back to S, D can set the value $E(ID_S)$ from $smAddr_S$ as its destination identifier. Additionally, S's masked identifier serves as destination identifier of a corresponding destination unreachable message in case that the packet cannot be forwarded to D. Furthermore, S and D can cache encrypted identifiers so that they do not have to encrypt the identifiers for each packet.

1) Masked Identifier Resolution: At the Edge, the masked destination identifier of a packet is resolved to a MAC address and the packet is forwarded to the destination endpoint. In case of an empty neighbour cache, the device S aiming to resolve the masked identifier $E(ID_D)$ of device D broadcasts the request message $\{E(ID_D), E(ID_S), T(ID_S), MAC_S\}$. Here,

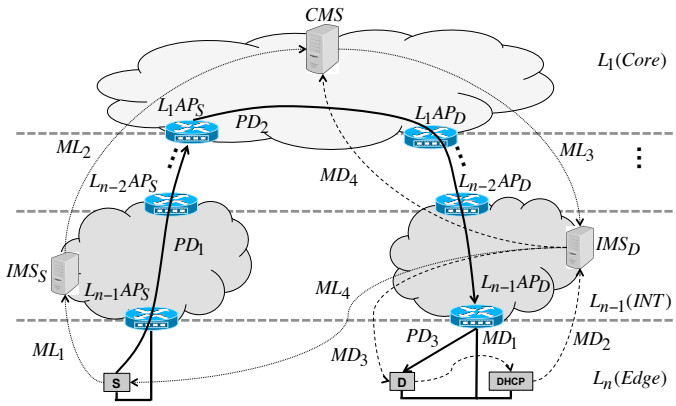

Fig. 1. Mapping distribution, lookup and packet delivery steps.

$T(ID_S)$ is the trapdoor for ID_S generated with S's private key, and MAC_S is S's MAC address. Each device H receiving the request message performs $Test(E(ID_D), T(ID_H))$, which returns 1 only at D. It caches the last three values from the request message in its neighbour cache and sends the reply message $\{T(ID_D), MAC_D\}$ to S, which updates its neighbour cache accordingly. In case that $L_{n-1}AP$, which itself has no identifier, aims to resolve a masked identifier, it uses its locator instead of ID_S to construct the second and third value in the request message.

In case that some entries in the neighbour cache already exist, S resolves the masked identifier $E(ID_H)$ by running $Test(E(ID_H), T(ID_{H_i}))$ for each cache entry i. The entry for which $Test()$ returns 1 contains the MAC address for $E(ID_H)$.

2) Semi-masked Mapping Distribution: The masked identifier of an endpoint newly connected to an Edge has to be reported to the IMS responsible for that Edge and to the CMS so that the masked identifier is bound to a cleartext locator sequence in the semi-BPF mode (see steps MD_1 to MD_4 in Figure 1). First, the endpoint D encrypts its identifier ID_D with its public key and generates the trapdoor $T(ID_D)$ for its identifier with its private key by means of PEKS. Next, D sends $\{E(ID_D), T(ID_D)\}$ to an enhanced DHCP server (step MD_1). The DHCP server maintains the edge-locator $L_{n-1}AP_D$, maps this to D's trapdoor and masked identifier and sends the semi-masked edge-mapping $\{(E(ID_D), T(ID_D)) \Rightarrow L_{n-1}AP_D\}$ to IMS_D (step MD_2). IMS_D maintains the remaining cleartext locator sequence $L_1AP_D|...|L_{n-2}AP_D$. This entry can be manually configured or dynamically set up by running an enhanced traceroute, e.g., for CMS.

A semi-masked mapping table entry for an endpoint H is $(E(ID_H), T(ID_H)) \Rightarrow L_1AP_H|...|L_{n-1}AP_H$. For an incoming edge-mapping, IMS_D performs $Test(E(ID_{H_i}), T(ID_D))$ for each entry i. If $Test()$ returns 1 for an entry, i.e. a mapping for D already exists, the last locator in the sequence in the entry is replaced with the edge-locator sent from the DHCP server. If $Test()$ returns 0 for all entries, i.e. no entry for D exists so far, a new entry is created with the part of the locator sequence held already at IMS_D and the edge-locator sent from the DHCP server. Afterwards, IMS_D sends the entire locator sequence back to D's semi-masked local address $L_{n-1}AP_D \mid E(ID_D)$ (step MD_3). Finally, if a new mapping entry is created for D, IMS_D sends the mapping pointer $\{(E(ID_D), T(ID_D)) \Rightarrow Addr_{IMS_D}\}$ to CMS (step MD_4). Here, we want to point out that the DHCP server, IMS_D and CMS can communicate with each other by using their cleartext or semi-masked addresses.

3) Semi-masked Mapping Lookup: Before sending a semi-masked packet from the source endpoint S to the destination endpoint D, S has to resolve D's actual locator (see steps ML_1 to ML_4 in Figure 1). By means of PEKS, S first encrypts D's identifier ID_D with its public key. Next, S generates a request message for $E(ID_D)$ and sends it to IMS_S (step ML_1). S sets its own semi-masked address as the source address of the request message. For the incoming request message, IMS_S performs $Test(E(ID_D), T(ID_{H_i}))$ for each entry i. If $Test()$ returns 1 for an entry, i.e. S and D are connected to Edges in the same INT, IMS_S sends D's actual edge-locator to S. Otherwise, IMS_S forwards the request message to CMS (step ML_2). For the forwarded request message, CMS performs $Test(E(ID_D), T(ID_{H_i}))$ for each entry i and determines the IMS maintaining D's actual locator, namely IMS_D. Afterwards, the request message is forwarded to IMS_D (step ML_3). IMS_D finds the entry holding D's actual locator sequence LOC_D by performing $Test(E(ID_D), T(ID_{H_i}))$ for each entry i. Finally, IMS_D sends LOC_D back to S (step ML_4). S can cache the locator sequence so that it does not have to perform a mapping lookup for each packet.

4) Semi-masked Packet Delivery: The endpoint S aims to send a packet to the endpoint D. First, S tries to find D's masked identifier in its neighbour cache as described in Section III-A1. If D's masked identifier can be resolved to a MAC address, i.e. S and D are connected to the same Edge, the packet is directly sent to this MAC address. Otherwise, S resolves D's masked identifier to its actual locator as described in Section III-A3. The semi-masked packet delivery consists of three steps (see steps PD_1 to PD_3 in Figure 1). First, the packet is forwarded up to the Core either by using the default route or by utilising LOC_S in the semi-masked source address (step PD_1). Next, the packet is domain-wise forwarded so that L_kAP_D acts as an ingress node (and $L_{k+1}AP_D$ as an egress node) of the destination routing domain at level $k+1$ with $1 \leq k \leq n-2$ (step PD_2). Eventually, the packet arrives at $L_{n-1}AP_D$, which resolves D's masked identifier to its MAC address as described in Section III-A1. Finally, the packet is delivered to D's MAC address (step PD_3).

B. Fully Blind Packet Forwarding

The fully masked address of the endpoint D with the identifier ID_D is $fmAddr_D : mLOC_D \mid E(ID_D)$, where $mLOC_D$ is D's actual masked locator sequence consisting of $E(L_1AP_D)|...|E(L_{n-1}AP_D)$. Here, $E(L_kAP_D)$ is the masked locator of L_kAP_D encrypted with its public key by means of PEKS, where $1 \leq k \leq n-1$. The fully masked address fields in a packet from the endpoint S to the endpoint D consist of $fmAddr_D \mid fmAddr_S \mid C(ID_S)$. Here, $C(ID_S)$ serves the same purpose as described in Section III-A.

1) Masked Routing: For a masked routing table setup, we redesign the Distance Vector Routing algorithm to a

blind one as we introduced in [3]. The masked routing table entry for a network node with the locator N is $\{(E(N), T(N)), P_N, D_N\}$. Here, $E(N)$ is N's masked locator encrypted with N's public key, and $T(N)$ is N's trapdoor generated with N's private key. P_N is the number of the port via which N can be reached, and D_N is the distance to N. The masked routing is performed domain-wise so that a network node in a routing domain at level k maintains masked entries only for the network nodes in the same domain at level k, $1 \leq k \leq n-1$.

2) Fully masked Mapping Distribution and Lookup: The fully masked mode of the mapping distribution works just like the semi-masked mode in Section III-A2 (see steps MD_1 to MD_4 in Figure 1). However, a masked identifier is bound to a masked locator sequence in this mode. Therefore, the DHCP server maintains $E(L_{n-1}AP_D)$ and sends the fully masked edge-mapping $\{(E(ID_D), T(ID_D)) \Rightarrow E(L_{n-1}AP_D)\}$ to IMS_D in step MD_2. IMS_D keeps the remaining masked locator sequence $E(L_1AP_D)|...|E(L_{n-2}AP_D)$. It can be dynamically resolved by running an enhanced masked traceroute, e.g., for CMS. There, each L_kAP_D puts its masked locator into the message to be sent back to IMS_D.

A fully masked mapping entry for an endpoint H is $(E(ID_H), T(ID_H)) \Rightarrow mLOC_H$. In step MD_3, IMS_D sends the entire masked locator sequence back to D's fully masked local address $E(L_{n-1}AP_D) \mid E(ID_D)$. Finally, if a new mapping entry is created for D, IMS_D sends the mapping pointer $\{(E(ID_D), T(ID_D)) \Rightarrow fmAddr_{IMS_D}\}$ to CMS (step MD_4). The fully masked lookup is analogous to the semi-masked one except that the source endpoint S gets D's actual masked locator sequence $mLOC_D$. Additionally, S sets its own fully masked address as the source address of the request message.

3) Fully masked Packet Delivery: The fully masked mode of the packet delivery proceeds just like the semi-masked one except step PD_2. In this step, for an incoming packet with $mLOC_D = ...|E(L_kAP_D)|...$, a network node in the routing domain at level k performs $Test(E(L_kAP_D), T(N_i))$ for each masked routing table entry$i$, where $1 \leq k \leq n-1$. The packet is then forwarded via the port mapped in that entry for which $Test()$ returns 1.

IV. Analysis

The extended BPF adopts the architectural features of HAIR, which provides better scalability, fine-granular mobility and multihoming due to its hierarchical scheme. This scheme supports super- and subnetting by design and the same applies for our extended BPF. Furthermore, due to HAIR's hierarchical mapping system structure, the communicating endpoints need only the public key of each other, which they can obtain, e.g., via DNS. However, the extended BPF adopts the issues of HAIR too. Thus, the deeper the level, the longer are the cleartext/masked locator sequences of a packet in the extended BPF. Furthermore, attaching endpoints to nodes at an arbitrary level is not defined.

In the semi-blind mode of the extended BPF, the source and destination locator sequences in packets are transferred in cleartext. Thus, the entire route of a packet is known to any node on the route as well as to third parties, e.g., potential attackers. This means that nodes in intermediate domains and third parties can identify which domains communicate with each other. But there is still no possibility to match packets to communicating endpoints, since the source and destination identifiers are encrypted. Additionally, due to the mapping of locator sequences to masked identifiers, domain providers do not know which endpoints are connected to the Edges in their domains. Furthermore, the identifiers are encrypted in the CMS so that it is unknown which endpoints are actually located in which domains. However, the current number of endpoints located in a domain is known, since the domain location (locator sequence in the semi-masked address of the domain IMS) is maintained in cleartext by the CMS. Additionally, the domain location can eventually disclose information about the domain, e.g., its geographic or organisational position.

In the fully blind mode of the extended BPF, identifiers as well as locator sequences are encrypted. Thus, transferred packets cannot be matched to the communicating endpoints. Moreover, the routes of the packets are masked from nodes in intermediate domains and from third parties. Hence, they do not know which domains communicate with each other. The domain locations are encrypted in the CMS so that information about the domains is masked from the CMS. Furthermore, the location of a domain is masked from endpoints located in the domain too, since the endpoints get masked locator sequences from the IMS in the mapping distribution process. Moreover, the locator sequences of the communicating endpoints are masked from each other, since they get only the encrypted locator sequences of each other from the mapping system in the mapping lookup process. This means that an endpoint does not know in which domain its communication partner is actually located.

However, a domain provider can identify the route of a packet inside its own domain by running a costly process. Here, the provider monitors all ports of all nodes and compares the masked addresses of all incoming and outgoing packets byte by byte with each other. But the provider still cannot match the packet to the communicating endpoints, since the identifier is masked and separated from the masked locator sequence in the packet addresses. By means of this costly monitoring process the provider can identify the entire route of a packet only in case of intra-domain communication. Otherwise, the provider can only identify a part of the entire route of a packet, namely the part that the packet takes inside its own domain. But the provider still does not know which domains communicate with each other. In case of intra-domain communication, full masking makes sense if the route of a packet has to be masked from third parties, e.g., potential attackers. But if the domain provider guarantees that no node in the domain can be attacked and the endpoints trust the provider, the packet addresses can be semi-masked.

V. Implementation and Evaluation

Software-Defined Networking (SDN) eases the development of clean-slate designs such as ours. OpenFlow [8] is a SDN protocol providing a variant of flow-based packet forwarding, which supplies flexibility, dynamics and high performance. In OpenFlow, an external controller defines flow entries containing matching fields and instructions, which are performed on matching packets. However, the flow match field types implemented currently in an OpenFlow switch rely on the IP packet structure. Thus, by developing a clean-slate design in Open-

Flow, we have either to extend the implementation accordingly to the design by means of the so called "*Experimenter Flow Match Field*", or packets have to be handled by the controller hop-by-hop, which causes certain overhead. Nonetheless, we can reinterpret the types supported already in an OpenFlow switch so that the controller implementing a clean-slate design still can define flows for its own purposes. Thus, we still utilise the benefits of flow-based packet forwarding. In our prototype implementation of the extended BPF, we reinterpret the field type for *VLAN-ID* to match packets. At each node in a domain at level k, we define flows for each node N_i maintained in its routing table. In the flow for N_i, the controller uses the last byte of N_i's masked locator as VLAN-ID in the match field and sets the action "*Output*" with the port number via which to route to N_i. Furthermore, we define a flow with a default VLAN-ID value for which matching packets are passed to the controller. An endpoint sending a packet sets the VLAN-ID to this default value. At the node to which the endpoint is attached in the Edge the packet is sent to the controller which runs $Test()$. Next, the VLAN-ID in the packet is updated and the packet is sent back to the node. From then on, the packet is forwarded using the flows defined at the nodes on the route. For each domain crossing on the route, the packet has to be sent to the controller in order to update the VLAN-ID in the packet.

We have implemented our design on an OpenFlow 1.3 compatible user-space software switch [12] in an emulated OpenFlow network using Mininet [13]. We have extended the controller NOX [9] by creating the components *hop-by-hop-HAIR*, *flow-based-HAIR*, *hop-by-hop-semi-BPF*, *flow-based-semi-BPF*, *hop-by-hop-fully-BPF* and *flow-based-fully-BPF*. Each node, endpoint, mapping server and DHCP server is emulated by an OpenFlow switch so that they all can be managed by our controller components. In our implementation, a cleartext identifier/locator consists of 16 bytes, while a masked identifier/locator has 193 bytes and a trapdoor for a locator has 65 bytes. We have evaluated the performance of our controller components in a ping-pong scenario in a basic topology set up with Mininet. The topology consists of five domains (the Core and four INTs) at four levels, where each domain has ten nodes (see Figure 1, $n = 4$). Thus, each node in each domain maintains ten routing table entries. In our scenario, S pings D five times and D sends for each ping packet a pong packet back to S. In this topology, a packet from S to D, or vice versa, takes 46 hops. Table I shows the packet round trip times and the execution times for mapping distribution and lookup in the controller components which are implemented in the emulated OpenFlow network on an Ubuntu virtual machine with an Intel Core2 Duo 2.53 GHz CPU and 512 MB RAM.

The round trip for the first packet takes more time than for the remaining packets since S has first to resolve D's locator sequence. S caches the locator sequence so that it does not need to perform a mapping lookup for the remaining packets. Because only the identifiers are masked, the semi-blind mode is more performant than the fully blind one. Moreover, this basic experiment demonstrates the benefits of the flow-based packet forwarding compared to hop-by-hop packet handling by the controller. Here, we especially want to point out that flow-based semi-BPF is almost as performant as hop-by-hop-HAIR. Due to the size and execution time overheads, fully BPF cannot satisfactorily be put into practice for environments such as the Internet nowadays, at least not by default. Nonetheless, due to the security level provided by fully BPF, it could be offered as a premium service on demand in case of sensitive transmissions. In contrast, however, flow-based semi-BPF, which provides acceptable end-to-end security, seems to be promising to be realised in practice.

TABLE I. PACKET ROUND TRIP TIMES AND EXECUTION TIMES FOR MAPPING DISTRIBUTION AND LOOKUP

	average RTT in ms		**mapping lookup time in ms**	**mapping distribution time in ms**
	first packet	*remaining packets*		
hop-by-hop-HAIR	76.37	42.86	37.98	17.55
flow-based-HAIR	25.32	7.48	20.39	11.57
hop-by-hop-semi-BPF	166.31	69.51	87.73	54.07
flow-based-semi-BPF	99.05	29.98	53.58	44.78
hop-by-hop-fully-BPF	4997	2482	2364	493.32
flow-based-fully-BPF	906.68	377.44	473.78	165.08

VI. CONCLUSION

This paper discussed the issues of the basic BPF and resolved them by applying the Loc/ID principle, which supports scalability, mobility and multihoming by design. We introduced the extended BPF, where we adopted the architectural features of HAIR. We presented constructions for a semi- and fully blind mode of the extended BPF. After that, we pointed out important features of the extended BPF and discussed the security levels provided in the semi- and fully blind mode. In the prototype implementation of the extended BPF, we leveraged the flow-based packet forwarding using OpenFlow, and evaluated it in an emulated network using Mininet.

REFERENCES

[1] C. Henke, A. Siddiqui, and R. Khondoker, "Network functional composition: State of the art," in *Telecommunication Networks and Applications Conference (ATNAC), 2010 Australasian*, 2010, pp. 43–48.

[2] D. Boneh, G. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Advances in Cryptology - EUROCRYPT 2004*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, vol. 3027, pp. 506–522.

[3] I. Simsek, M. Becke, Y. I. Jershow, and E. P. Rathgeb, "A clean-slate security vision for future networks," in *Proceedings of the IEEE International Conference on Network of the Future (NoF'13)*, Pohang, South Korea, Oct. 2013.

[4] D. Meyer, L. Zhang, and K. Fall, "Report from the iab workshop on routing and addressing," RFC 4984, Sep. 2007.

[5] T. Li, L. Zhang, and K. Fall, "Recommendation for a routing architecture," RFC 6115, Feb. 2011.

[6] *ITU-T SG 13 Y.3031*, Identification framework in future networks, May 2012. [Online]. Available: http://handle.itu.int/11.1002/1000/11564

[7] A. Feldmann, L. Cittadini, W. Mühlbauer, R. Bush, and O. Maennel, "Hair: Hierarchical architecture for internet routing," in *Proceedings of the 2009 Workshop on Re-architecting the Internet*, ser. ReArch '09. Rome, Italy: ACM, 2009, pp. 43–48.

[8] Openflow. [Online]. Available: https://www.opennetworking.org/sdn-resources/onf-specifications/openflow

[9] Nox. [Online]. Available: http://www.noxrepo.org/

[10] S.-T. Hsu, C.-C. Yang, and M.-S. Hwang, "A study of public key encryption with keyword search," *International Journal of Network Security*, vol. 15, no. 2, pp. 71–79, 2013.

[11] M. Hoefling, M. Menth, and M. Hartmann, "A survey of mapping systems for locator/identifier split internet routing," *Communications Surveys Tutorials, IEEE*, vol. 15, no. 4, pp. 1842–1858, 2013.

[12] Openflow 1.3 software switch. [Online]. Available: https://github.com/CPqD/ofsoftswitch13

[13] Mininet. [Online]. Available: http://mininet.github.com/