Naming, Assigning and Registering Identifiers in a Locator/Identifier-Split Internet Architecture

Christoph Spleiß, Gerald Kunzmann[1]
Technische Universität München
Department of Communication Engineering
Munich, Germany
{*christoph.spleiss, gerald.kunzmann*}*@tum.de*

***Abstract*—Splitting the IP-address into locator and identifier seems to be a promising approach for a Future Internet Architecture. Although this solution addresses the most critical issues of today's architecture, new challenges arise through the necessary mapping system to resolve identifiers into the corresponding locators. In this work, we give an overview of a scheme how to name identifiers not only for hosts, but basically for anything that needs to be identified in a future Internet. Our approach is based on the HiiMap locator/ID split Internet architecture and supports user-friendly identifiers for hosts, content, and persons and does not rely on DNS. We show how the registration and assignment for identifiers is handled and which modifications in the network stack are necessary. Furthermore, a possible solution for a lookup mechanism that can deal with spelling mistakes and typing errors in order to improve the quality of experience to the user is provided.**

***Keywords*-Locator/ID-split, Future Internet, Naming schemes, Content Addressing.**

I. Introduction

Today's Internet architecture has been developed over 40 years ago and its only purpose was to interconnect a few single nodes. At that time, no one expected that the Internet and the number of connected devices would grow to the current size. Measurements show that the Internet continues growing at a tremendous high rate. The address space of the current IPv4 addresses is already too small to address every single node in the Internet and the growth of BGP routing tables sizes in the Default Free Zone (DFZ) becomes critical for the Internet's scalability [1]–[4]. Beyond that, mechanisms like traffic engineering (TE) or multi homing are further increasing the BGP tables sizes, as IP address blocks are artificially de-aggregated or advertised multiple times. While IPv6 is a promising solution for the shortage of addresses, it will probably increase the Border Gateway Protocol (BGP) routing table problem. Therefore, at least for IPv6, a new and scalable routing architecture is necessary. Besides that, more and more devices connected to the Internet are mobile, such as smart phones or netbooks. Even more and more cars, which are mobile by definition, have access to the Internet. However, the current Internet architecture has only very weak support for mobility, as the IP-address changes whenever a device roams between different access points.

All problems mentioned above occur because the IP-address, no matter if IPv4 or IPv6, is semantically overloaded with two completely independent values. It is used to *identify* a specific host in the Internet, while on the other hand, it is also used to *locate* this node. Separating the current IP address into two independent parts for reachability and identification is a promising solution to many problematic issues with today's Internet [5]. With this approach a known identifier (ID) can always be used to reach a specific host, no matter where it is currently attached to the network. Thereby, a logical communication session is always bounded to the ID and not to the locator. A locator change due to mobility is handled by the network stack and is completely transparent to the communication session and the overlying application. It does not disconnect an ongoing communication anymore. Furthermore, it is possible to assign more locators to a specific ID, e.g., if a host is multihomed.

However, not only the number of hosts has developed differently than initially expected, but also the way people use the Internet. While the current Internet architecture is designed for accessing a specific machine, todays focus has shifted on accessing a specific piece of information. The host storing the information is thereby of minor interest. The idea is a new network architecture named Content Centric Network (CCN) [6]–[8] where content and information can be directly addressed independent of its storage location, which usually points to a host. Furthermore, the emergence of social networks, Web 2.0 applications, Voice over IP (VoIP) and instant messaging applications additionally put the person in the focus of interest. People want to communicate with each other, regardless of the device each of the person is using.

The split of locator and ID thereby offers ideal prerequisites for the support of addressing schemes for content, information and persons. Using this paradigm, an ID is assigned for every host, content object and person. A highly scalable mapping system, which is a mandatory part of every

[1]Dr. Kunzmann is now working for DOCOMO Communications Laboratories Europe GmbH, Landsberger Strasse 312, Munich, Germany.

locator/ID separated Internet architecture, translates IDs into the corresponding locators. Note that the mapping system can not only return a set of locators, which are necessary to access a specific host, but it can also return a more complex description of a content object or a person.

A crucial question in conjunction with a locator/ID separated Internet architecture is how to name, assign and register IDs. As IDs are used as control information in communication protocols and packet headers, they are mostly fixed-length bit strings that can be hardly memorized by humans. However, in order to avoid a second resolution system like DNS that translates easy memorizable names to IDs, we need a way how to uniformly name IDs. In this work, we present a flexible and adaptable naming scheme for IDs that can be used to identify hosts, content, persons and is open for future extensions. We furthermore present techniques how to register IDs and how assign them accordingly. Although our approach can be adapted to basically any locator/ID separated Internet architecture, it is currently based on the HiiMap Internet architecture [9], as HiiMap provides a highly scalable and customizable mapping system. It does not rely on the Domain Name System and allows each entity to calculate the requested ID on its own.

The paper is structured as follows. In Section 2, we discuss related work and different concepts of locator/ID split architectures. Section 3 describes our approach of a new naming scheme for IDs while Section 4 deals with registration and assignment issues. Section 5 demonstrates necessary modifications in the network stack and Section 6 discusses a possible lookup algorithm that tolerates spelling mistakes and allows unsharp queries to a certain extent. Section 7 summarizes the results and concludes.

II. Related Work

Many proposals dealing with the split of locator and ID have been published so far, but only a few of them discuss how to name IDs. However, almost all of them use a bit-representation of constant length as ID.

A. Host-based approaches

The majority of these proposals are solely host-based approaches. Among them Hair [11], FIRMS [13], LISP [14], HIP [15], HIMALIS [16] and many others. In the following, we will describe three of them more detailed, as their way how to implement the locator/ID-split is representative.

LISP: In contrast to other architectures that are examined in this work, LISP does not separate the identifier from routing purposes. Within an edge network or autonomous system (AS), the normal IP-address still serves as so called *Endpoint Identifier (EID)* and routing address at the same time. While the EID is only routable inside a LISP-domain, an additional set of addresses is used for the routing between different LISP-domains, which are called *Routing Locators (RLOC)*. RLOCs are the public IP-addresses of the border routers of a LISP-domain, globally routable, and independent of the nodes' IP addresses inside the domain. Whenever a packet is sent between different LISP-domains, the packet is first routed to the *Ingress Tunnel Router (ITR)*, encapsulated in a new IP packet, and routed to the *Egress Tunnel Router (ETR)* according to the RLOCs. The ETR unpacks the original packet and forwards it to the final destination. A mapping system is necessary to resolve foreign EIDs (EIDs that are not in the same domain) to the corresponding RLOCs. However, as in normal IP networks, the EID changes whenever a node changes its access to the network. Furthermore, DNS is still necessary to resolve human readable hostnames to EIDs.

HIP: The Host Identity Protocol implements the locator/ID split by introducing an additional layer between the network and the transport layer. For applications from higher layers the IP address is replaced by the *Host Identity Tag (HIT)*, which serves as identifier. HIP leaves the IP-layer untouched and the locator is a simple IPv4 or IPv6 address. Therefore, HIP does not rely on special gateways, which is very migration-friendly. However, as it does not influence the routing system, it does not take any countermeasures against the growth of the BGP routing tables. Instead, the main focuses of HIP are security features. The *Host Identifier (HI)* is a public key of an asymmetric key pair and used for identification and cryptographic purposes at the same time. The HI is not used in packet headers, as the key length can vary over time. Instead, the HIT, which is a hash value from the HI, is used for addressing purposes. Encryption, authenticity and integrity can be achieved due to the presence of an asymmetric key pair. However, the coupling of identifier and public key is a major drawback, as the ID changes whenever the key pair changes. While no permanent validity may be a desirable feature for cryptographic key pairs, it is not for an identifier.

HIMALIS: Like HIP, the HIMALIS (Heterogeneity Inclusion and Mobility Adaption through Locator ID Separation in New Generation Network) approach realizes the locator/ID split by introducing an extra layer between network and transport layer, the so-called Identity Sublayer. Upper layers solely use the host IDs for session identification, while the lower layer use the locator for packet forwarding and routing. HIMALIS can use any kind of addressing scheme for locators and supports security features based on asymmetric keys. Despite that, it does not burden the ID with the semantic of a public key. HIMALIS uses domain names as well as host IDs to identify hosts. In contrast to other approaches, a scheme how to generate host IDs out of the domain name using a hash function is shown. However, HIMALIS uses three different databases for resolving domain names and hostnames to IDs and locators (Domain Name Registry, Host Name Registry and ID Registry), which results in increased maintenance effort to achieve data consistency.

B. *Content-based approaches*

Contrary to host based approaches, several proposals for a future Internet architecture take into account the changing demands regarding the intended use of the Internet, namely the focus on retrieving information [6].

The **NetInf** (Network of Information) architecture shows how locator/ID separation can be used for content-centric networking [8]. By introducing an information model for any kind of content, NetInf allows fast retrieval of information in the desired representation. Thereby, each information object (IO) includes a detailed description of the content and its representations, with locators pointing to the machine that stores the information. The ID is assigned to the IO and is composed out of different hash values of the content creator's public key and a label created by the owner. In order to find a specific IO, the creator's public key and label must be known exactly. A first approach of NetInf is realized as an overlay architecture that uses the current routing and forwarding system, while a second alternative plans to deal with name-based routing mechanisms that provide an integrated name resolution and routing architecture.

Another Future Internet Architecture focusing on content is **TRIAD** [7]. One key aspect of TRIAD is the explicit introduction of a content layer that supports content routing, caching and transformation. It uses character strings of variable length as content IDs and uses the packet address solely as locator. The TRIAD content layer consists of *content routers* that redirect requests to *content servers* and *content caches* that actually store the content. DNS is used to resolve locators for content objects and the content delivery is purely based on IP in order to achieve maximum compatibility with the current Internet architecture.

C. *Hybrid approaches*

A proposal for a Next Generation Internet architecture that supports basically any kind of addressing scheme is the **HiiMap** architecture [9]. Due to the locator/ID separation and a highly flexible mapping system, HiiMap allows for addressing hosts as well as content and is still open for future extensions and requirements. In the following, we use the term *entity* for any addressable item.

The HiiMap architecture uses never changing IDs, so called UIDs (unique ID) and two-tier locators. One part of the locator is the LTA (local temporary address) that is assigned by a provider and routable inside the provider's own network. The other part is the gUID (gateway UID). This is a global routable address of the provider's border gateway router and specifies an entrance point into the network. Thereby, each provider can choose its own local addressing scheme that can be adapted to specific needs. However, a common addressing scheme for all providers is necessary for the gUID in order to route packets [17].

HiiMap splits the mapping system into different regions, whereby each region is its own independent mapping system that is responsible for the UID/locator mappings of entities registered in this region. The mapping system in each region consists of a one-hop distributed hash table (DHT) to reduce lookup times. As DHTs can be easily extended by adding more hosts, the mapping system is highly scalable. In order to query for UIDs which regions are not known, a region prefix (RP) to any UID is introduced (compare Figure 1). This RP can be queried at the so-called Global Authority (GA), which resolves UIDs to RPs. The GA is a centralized instance and acts as root of a public key infrastructure, thus providing a complete security infrastructure. As RP-changes are expected to be rare, they can be cached locally.

Like other approaches, HiiMap uses fixed length bit strings of 128 bits as UID. As plaintext strings are not feasible as UIDs due to their variable length, a naming scheme is necessary to assign UIDs to all kinds of entities. Thereby, the existing Domain Name System is to be replaced by the more flexible HiiMap mapping system.

III. New naming scheme for Identifiers

In this section, we introduce a naming scheme for IDs that is suitable to address basically any entity and that can be generated out of human friendly information. Although we use the HiiMap architecture exemplarily for introducing this approach, it can also be adapted to other locator/ID split architectures.

A. *General Requirements for Identifiers*

When introducing a Future Internet Architecture based on locator/ID separation, the ID has to fulfill some mandatory requisites. In the following, we sum up general requirements for IDs proposed by the ITU [18]:

- The ID's namespace must completely decouple the network layer from the higher layers.
- The ID uniquely identifies the endpoint of a communication session from anything above the transport layer.
- The ID can be associated with more than one locator and must not change whenever any locator changes.
- A communication session is linked to the ID and must not disconnect when the locator changes.

In addition to the ITU we add further requirements:

- An ID must be able to address any kind of entity, not only physical hosts.
- Every communication peer can generate the ID of its communication partner out of a human readable and memorable string.
- The ID is globally unique, but it must be possible to issue temporary IDs.
- The registration process for new IDs must be easy.
- IDs must be suitable for DHT storage.

While some of these aspects mainly affect the design of a Future Internet Architecture based on a locator/ID split, some issues are directly related with the naming of IDs.

T	Type	Input to Hash(name)	Ext 1	Ext 2
1	static host	plain text domain name	hash of local hostname	service
2	non-static host	global prefix assigned by provider	hash of local hostname	service
3, 4	content	plain text content name	child content	version number
5	person	first + last name	random	communication channel

Table I: Generic contents of UID fields corresponding to different types

B. Generalized Identifier

As IDs are used in the transport layer protocol to determine the endpoint of a communication, we cannot avoid using fixed-length bit strings to realize packet headers of constant size. In combination with DHTs, which also require fixed-length bit strings, the usage of a hashing function is obvious. In contrast to other approaches, which compose the ID of one hash value only, we split the ID in several predetermined fields whose purposes are known to all entities.

In the following, we introduce a generalized scheme how to compose global unique IDs (UID) for any entity and give concrete examples how to name hosts, content and persons. Our scheme allows storing all these IDs in the same mapping database and is yet flexible enough to support different databases for different types of IDs.

Figure 1 shows the generalized structure of an ID, which is composed of a region prefix (RP) according to HiiMap and an UID. The UID consists of a type field (T), the hash value of a human friendly name for the entity to be identified as well as two extension fields (Ext 1 and Ext 2). The UID is stored in the mapping system of a specific region, denoted by the RP.

The type field T denotes to which type of entity the UID belongs to. T allocates the most significant bits (MSB) in the UID, which allows to map different ID types to different databases in the mapping system. As some entities require more complex entries in the mapping system, it may be desirable to use different databases that are optimized for the needs of a specific entity. We suggest using 128 bits for the UID, whereby 4 bits are used to determine the type, 76 bits are assigned for the hash value, 32 bits for Ext 1 and 16 bits for Ext 2. In the following, we show realizations for applying UIDs to different types: host, content and persons. Table I gives an overview how the UID is composed according to the type of entity. Each part is described in detail in the following subsections. Note that our scheme is not limited to these types, but can easily be extended.

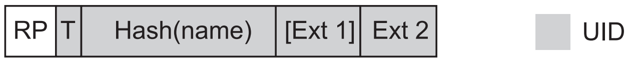

Figure 1: Identifier UID with regional prefix RP

C. Identifiers for Hosts

IDs for hosts are the most common use case today and DNS is used to resolve hostnames to IP addresses in order to access a specific machine. The hostname, or FQDN (full qualified domain name), which specifies the exact position in the tree hierarchy of the DNS, can be roughly compared to the ID in a locator/ID separated Internet architecture. However, the FQDN is solely used in the application layer and is not present in any lower layer network protocol.

Similar to today's hostnames, we introduce a hierarchy to our UIDs. However, contrary to FQDNs, our scheme is limited to two hierarchical levels: a global part and a local part. While the global part is used to identify a whole domain, e.g. a company or an institute at a university, the local part is used to identify single machines within this domain. Note that the term *domain* does not refer to a domain like in today's DNS hierarchy. A domain in our solution has a flat hierarchy and simply defines an authority for one or more hosts. We differentiate between two different types of host UIDs and give an overview in Table II:

1) Static Host Identifier: Static host UIDs are never changing IDs of type T=1 that can be generated by hashing a human readable hostname. Their main purpose is for companies or private persons that want to have a registered UID that is always assigned to their host or hosts.

Hash: The domain name part of the plain text hostname is used to generate the hash field of the UID. An example can be *mycompany.org* as domain name.

Ext 1: The hash value of the local host name is used to generate this field. A local hostname is unique inside a specific domain. An example for a local hostname could be *pc-work-003*. That way, together with the domain name, a host in the Internet is unambiguously identified.

Ext 2: The Ext 2 field is used to identify a specific service on the host. It can be compared to today's TCP or UDP ports. An application or service listening for connections must register its Ext 2 value at the network stack in order to receive data. Some values for Ext 2 are predetermined, e.g., for file transfer or remote administration, others can be chosen freely. However, specifying a value in Ext 2 is not necessary when requesting the locator for a specific host from the mapping system and can therefore be set to zero. As the host is precisely identified with the global and local UID part, it is not necessary to store identifiers for each service of

T	Type	Input to Hash(name)	Ext 1	Ext 2	
1	static host	plain text domain name, e.g., *mycompany.org*	hash of local hostname, e.g., *pc-work-003*	0	used for requesting the locator from the mapping system
				$1..n-1$	standardized Ext 2 values for $n-1$ common applications and services
				$n..2^{16}$	further values for proprietary or open use
2	non-static host	global prefix assigned by provider, e.g., *provider12345*	as above	as above	

Table II: Contents of UID fields for hosts

the host as they would all point to the same locator. Instead, Ext 2 is set to zero in the UID when querying the mapping system and filled with the specific service identifier when actually accessing the node.

For privacy reasons it is possible not to publish the UID for a private host in the global mapping system but only in a local database. For a single point of contact it is possible to use an UID with Ext 1 set to zero, which points to, e.g., a login server, router or load balancer that forwards incoming requests to designated internal hosts.

Note that the host has to update its mapping entry pointing to new locator(s) upon any locator change.

2) Non-static Host Identifier: Contrary to static host IDs and the basic idea of never changing UIDs there will always be the need for non-static host UIDs, i.e., IDs that do not have to be registered, that are assigned to a host for a specific time, and that are returned to the issuer if no longer needed. We assign the type value T=2 to these class of UIDs. An example can be a private household with a DSL, cable or dial-up Internet connection and a few hosts connected through a router. Each host needs its own, distinct UID to make connections with other hosts in the Internet. However, it does not need to have a registered, never changing UID if no permanent accessibility is needed.

Hash: The global part is assigned during the login process to the router or middlebox that provides Internet access to the other hosts. It can be compared to the advertisement of an IPv6 prefix. The global part, e.g., the hash of *provider12345*, is valid as long as the customer has a contract with its provider. A new global part is assigned if the customer changes its provider. Yet, the transfer of a global UID part between different providers should be possible. In order to assign non-static UIDs to customers, each provider holds a pool of global UID parts. The mapping entry for a specific non-static host UID is generated by the corresponding host immediately after assignment and whenever its locator changes. However, each host with no static UID assigned must proactively request a non-static host UID, either by its provider or router and middlebox, respectively. Note that the global part of a non-static host UID does not necessarily consist out of the hash value of a plaintext string. It depends on the provider if he uses hash values or just consecutive numbers out of a contiguous pool.

Ext 1: Like with static UIDs, the local part of a non-static UID is generated from the local hostname of a machine. Therefore, a name for each host is mandatory.

Ext 2: Identical to the Ext 2 field used for static host UIDs.

D. Identifiers for Content

As the focus of the users in the Internet is shifting from accessing specific nodes to accessing information and content, different approaches towards a content-centric network have been made as shown in Chapter II. By applying the idea of information models, like the NetInf approach, to our naming scheme, each content, which can be, e.g., a webpage or an audio or video file, gets its own distinct UID. Hereby, the UID does not point to the data object itself but to the information model of the content that has a further description and metadata stored. We use two different types for content UIDs: T=3 and T=4, whereas T=3 is used for content that is not subjected to public change, and T=4 which is used for content that is free to public changes.

Hash: For generating the UID of a content we have to use a meaningful name that can describe the corresponding content or information. While this is indeed quite a difficult task, possible solutions can be, e.g., the name of a well-known newspaper like *nytimes*, which refers to the front page of the New York Times online version. Similar, the name of an artist could refer to an information object where albums or movies are linked.

In our proposal, this plaintext name is used as input to a known hash function to generate the hash part of the UID. As the spelling of the content description is not always exactly known, we suggest a lookup mechanism that can cope with minor spelling mistakes in Section VI.

Ext 1: This field is optional and can be used to access some more specific parts of the content or information that

T	Type	Input to Hash(name)	Ext 1 (optional)	Ext 2 (optional)	(Pointer to)
3, 4	content	meaningful content name or description, e.g., *nytimes*	**Child content or information:** • hash of child content name, e.g., *sports* • number/ID: e.g. *DOI* • value 0: request toplevel description of content object, i.e., list of possible values with short description	**Version of content:** 0 (when querying mapping)	mapping entry of content object
				0 (when requesting content)	current or actual version
				$1..2^{16}$ (when requesting content)	some older versions

Table III: Contents of UID fields for content

is directly related with the main object. This can be, e.g., one specific article from a newspaper site or a sub-page that deals with a specific topic like sports or politics. Other examples are specific albums or pieces of music from an artist. Ext 1 can help to avoid downloading a maybe bigger object description of the main content to gather the desired information. Another benefit is that each child object has its own locator and therefore can be stored on different locations while still being accessible through its parent UID. This is not possible today as, e.g., the URL of a newspaper article is directly coupled with the host storing the information. Ext 1 is created by hashing the human readable name for the detailed description or just by using consecutive numbers. The latter is for example useful to assign a specific article of a newspaper. It can be compared to the Digital Object Identifier (DOI) in digital libraries [19]. In order to access a specific content object which Ext 1 value is not known, the user can access the top-level description of this content object by setting Ext 1 to zero. This description then includes lists with all possible values for Ext 1 with a short description.

Ext 2: This field can be used to access a specific version of the desired content or information. Like in a versioning system, the Ext 2 field allows the user to easily access any earlier version and the changes made to the information. The content description is obtained by setting Ext 2 to zero where different values for Ext 2 are listed. Of course, each new version needs an update of the mapping entry including potential new locators. Note that we do not create own mapping entries for each Ext 2 value. Instead, available values for Ext 2 are listed in the mapping entry that is identifier by the hash part together with Ext 1. Like with host addressing, Ext 2 is always set to zero when querying the mapping system and filled with the desired value when actually requesting the content.

Table III summarizes the purposes of the UID fields for content objects. Unlike with host addressing, we cannot simply connect to a locator returned by the mapping system. As the information object is a description of content or information, the requesting application or user has to evaluate the information object and select the desired representation according to the user's needs. Thus, the network stack will not evaluate the data received from the mapping system for a content UID query but forward it to the corresponding application. The detailed interaction of applications requesting content UIDs and the network stack will be discussed in Chapter V.

Note that in case a name, e.g., *nytimes*, refers to both a host (company) as well as content (webpage), the type field is used to differentiate whether a locator (for type host) or a content description (for type content) is returned.

E. Identifiers for Persons

With the emergence of social networks, Internet-capable devices, Voice-over-IP (VoIP), etc., the need for personal IDs arose, as the *person* itself is moving in the focus of interest. Whenever somebody wants to contact a specific person he is interested in the communication with that person and does not want to care about the device, e.g., which phone or computer the person is currently using for communication. However, the user must have the possibility to choose the desired communications channel. That can be an email, a phone call, a message on a mailbox, a chat with an instant messenger or a message in a social network and so on. Furthermore, the personal UID can be used to make digital signatures which is necessary for contracts whatsoever. It has the assigned type T=5.

Hash: The main part of a person's UID consists of a hash value calculated from the person's full name, i.e., first name plus last name. As many people have the same first and last name, the hash value is ambiguous and we need further information to distinguish between different persons.

Ext 1: For this purpose we use a pseudo-random number for Ext 1 when initially generating a person's UID [20]. This initial generation is not done by the person itself in order to avoid multiple usage of the same Ext 1 value, but is issued by a federal authority and valid for lifetime . It can be compared to the number of a passport or the social security number, which do not change over the persons lifetime.

Ext 2: This field is used to specify the communication channel to the corresponding person and has a set of predetermined values, e.g., for email, VoIP, or instant messaging.

T	Type	Input to Hash(name)	Ext 1 (optional)	Ext 2 (optional)		(Pointer to)
5	person	first + last name	(random) number issued by federal authority	0	request personal profile of person	mapping entry of personal profile
				1	request locator of machine	machine user is working on
				$2..n$	predetermined standart values	mail server, VoIP phone or VoIP provider, chat server, etc.
				$n..2^{16}$	open for future extensions	
			0	0		access directory

Table IV: Contents of UID fields for persons

Note, there are still enough unused values for future needs. According to each Ext 2 value, different locators can be stored in the mapping system. For example, the Ext 2 value referring to the VoIP account can point to the locator of a VoIP provider or directly to a VoIP phone, the value referring to the mailbox can point to a mail server. The mapping entry for Ext 2 set to zero includes the person's full name and, depending on the person's privacy settings, further details about the person like birth date or current residential address. We refer to this as a*personal profile* according to [21]. Ext 2 set to one is used to get the locator of the machine the person is currently working on if the corresponding person agreed to publish this information. Thereby, the communication channel can be signaled in a higher layer or by using the machines service identifier Ext 2 when contacting the corresponding host. Note that the exposure of the device's locator the person is currently using can be abused to generate a movement profile. We suggest to use a privacy service like showed in [22] to avoid these issues.

However, to contact a specific person, not only the person's name but also Ext 1 must be known. There are two possibilities: First, the initiating person knows the correct UID of its communication partner because they have exchanged it in any way, e.g., with business cards. Second, the holder of a personal UID can agree to be indexed in a directory that is accessible through a personal UID with Ext 1 and Ext 2 set to zero. It stores the personal profiles of all persons that have the same name including their random Ext 1 values plus additional information to distinguish persons. According to the persons privacy settings, further details about city or street can be published, like in a traditional printed or online phone book. Table IV summarizes the necessary fields for personal UIDs.

In order to avoid abuse of personal UIDs, the usage of PKI mechanisms that guarantee the presence of the corresponding private key is mandatory for every transaction.

IV. Identifier registration and assignment

As each UID is globally unique by definition, it must be ensured that only one entity at a time has a specific UID assigned. It must be further prevented that any entity is hijacking an UID for malicious purposes.

A. Static host UIDs

The registration process for a static host UID can be compared to today's domain names and is depicted in Figure 2. Whenever a user wants to register a new static host UID for his hosts, he has to register a UID for each host together with a public key of this host at his local registry or local NIC (Network Information Center) with a `REGISTER_UID`. If the UID is still available, the NIC creates the initial mapping entry in the mapping system with `CREATE_UID` and stores the hosts public key. Updates are only allowed if the `UPDATE_MAPPING` message is signed with the correct private key, thus avoiding the UID to be hijacked.

Whenever the host connects to the Internet, it first requests a valid locator at the designated router with `REQ_LOCATOR`. Together with the locator assignment, the locator for the mapping system is delivered (`ASSIGN_LOCATOR`) and the host can update its mapping entry. Whenever the host's access point changes, it will request a new locator and immediately update the mapping entry with the new valid locator.

The UID at the node is configured via a system file like */etc/hostname* and */etc/domainname*. The owner of a host must proclaim changing the key-pair of a node at the mapping region. Note that for static UIDs *every* new UID must be registered at the registry. It is not possible to register only the domain part and to freely choose values for Ext 1, as each static UID needs its own key pair.

B. Non-static host UIDs

The purpose of non-static UIDs is that they do not need a registration process, as their prefixes are dynamically assigned by a provider and therefore belong to that provider. However, it must be possible for hosts with non-static UIDs

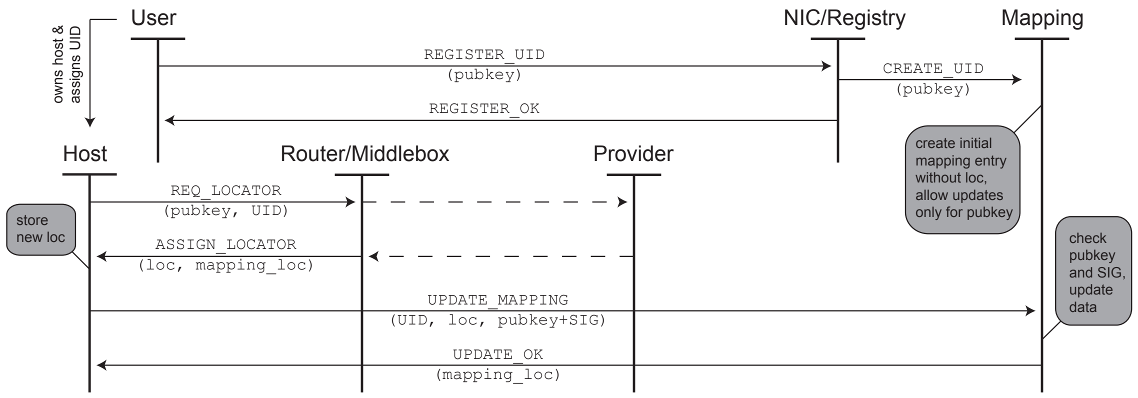
owns host & assigns UID
User
NIC/Registry
Mapping
REGISTER_UID (pubkey)
CREATE_UID (pubkey)
REGISTER_OK
create initial mapping entry without loc, allow updates only for pubkey
Host
Router/Middlebox
Provider
REQ_LOCATOR (pubkey, UID)
store new loc
ASSIGN_LOCATOR (loc, mapping_loc)
check pubkey and SIG, update data
UPDATE_MAPPING (UID, loc, pubkey+SIG)
UPDATE_OK (mapping_loc)

Figure 2: Signal flow chart for static UID registration and assignment

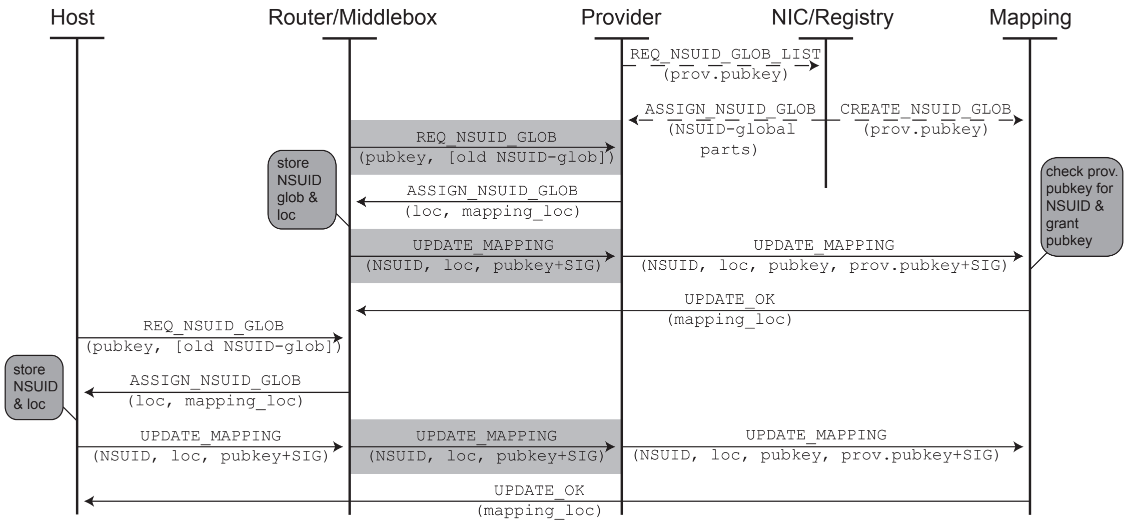
Host
Router/Middlebox
Provider
NIC/Registry
Mapping
REQ_NSUID_GLOB_LIST (prov.pubkey)
ASSIGN_NSUID_GLOB (NSUID-global parts)
CREATE_NSUID_GLOB (prov.pubkey)
REQ_NSUID_GLOB (pubkey, [old NSUID-glob])
store NSUID glob & loc
ASSIGN_NSUID_GLOB (loc, mapping_loc)
check prov. pubkey for NSUID & grant pubkey
UPDATE_MAPPING (NSUID, loc, pubkey+SIG)
UPDATE_MAPPING (NSUID, loc, pubkey, prov.pubkey+SIG)
UPDATE_OK (mapping_loc)
REQ_NSUID_GLOB (pubkey, [old NSUID-glob])
store NSUID & loc
ASSIGN_NSUID_GLOB (loc, mapping_loc)
UPDATE_MAPPING (NSUID, loc, pubkey+SIG)
UPDATE_MAPPING (NSUID, loc, pubkey+SIG)
UPDATE_MAPPING (NSUID, loc, pubkey, prov.pubkey+SIG)
UPDATE_OK (mapping_loc)

Figure 3: Signal flow chart for non-static UID registration and assignment

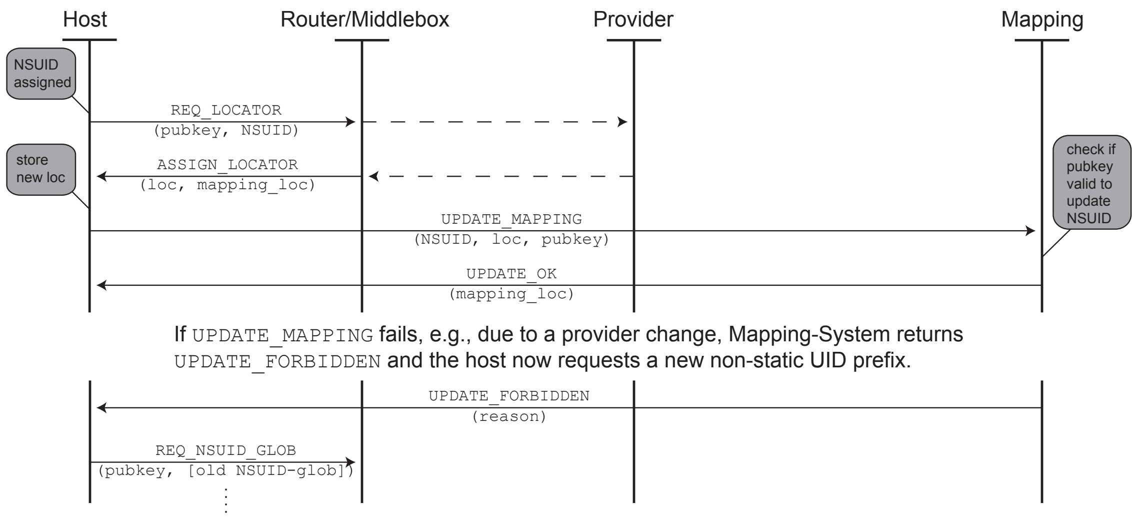
Host
Router/Middlebox
Provider
Mapping
NSUID assigned
REQ_LOCATOR (pubkey, NSUID)
store new loc
ASSIGN_LOCATOR (loc, mapping_loc)
check if pubkey valid to update NSUID
UPDATE_MAPPING (NSUID, loc, pubkey)
UPDATE_OK (mapping_loc)
If UPDATE_MAPPING fails, e.g., due to a provider change, Mapping-System returns UPDATE_FORBIDDEN and the host now requests a new non-static UID prefix.
UPDATE_FORBIDDEN (reason)
REQ_NSUID_GLOB (pubkey, [old NSUID-glob])

Figure 4: Signal flow chart for non-static UID roaming

to change their mapping entries due to roaming although they have not been individually registered.

In order to distribute global parts for non-static UIDs, the provider has to request its pool of non-static UID global parts at the local registry together with its public key. The NIC creates the initial mapping entries for these global non-static UID parts, whereby changes are only allowed by the provider. Figure 3 depicts the complete registration process.

Whenever a home router or middle-box without assigned global UID part tries to connect to the Internet, it requests a global non-static UID part by sending a `REQ_NSUID_GLOB` message to the provider. Note that this message must be sent over a providers line (messages on gray background), as the provider must verify if the device is allowed to request a global UID part. He can do this by, e.g., checking login credentials. Similar to a DHCP request, the middle-box can request a global UID part that has been assigned before (`old NSUID_GLOB`). After the assignment of a global UID part, a valid locator for the device, and the locator of the mapping system (`ASSIGN_NSUID_GLOB`), the middle-box updates its mapping entry. Again, this process must be done over the providers line, as only the provider can update the initial mapping entry, because it has the corresponding private key. Therefore, `mapping_loc` in the `ASSIGN_NSUID_GLOB` message points to a mapping relay at the provider, which signs `UPDATE_MAPPING` with the providers private key and instructs the mapping system to allow updates directly from the device in the future. The mapping system responds with `UPDATE_OK` and a new `mapping_loc`, which directly points to the mapping system. However, `mapping_loc` that points to the providers mapping relay is still stored in the middlebox, as it is needed for internal hosts to initially update their mapping entries.

Internal hosts are sending their request for a global UID part to the middle-box, which assigns the same global UID part received by the provider, together with a locator and the locator of the mapping relay. The update of the mapping entry is the same as for the middle-box.

In case of roaming (Figure 4), the host usually already has an assigned non-static UID. Therefore, it does not request a global UID part at its new access point, but just requests a new locator with `REQ_LOCATOR`. It receives a new locator with `ASSIGN_LOCATOR`, which also includes the locator of the mapping system to update its mapping entry. However, if the update at the mapping system fails, e.g., because the global UID part has become invalid, the mapping system returns `UPDATE_FORBIDDEN` and a reason for the reject. Depending on that reason the host will then start to request a new global UID prefix. Such an error can occur if, e.g., the contract between the provider and the customer that owns the host has expired.

C. Content UIDs

The procedure for content UIDs is basically the same like for static host UIDs. The content creator has to initially register the hash part of the UID at the mapping system. However, it does not need to register each single content object that is provided. Then the content provider can freely create new content that only differs in Ext 1 and Ext 2. By doing so, the content creator always uses the same public-key pair for all content objects, and the mapping system requires a valid signature upon changing any mapping entry.

While this procedure is sufficient for professional or commercial content that is managed by one authority, a different approach is necessary for content that is free to public changes, like articles in Wikipedia. Here, everybody is allowed to create a new version of the corresponding content that differs in Ext 2. However, no new mapping entry is generated for every new Ext 2 value, but the new Ext 2 value including the corresponding locator is added to the existing mapping entry. The changing person has to sign this new entry with its own private key and the mapping system must grant changes not only to the initial creator of the mapping entry. In order to differentiate between content that may be changed by anyone, we have introduced two different type values 3 and 4 that simplify the setting of the correct permissions for each mapping entry.

D. Personal UIDs

Unlike with UIDs for hosts or content, UIDs for persons are assigned by an authority of the state. As the personal UID can be used to make transactions and legal contracts, it has to be guaranteed that the UID cannot be abused. Furthermore, it has to be guaranteed that values for Ext 1 are unambiguous. That would not be the case if everybody would generate its own random value for Ext 1. Thus, during the registration process, an authority assigns a free Ext 1 value and creates the mapping entry for the person requesting a UID. The authority furthermore checks the presence of a valid key-pair and deposits the public key together with the mapping entry in the mapping system. Then, the person can update and create any entry for Ext 2 in its mapping entry on its own. Changing the key pair must always be accomplished through the issuing authority. Like before, Ext 2 is set to zero when querying for the mapping entry of a specific person and solely used when establishing a communication. The persons UID, together with the cryptographic key pair, maybe stored on a chip that is embedded in the ID card. This feature is already supported by many governments today.

V. Stack Interaction

As the current Internet architecture does not support any kind of locator/identifier separation, major changes in the network stack are necessary to enable this new network paradigm. Based on the HiiMap approach and in order to avoid a hierarchical and inflexible design like in the

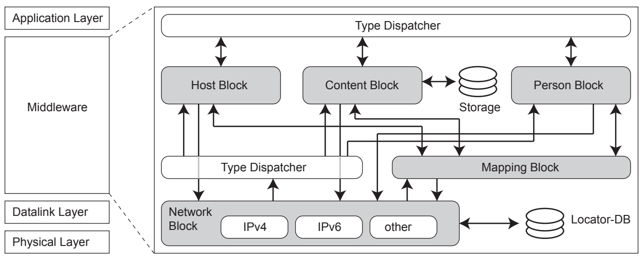

Figure 5: New network stack with middleware and interacting building blocks

Open Systems Interconnection (OSI) reference model, we suggest to use building blocks that can freely interact in the so-called middleware. Applications in the upper layer can choose to which types they want to connect and the middleware provides the mapping lookup, handles the connection setup and keeps track of changing locators. Note that this architecture requires no changes in the datalink and physical layer. Figure 5 shows the schematic model of the middleware. Building blocks are shown in gray and the *Type Dispatcher* forwards incoming and outgoing data packets to the responsible building block. Note this middleware is necessary on all devices connected to the Internet. However, during a migration phase, it is possible to gradually introduce the specific building blocks. Nevertheless, the *Network* and *Host Block* are mandatory from the beginning to allow at least host-to-host communication.

A. Network Block

The task of the network block is to select adequate protocols for inter-networking, according to the network configuration. These can be for example IPv4, IPv6 or future protocols with new addressing schemes and routing techniques. The network block is solely working on locator-level, and prepends the necessary routing and forwarding header to the data packets. Furthermore, the building block is responsible for the locator values of all connected network interfaces and also requests non-static global UID parts if necessary. It keeps track of all UIDs and the corresponding locator values that are being used in the locator database. It notifies all these communication partners upon a local locator change, thus ensuring a interruption-free data transfer. Moreover, it notifies the mapping module to update the entry in the mapping system.

B. Mapping Block

This building block is responsible for keeping the host's mapping entries up to date. Locator changes of the network interfaces are reported from the network block and updated in the mapping system. Despite that, receives requests for UID resolution from other building blocks and forwards these requests to the mapping system. It returns the corresponding locators, content descriptions or personal profiles to the requested building block. The mapping block uses the `mapping_loc` value that is returned at the locator assignment process for the communication with the mapping system.

C. Host Block

Every application that wants to be accessible through a specific service identifier (Ext 2 in host UIDs) must register this identifier at the host block. Incoming connections for UIDs with type "host" are dispatched at the type dispatcher and forwarded to the host block. If any application is listening at the corresponding service identifier, the data is forwarded. Furthermore, the host block handles outgoing connections to other hosts and accepts connection requests to host UIDs. It selects one locator value returned by the mapping block and forwards the data packet to the network block. The host block also keeps track of locator changes at its communicating peers.

D. Content Block

The tasks of the content block contain the reception of any kind of content, including the evaluation of the information object with the content description returned by the mapping system, as well as the provision of own content that is stored locally.

1) Content reception: The request of any application that wants to receive a specific content object is dispatched to the content block. According to the desired UID, the content block requests the information object from the mapping system. Furthermore the application has the possibility to set some filter rules regarding the requested content, e.g., minimal/maximal bit-rate for audio and video files, or resolution for pictures. Mobile devices can benefit from that filtering option, as the application has the possibility to request only the smallest available version of the content. If the filtered content description still has more than one source available, the final choice has to be taken by the application or user respectively.

2) Content provision: The other way round, the content block is also responsible for the provision of content objects that are stored on the local machine. Thereby, content that shall be publicly available is registered at the content block including the corresponding content UID. The content block notifies the mapping block about new available content, which in turn registers the content UIDs at the mapping system. Requests for content UIDs are dispatched to the content block, which delivers the desired content to the inquirer. Note that the content block must have access to the storage location of all registered content objects.

E. Person Block

The person block is responsible for any kind of personal communication between users. Every application dealing with chat capabilities, instant messaging, email, and voice-

or video communication registers itself at this building block. According to the connected applications and settings of the user, the person block, in conjunction with the mapping block, modifies the mapping entry of the persons UID in the mapping system. The personal UID is configured via a card reader and the persons ID card that also stores the persons private key. Incoming connections are immediately forwarded to the corresponding registered application. If the user agrees to continuously update the locator(s) of his machine, the person block takes care of this action.

VI. Lookup mechanism

The idea of our naming scheme for IDs is based on the fact that each UID can be generated out of a known plain text string with a known hash function and without an additional naming system like DNS. However, as the main part of any UID consists of a hash function, the desired entity can only be found if the plain text string that builds the UID is exactly known and no spelling mistake or typing error occurred. This is particularly a problem for querying UIDs, where phonetical identical search strings can be spelled in different ways, e.g, "color" and "colour". To overcome this drawback, we suggest a lookup mechanism that is based on n-grams in addition to the pure UID lookup. Harding [23] and Pitler [24] showed that the usage of n-gram based queries can significantly improve the detection of spelling errors in noun phrases and word tokens. Note that this feature must be supported and implemented in the mapping system as well as in the mapping block of each querying machine [25].

A. n-gram generation

Although DHTs only support exact-match lookups, it is possible to use n-grams to perform substring and similarity searches. Hereby, each plaintext string is split up into substrings of length *n*, which are called *n-grams*. The hash value of each n-gram together with the corresponding complete plaintext string is then stored as key/value pair in the DHT [25].

A typical value for n is two or three. As an example with $n = 3$, the content name P set to `nytimes` is split up into $I = 5$ trigrams h_i with $i = 1, ..., I$: `nyt`, `yti`, `tim`, `ime`, `mes`. In addition to the actual mapping entry indexed by the UID, the hash value $H(h_i)$ of each n-gram h_i is inserted in the mapping system together with the corresponding plain text name P. Thereby, the mapping entry for an n-gram consists of the tuple $\langle H(h_i); plaintext\ string \rangle$ [20]. Although these tuples are stored in the same mapping system like the UID, we suggest using a different database within the mapping system for performance reasons. Whenever the entity changes its location, no updates of the n-grams are necessary, as they do not contain any locator information but only the entities' plaintext name.

B. Querying UIDs

Whenever querying the mapping system for a specific UID, the first step in the lookup process is using the pre-calculated (or already known) UID as query parameter. Only if the mapping system is not able to find a mapping entry to the corresponding UID, e.g., because of a spelling mistake, the n-gram lookup is executed. It is up to the user or application if an n-gram based query request is initiated. An n-gram based query can also be initiated if the query does not return the desired result.

In doing so, the second step is to calculate the corresponding n-grams out of the plaintext string and query the mapping system for each n-gram. The mapping system sorts all matching n-grams according to the frequency of the plaintext string and returns the list to the user. With high probability, the desired plaintext has a high rank in the returned list. By further correlating the input string with each returned plaintext string, e.g., by using the Levenshtein distance, the result is even more precise [26].

As the results returned by an n-gram query must be further evaluated, the mapping block in the middleware will forward that data to the corresponding building block. This block can already select the best matching result, or forwards the possible choices directly to the application, which is responsible for correct representation to the user that takes the final decision. However, although this feature is similar to Google's "Did you mean...?", the mechanism is not suitable to handle complex queries with semantically coherent terms as Google can do. However, it can help to significantly improve the quality of returned search results and thus the quality of experience to the user, as phonetical similar words can be found despite different spelling.

VII. Conclusion

In this work, we presented a new naming scheme for IDs in locator/ID separated Future Internet Architectures based on the HiiMap proposal. The generalized ID scheme is suitable for basically addressing any kind of entity and we gave examples for hosts, content and persons. As each UID can be computed out of a human readable plaintext string, an additional naming system like DNS is not necessary any more. Due to the extendible type field, we have the possibility to assign ID-types also for, e.g., mobile phones, sensors, or even cars or abstract services that provide any functionality to a user. As IDs are independent from locators, a communication session is not interrupted upon an access point change. We showed instructions how new UIDs for each type are assigned and registered and which changes in the network stack are necessary in order to enable the addressing scheme proposed in this work. Furthermore, by introducing an n-gram based extended lookup mechanism we are able to cope with spelling errors and typing mistakes.

Acknowledgment

This work was funded by the Federal Ministry of Education and Research of the Federal Republic of Germany (Project ID 01BK0807). The authors alone are responsible for the content of the paper.

References

[1] C. Spleiß and G. Kunzmann, "A Naming scheme for Identifiers in a Locator/Identifier-Split Internet Architecture," in *ICN 2011, Proceedings of 10th International Conference on Networks*, Sint Maarten, Netherland, January 2011, pp. 57–62.

[2] A. Afanasyev, N. Tilley, B. Longstaff, and L. Zhang, "BGP routing table: Trends and challenges," in *Proc. of the 12th Youth Technological Conference Ḧigh Technologies and Intellectual Systems¨*, Moscow, Russia, April 2010.

[3] X. Meng, Z. Xu, B. Zhang, G. Huston, S. Lu, and L. Zhang, "IPv4 address allocation and the BGP routing table evolution," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 1, p. 80, 2005.

[4] ISC, "The ISC Domain Survey," http://www.isc.org/solutions/survey, Internet System Consortium, 2010.

[5] B. Quoitin, L. Iannone, C. de Launois, and O. Bonaventure, "Evaluating the benefits of the locator/identifier separation," in *Proceedings of 2nd ACM/IEEE International Workshop on Mobility in the evolving Internet Architecture*, 2007, pp. 1–6.

[6] V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs, and R. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM, 2009, pp. 1–12.

[7] D. Cheriton and M. Gritter, "TRIAD: A new next-generation Internet architecture," Tech. Rep., 2000. [Online]. Available: http://www.dsg.stanford.edu/triad

[8] C. Dannewitz, "NetInf: An Information-Centric Design for the Future Internet," *Proceedings of 3rd GI/ITG KuVS Workshop on The Future Internet*, May 2009.

[9] O. Hanka, G. Kunzmann, C. Spleiß, and J. Eberspächer, "HiiMap: Hierarchical Internet Mapping Architecture," in *1st International Conference on Future Information Networks*, October 2009.

[10] M. Menth, M. Hartmann, and D. Klein, "Global locator, local locator, and identifier split (GLI-split)," *University of Würzburg, Institute of Computer Science, Technical Report*, 2010.

[11] 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*. ACM, 2009, pp. 43–48.

[12] J. Pan, S. Paul, R. Jain, and M. Bowman, "MILSA: A Mobility and Multihoming Supporting Identifier Locator Split Architecture for Naming in the Next Generation Internet," in *IEEE GLOBECOM*, 2008, pp. 1–6.

[13] M. Menth, M. Hartmann, and M. Höfling, "FIRMS: a mapping system for future internet routing," *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 8, pp. 1326–1331, 2010.

[14] D. Farinacci, V. Fuller, D. Oran, D. Meyer, and S. Brim, "Locator/ID separation protocol (LISP)," Draft, 2010. [Online]. Available: http://tools.ietf.org/html/draft-ietf-lisp-07

[15] R. Moskowitz and P. Nikander, "Host identity protocol (HIP) architecture," RFC 4423, Tech. Rep., May 2006.

[16] V. Kafle and M. Inoue, "HIMALIS: Heterogeneity Inclusion and Mobility Adaptation through Locator ID Separation in New Generation Network," *IEICE TRANSACTIONS on Communications*, vol. 93, no. 3, pp. 478–489, 2010.

[17] O. Hanka, C. Spleiß, G. Kunzmann, and J. Eberspächer, "A novel DHT-based Network Architecture for the Next Generation Internet," in *Proceedings of the 8th International Conference on Networks*, Cancun, Mexico, March 2009.

[18] ITU, "Draft Recommendation ITU-T Y.2015: General requirements for ID/locator separation in NGN," International Telecommunication Union, 2009.

[19] N. Paskin, "Digital Object Identifier (DOI®) System," *Encyclopedia of library and information sciences*, pp. 1586–1592, 2010.

[20] G. Kunzmann, "Performance Analysis and Optimized Operation of Structured Overlay Networks," Dissertation, Technische Universität München, 2009.

[21] C. Spleiß and G. Kunzmann, "Decentralized supplementary services for Voice-over-IP telephony," in *Proceedings of the 13th open European summer school and IFIP TC6. 6 conference on Dependable and adaptable networks and services*, 2007, pp. 62–69.

[22] O. Hanka, "A Privacy Service for Locator/Identifier-Split Architectures Based on Mobile IP Mechanisms," in *Proceedings of 2nd International Conference on Advances in Future Internet*, Venice, Italy, July 2010.

[23] S. Harding, W. Croft, and C. Weir, "Probabilistic retrieval of OCR degraded text using N-grams," *Research and advanced technology for digital libraries*, pp. 345–359, 1997.

[24] E. Pitler, S. Bergsma, D. Lin, and K. Church, "Using web-scale N-grams to improve base NP parsing performance," in *Proceedings of the 23rd International Conference on Computational Linguistics*. Association for Computational Linguistics, 2010, pp. 886–894.

[25] M. Harren, J. Hellerstein, R. Huebsch, B. Loo, S. Shenker, and I. Stoica, "Complex queries in DHT-based peer-to-peer networks," *Peer-to-Peer Systems*, pp. 242–250, 2002.

[26] L. Mangu, E. Brill, and A. Stolcke, "Finding consensus among words: Lattice-based word error minimization," in *6th European Conference on Speech Communication and Technology*, 1999.