UOA: User-Oriented-Addressing for Slice Computing

Maoke Chen* † Akihiro Nakao‡ Olivier Bonaventure§ Taoyu Li*

* Tsinghua University, Beijing, China
† National Institute of Information and Communications Technology (NICT), Tokyo, Japan
‡ The University of Tokyo, Tokyo, Japan
§ Université catholique de Louvain, Louvain-la-Neuve, Belgium

***Abstract*—Sharing computing resources over the Internet has become a popular approach for creating innovative network services. This emerging model of using networked computers is called "slice computing". The term "slice" refers to an intuitive description for the collection of isolated resources. Isolation in namespace, security and performance is the key to preventing interference among users sharing the same slice computing environment. Observing that the network identifier is the most essential resource to be isolated, we try to add a minimal change on top of the OS built-in features to enable network namespace isolation. The idea is assigning network addresses to user identifiers within a native OS. It is different from the traditional Internet addressing architecture where IP addresses are assigned to network interfaces and shared by all users. Accordingly, we call it User-oriented Addressing, or UOA in short. Case studies illustrate that UOA can be also applied to facilitating innovation in routing architectures.**

***Index Terms*—Slice computing, isolation, Internet addressing architecture, overlay management**

I. Introduction

The way of humans using computers is changing from sharing a mainframe to having individual PCs and then to networking them to each other. New trends in computing are emerging where inter-networked computers, not only their data or specific type of resources, but their whole computational facilities and environments, are merged into a single system and re-allocated to multiple users. Users benefit not only from the summation but also, more importantly, from the networking of these computers, enabling innovative network services. These trends have been reflected in the success of PlanetLab [19], where research communities put their PC servers into a shared testbed for a global scale network experiments. Taking the PlanetLab as one of the prototyping approaches, the project of Global Environment for Networking Innovation (GENI) builds a federated facility to support long-term research and development activities among the communities [6].

We define the "slice computing", borrowing the term "slice" from the PlanetLab, for a new model where computers are organized into a shared, uniformed infrastructure. Each user utilizes a certain unit of resources on each of individual computers within it, and organizes its own, overlay networks with these units. Each unit is treated as a "logical computer" in the system, which we call "sliver" as in PlanetLab. A slice consists of multiple slivers distributed across the Internet, while a sliver usually stays in a single node. It is important to prevent interferences among different slices on the same physical node and to avoid cross-talks between applications running on top of different slices. Resource isolation is a technique used for avoiding interferences and is categorized into three types, each targeted at the following resources, respectively: namespace, security context, and performance context. A variety of virtualization approaches has been applied into slice computing, with attempts to isolate these three types of resources. For example, root-change (chroot) approaches, such as Linux VServer [3] and FreeBSD Jail [1], are employed in platforms of PlanetLab and EmuLab [15]; "paravirtualization" of Xen [5], [9] is applied in Grid5000 [2]; A full hardware-emulation approach such as VMWare [4] is not widely used for slice computing due to the limited scalability in terms of the number of slices.

In this paper, we propose a new approach to the slice computing through adding a minimal change to the OS's built-in resource isolation to enable isolation of network namespace. We observe that native operating systems (OS) already implement access permissions (for the security context) and allocations of disk, CPU as well as bandwidth among users and their processes (for the performance context), but have no means for the isolation of network namespace, i.e., the network addresses that identify slices. Although any of the above virtualization approaches can certainly resolve the namespace isolation, they are often over-engineered to enable more features than are necessary, sacrificing performance and flexibility over such stringent resource isolation.

Our proposal is to assign each *user id* in an OS to an individual IP address and to map a slice identifier to a *user id*. Therefore, we call the proposal "User-oriented Addressing" or UOA, in short, in contrast to the traditional Internet addressing architecture where IP addresses are assigned to interfaces of a host and users on the same node have to share those addresses without distinction even if there are a number of addresses.

UOA can be used to support a network testbed for innovations of new network services, which is the most popular usage of the slice computing. In a broader sense, UOA offers a novel fashion of using addresses in IP networks. The Internet community is trying to split the role of IP address into identifier (EID) and routing locator (RLOC). They have proposed network-based [13], [22] and host-based techniques [17]. These techniques benefit from UOA, as user-oriented EID can meet different requirements on EID/RLOC mapping for users running different applications. The community also attempts to allow end systems leveraging end-to-end routing by selecting source and destination addresses in a multi-homed environment [25]. UOA can be leveraged to accommodate different demands of users who are sharing the hardware and the network connectivity of the same host.

In the other work[11], we have identified the motivations and design decisions of the UOA-supported *node system* for a networking test-bed and have implemented it. Different from that, this paper expands our view into generic slice computing, and focuses on the *architecture* level issues regarding the UOA approach, i.e., how the IP addresses, which are to be directly assigned to *user id*s rather than network interface, should be formatted, generated, managed and utilized differently from before.

The rest of the paper is organized as follows. First, we focus on UOA architecture for a large-scale testbed. Section II enumerates the requirements for slice computing in terms of addressing and Section III illustrates the necessary components and operations in detail. Then, in Section IV, we discuss the variants of UOA architecture in testbed deployment and in routing innovations. Section V concludes the paper with future work.

II. Design Considerations

In the Internet, an address (either IPv4 or IPv6) is defined as the identifier for an interface of a host connected to a certain network. A so-called multi-homed host may have several interfaces, thus multiple addresses. However, in such a host, there exists no mechanism limiting the access to a particular interface or address to a given *user id*. Application process can either select any address or let the kernel automatically pick one as the default address for its communication.

The lack of such address isolation makes a native OS not suitable for slice computing, or results in inconvenience in security protection. For example, port contention often occurs among different slivers, if the applications on the slivers are randomly initiating connections. For another example, tracking back malicious behaviors for each user's (*address*, *port*) usage over time could be eased if such activities could be traced to a particular set of users.

UOA aims to enable *address isolation* that limits access to an address to a given slice. It is designed as a minimal change atop the built-in features of network namespace isolation in operating systems. It can be viewed as a light-weight alternation for those over-engineered virtualization techniques employing either hardware emulation and root environment change. It is expected that built-in security and performance isolation mechanisms plus this newly enabled address isolation are adequate to support slice computing where moderate security isolation is enough.

In the view of a remote peer in the slice computing environment, UOA creates a sliver for *user id* inside an OS. In comparison to the traditional interface-oriented addressing architecture, the introduction of UOA brings two significant consequences: (1) it utilizes a large number of addresses on a single host, thus address management becomes a non-trivial issue; (2) when we consider the situation where a sliver can migrate to another hardware, we may have to remap IP address for such a sliver. The migration of sliver in UOA poses exactly the same remap problem as in ID/Locator separation discussion elsewhere, since UOA maps slivers to the traditional IP addresses. These two consequences imply that the realization of UOA involves not only the implementation within OS but also addressing architecture.

However, architecture design is related to the detailed usage of the slice computing. We first regard the slice computing as a platform for network innovations. As an example, we explore its design considerations and further illustrate the architecture of UOA for platforms. Then, we briefly outline the UOA for routing innovations as a variant case study.

A. Assumptions

Regarding UOA as a platform for network innovations, we assume the following prerequisites for its design.

1) There is a slice manager (SliceMan) globally deployed [21], [20]. SliceMan maintains a database for all the involved users and their slices and all the nodes.
2) SliceMan holds an enough amount of IP addresses to be assigned to users.
3) Each hardware node has a multi-user operating system installed.
4) Each hosting system has a pre-defined `root` (or the Local System account in the Windows, equivalently) associated with a reachable IP address and runs a set of necessary services over the address. It is also typical that the OS defines a set of system account, i.e., the *user id*s only used to running services.

Assumption *1* conforms to the overall architecture of GENI slice computing [21]. Assumption *2* implies that UOA architecture is basically designed for IPv6; in IPv4, however, even though the idea of assigning addresses to users can be implemented, the architecture is hard to be comprehensively deployed with the limited address space. Assumption *3* is satisfied with most main-stream operating systems including UNIX, Linux and Windows, while Assumption *4* ensures the connectivity between the SliceMan and each hardware computer and also the accessibility of any user to a computer from any remote client.

B. Requirements on Address Uniqueness

The concept of Internet address uniqueness is comprehensively defined and discussed for both IPv4 and IPv6 several year ago [10]. Up to now, the problem proposed in RFC2101 — the contrast between stable identifier and dynamic locator — still exists. People are attempting to solve the problem by splitting the roles of EID and RLOC of IP addresses.

A sliver for *user id* is identified by peer with a unique IP address. However, without the EID/RLOC-separation, we cannot expect the uniqueness can be fully supported. We only require that different slivers have different addresses (spatial uniqueness) while one sliver has a distinct address or a group of addresses (temporal uniqueness).

1) Spatial uniqueness: Spatial uniqueness in the Internet means that one address identifies the unique host connected to a certain network over the Internet. In UOA, the same sentence above holds with *host* replaced with *user id* on a host. The current automatic address assignment methods, including DHCP[12] and IPv6 auto-configuration[24], typically assign a unique IP address to an interface of a host, identifying an IP address requester by its link-layer address.

2) Temporal uniqueness: Temporal uniqueness ensures a host, which an address identifies, can be identified with the same address for a duration of time. In UOA, the sentence above holds with *host* replaced with *user id*. When a sliver migrates from one node to another in the same network, temporal uniqueness requires the sliver to have the same IP address as before.

Without the EID/RLOC separation, the full support of temporal uniqueness is not possible to be achieved if a sliver migrates across different networks. In this case, the network prefix has to be changed, since the sliver moves to a different network, and then the temporal uniqueness is violated. Once UOA and EID/RLOC-separation are merged into a new addressing-routing architecture, the temporal uniqueness will be well supported.

C. Requirements in Security Management

From the management view, identifying the origin slice of a certain traffic is important for slice computing. IP address plays the role of a handle for the origin. In order to make the temporal trace-back of malicious behavior, the user-oriented addresses must be static rather than frequently changed. Furthermore, since a slice spans across multiple nodes, thus, is mapped to a set of addresses, those addresses should share the same prefix or suffix, or some token embedded in the addresses that identifies the slice to ease the trace-back.

On the other hand, in the context where we need to protect the privacy of the slice identification, we must employ hashing or encryption in generating addresses for a slice, so that the slice identity cannot be easily tracked down except by the authority that has generated the addresses. Hashing and encryption are also required to prevent spoofing and port scanning. If the addresses are contiguously utilized, attackers can easily predict all the addresses and attack them.

D. Requirements on Compatibility and Extensibility

The existing applications should still work transparently with UOA, without any change and recompile on the part of the applications. There are some applications running with the mechanism of *effective user id* for execution, usually for the purposes of taking over the privileges of the other *user id*. With UOA, the mechanism of *effective user id* grants a sliver the right to utilize the addresses assigned to another sliver.

In the case where a group of users share the same slice (like several users having the same slice in the PlanetLab), a group-wise address is required; on the other hand, assigning IP address per application may simplify firewall or traffic control regarding such application. We expect that group-wise and application-wise addresses can be directly achieved with proper configurations of multiple users having the same address and one user having multiple addresses, respectively. To this requirement on extensibility, UOA defines the correspondence from *user id* to IP address many-to-many rather than one-to-one or one-to-many.

E. Benefits

As a summary for the design objectives and requirement analysis, we expect the following benefits if UOA is implemented and deployed.

- *Easy and fast deployment of a slice computing platform*: UOA can be merged into a native OS as a built-in feature and turns such OS into a platform for slice computing easily.
- *Enlarged and isolated port space for applications*: UOA isolates port usage among users and reduces port conflict in the multi-user environment.
- *Fine grained routing control*: if combined with EID/RLOC separation or the NIRA approach [25], UOA reflects the diversity in user's demand on path selection and enables each user leveraging routes by changing addresses for its packet delivery independently in a multi-homed environment.

F. Limitations

There are some inevitable limitations in UOA.

- *Not supporting customized kernels*: since we design UOA as a minimal patch to a stock multi-user OS, slivers will share the same kernel and execution environment, just as in resource container virtualization.
- *Not supporting run-time states migration*: again, since UOA designed on top of a stock OS, live migration of run-time states are not supported unless the stock OS supports. Application-level migration is still possible, however, for example, Migratory TCP may be useful [23].
- *Overhead in neighbor discovery*: UOA increases the workload for neighbor discovery (or address resolution in IPv4) in both neighboring hosts and leaf routers[1]. Sliver migration in UOA also suffers from renew period of neighbor discovery.

III. UOA ARCHITECTURE

UOA is an IP addressing model. The "architecture" of an IP addressing model refers to the formation, management, validation and usage rules of the addresses as well as the role of addresses in routing. The IPv4 addressing architecture has undergone a lot of changes, from class-specific to subnet addressing and finally to the classless approaches [14]. In IPv6, each interface can be configured with any number of

[1]The size of ARP or Neighbor Discovery (in IPv6) cache is often a hot topic discussed among the engineers. See, e.g., a discussion entitled as "neighbor/ARP cache scalability" in the following mail archive: http://www.linux.sgi.com/archives/netdev/2004-09/msg00879.html

addresses[16]. The IPv6 addressing architecture also defines the scope and format of addresses.

A. Types of User-oriented Addresses

In a slice computing environment, a user-oriented address is the network identifier for a sliver of a slice, which is mapped to *user id* in a certain hardware computer. According to Assumption *4*, there are also some *user id*s on a host used for running services only, not mapped to any slice. To make the UOA implementation self-contained, we treat such *user id*s also as "user-oriented" but the term "user" here has only the significance of local *user id* rather than an entity mapped to a slice. Therefore we have two classes of user-oriented addresses: slice-wise addresses, which are mapped to slices, and node-wise, which are only used within one OS distinguishing contexts of different *user id*s.

Node-wise user-oriented address is defined as a function of the *user id* within a node. Management of the node-wise user-oriented addresses, including the assignment and withdrawal, is undertaken by node administers. A node administrator obtains addresses prior to assigning them as node-wise user-oriented addresses to *user id*s. Node-wise UOA is quite simple and the direct result of the implementation enabling the UOA feature in operating system.

The following architecture discussions focuses on the slice-wise user-oriented address. If not specifically indicated, the term "user-oriented address" refers to that type only.

B. Definition of Address

As manageability discussed in Section II-C requires, a user-oriented addresses must be mapped to slice-specific information.

Thus, a user-oriented address (*uo_addr*) can be defined as a function (ψ) of the triple of a network prefix (*np*), a global slice id (*slice_id*) and the time of the assignment (*t_a*), as shown in Eqn. (1). At each hardware node, a *slice_id* is typically mapped to a certain *user id* with a specific *user name*, which is generated from the *slice_id* by a rule.

$$uo\_addr = \psi(np, slice\_id, t\_a) \tag{1}$$

It is not necessarily required that the *t_a* is measured uniquely across slices. Each slice can define the epoch for its clock independently.

It is possible that a *user id* of a node is assigned with more than one *uo_addr* generated from the same *slice_id*, exactly the same *t_a* but different np. This is the typical assignment in a multi-homed environment, where more than one networks connect the host.

A user-oriented address *uo_addr* must have the form of the combination of the network prefix *np* and a user-oriented suffix, which is similar to the interface ID in IPv6 addressing architecture, but it is not for an interface and therefore we call it a *pseudo-interface id*, or *pseudo_iface_id* in the formula.

Then the instance of function ψ in Eqn. (1) can be further written in Eqn. (2), where φ_1 is a function hiding the user privacy in the $pseudo\_iface\_id$ while φ_2 is a simple function, as long as it can differentiate instances of the same slice at different nodes. The operator "||" represents concatenation of two bit-strings.

$$\begin{aligned} \psi(np, slice\_id, t\_a) &= np \parallel pseudo\_iface\_id \\ pseudo\_iface\_id &\triangleq \varphi_1(slice\_id) + \varphi_2(t\_a) \end{aligned} \tag{2}$$

C. Generation of the Addresses

The idea behind φ_1 is similar to the Cryptographically Generated Addresses (CGA) [7] and the Hash-based Addresses (HBA) [8] but there are some obvious differences. CGA encodes public key with the MAC address of an interface while HBA links the prefixes of the multihoming providers into the host-id part of address together. We can simply select a well-known cryptographic or hash function for φ_1 to achieve this.

Theoretic discussion on the strength of the hash function is out of the scope of paper. We demonstrate a simple algorithm in the evaluation part, but that simple version can be replaced by any more sophisticated techniques in the real deployment.

D. User-oriented Address Management

The address management involves assignment, configuration and withdraw of addresses. It also involves transfer of addresses when we consider migration of slices from one hardware to another. In a slice computing platform where SliceMan exists, it coordinates the address management through specific protocols.

SliceMan in UOA maintains node identifier, *node_id*, and slice identifier, *slice_id*, indexed by assignment time, *t_a* in its database.

$$(slice\_id, t\_a) \mapsto node\_id \tag{3}$$

Without *Eid/Rloc* separation, *np* has to be selected from those prefixes owned by the *node*, and the *pseudo_iface_id* may be one or many of all the possible prefixes. SliceMan decides which addresses are assigned for a certain a slice at a node. If we had the EID/RLOC separation, a slice could be assigned to an EID with a prefix *np*.

Assigning a user-oriented address to a slice inserts an entry into the relationship (3), while withdrawing one deletes the corresponding entry. Transferring from one node to another is to update the value *node_id* of the corresponding record.

1) Assignment and withdrawal: The process of user-oriented address assignment and withdraw in a slice computing are depicted in Fig. 1. A slice user sends a request to SliceMan, which contacts the selected node. The node configures a proper local *user id* for the slice and then takes up the *uo_addr* assigned by SliceMan, configuring it with an interface connected with the network specified by the prefix *np* and binding it to the local *user id*. The timeline chart in Fig. 1 shows self-explanatory operation primitives to depict this process.

We intentionally decouple the *unit_add* and the *iface_up*, making the two operations atomic. The decoupling is significant in the process of address transfer.

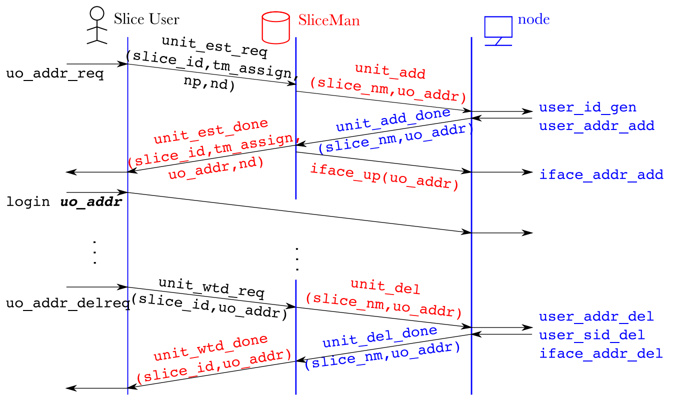


Fig. 1. Timeline for user-oriented address assignment and withdrawal

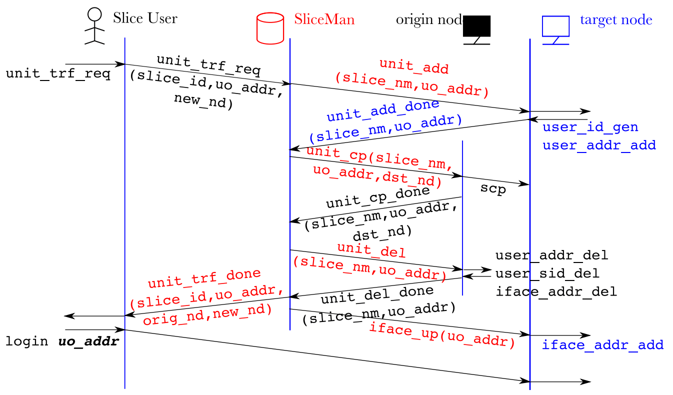


Fig. 2. Timeline for user-oriented address transfer

In IPv4, there are definitely few addresses possibly to be used, then we are forced to be back applying node-wise user-oriented addresses, i.e., let node administrators specify addresses, which they have already held, to *user id*s rather than getting the addresses from the SliceMan. Afterwards, the local administrator should register the assignment to the SliceMan. The logic has to be inverse in comparison to the above design.

2) Address transfer: Address transfer is an interesting mechanism in UOA architecture. When migrating a sliver from one node to another, we also need to transfer its user-oriented address, keeping the temporal uniqueness of the user-oriented addresses. Note that we may have to change the network prefixes for the address, if the migration of the sliver occurs across different networks; otherwise it is enough to simply transfer the address of the sliver from one hardware to another.

Fig. 2 illustrates the design for the transfer timeline. It is necessary to initiate a sliver (a *user id* and its home directory, etc.) on the target node first, and then copy it from the origin to the target.

The *uo_addr* can be enabled for the target *user id* a little earlier but it is not enabled on a certain interface until the original node has deleted the address. We need a strict synchronization mechanism when the target and the origin are in the same network to avoid address contention.

The process depicted in Fig. 2 shows only the transfer for a part of the sliver context—the data and programs stored in the file systems. This forms a basis of address transfer, and the complete sliver migration follows including states and live connections, etc.

E. User-oriented Address Sharing

Sharing user-oriented addresses among slices violates the requirement on spatial uniqueness. Although the architecture does not permit this, it may be sometimes useful in practice. In the circumstance where IP addresses are scarce, we have to share the precious addresses among different slices. In the case where slivers are managed into groups, it also makes sense to share a user-oriented address within a group.

1) Node-wise addresses sharing: Assumption *4* has stated the *root* account in the OS doesn't match any slice but also holds node-wise addresses. System accounts are of the same case as well. Because typically a system account is only used for a certain service or a certain type of services (e.g., the account *nobody* is used for HTTP daemon), it is not necessary to assign one dedicated address to each of them. Instead, we can let them just share the addresses held by the *root*.

2) Slice-wise address shared by users on the same node: Sharing a slice-wise user-oriented address doesn't affect the definition described in (1), but does affect the address management and utilization.

When a *uo_addr* is shared by a few slices, the operations *iface_addr_add* and *iface_addr_del* in Fig. 1 and Fig. 2 are implemented with a check on the address status. If another user has added the address to a node, then *iface_addr_add* does nothing. If anyone other than the *user id* still uses the address *uo_addr*, then the *iface_addr_del* cannot really delete this address from the interface. Those *user id*s sharing the same *uo_addr* also share the same space of port numbers, and the port selection and collision detection mechanisms in the vanilla protocol stack and application programming interface (API) take effect to prevent the contention.

3) Slice-wise address shared among different nodes: The same address might be shared *under the same user id (slice id)* at different nodes within the same network, although this appears to cause address contention at the first glance. In other words, two or more slivers belonging to the same *user id* (*slice id*) would share the same address. Duplicating slivers onto a different node would bring such situation. In this special case, the shared address could be reached by the *anycast* approach[18]. However, a slice user who attempts to do this address sharing must keep in mind that two (or more) slivers of the same address are logically dealt as the same sliver.

Cross-network sharing user-oriented address is not supported by the current Internet architecture but would be available with the help of EID/RLOC separation.

IV. VARIANTS OF UOA ARCHITECTURE

In the previous two sections, we have discussed the architecture of UOA for a large-scale network innovation platform. In this section, we first make a summary on what else are needed for a PlanetLab-like platform supported by UOA, and then take the NIRA as the example, explaining how UOA can impact on a new routing architecture.

A. *UOA-supported PlanetLab*

The idea of UOA originates from the practice in large-scale experiment platform, like PlanetLab. A UOA-supported PlanetLab would consist of a group of UOA-enabled nodes and the slice manager (SliceMan) in the Slice Facility Architecture [21]. SliceMan administers the network namespace and also the computational resource allocation regarding the slivers (*user ids*).

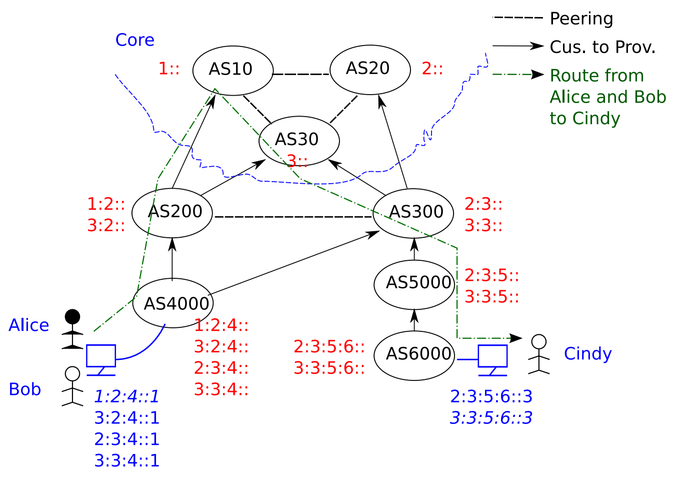
For network namespace, UOA enables isolated IP addresses among the *user id*s but their route (forwarding) tables are still shared. With the help of the Policy-based Routing (PBR) technique, e.g., the **ip rule** tool in Linux, it is quite easy to define each user's forwarding table by setting the matching condition with source addresses of the *user id*s. If PBR is integrated in UOA, slivers can easily control the outbound route (next hop). However, controls beyond the first hop and controls over the incoming traffics are only available with the help of EID/RLOC-separation.

Resource allocation on the UOA-supported PlanetLab would rely on the mechanism already enabled in the native OS. For disk space, disk-quota tools is applied. For CPU time, operating systems have provided a set of the system calls for scheduling, e.g., the Real-Time Scheduler (RTS) in the Linux. For outbound bandwidth, traffic controller like **tc** that work through hierarchical token bucket can be applied. However, RTS can only schedule processes, so in UOA-supported PlanetLab, we would have to extend RTS to schedule slivers. The tool **tc**, on the other hand, controls packets based on their attributes, such as source address and port, destination address and port, and protocol. We should be able to apply address-based control with **tc** to enabling traffic control at the granularity of slivers.

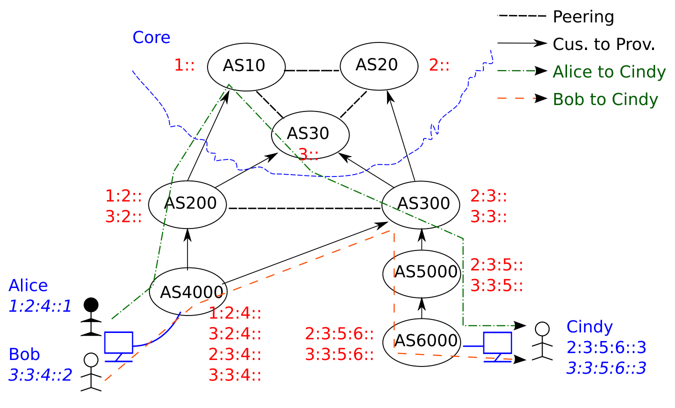
B. *UOA-supported Routing Innovation*

Proposals for new routing architectures, including EID/RLOC-separation and NIRA, introduce a variety of controls over routing for end-systems. A common strategy among these proposals is to interpret an address as a locator. UOA can enhance all of these approaches with providing fine-grained control over each individual user. In a slice computing environment, this is much useful because one user cannot predict a demand for a different routing from another user sharing the same hardware.

We take the example of NIRA to demonstrate how UOA can enhance its routing architecture. NIRA has proposed an idea of hierarchically assigning addresses from the core to the edge. An end-to-end route in NIRA is combined by an uphill section determined by the source address and a downhill section by the destination [25]. An example of NIRA routing is shown in Fig. 3 (a). A host is connected to AS4000 and accordingly has four addresses with different prefixes. Another host is connected to AS6000 and has two prefixes. Between them there are eight routes but, after the negotiation among the two ends, the hosts will select one by enabling their addresses in use. For example, here the host connected to AS4000 takes the address 1:2:4::1 (shown as italic) and the AS6000's host takes 3:3:5:6::3 and then the route AS4000 → AS200 → AS10 → AS30 → AS300 → AS5000 → AS6000 will be taken.



(a) Vanilla NIRA: Alice and Bob have the same selection on the routes towards Cindy



(b) UOA-enhanced NIRA: Alice and Bob have diverse selections

Fig. 3. UOA combined with NIRA, supporting user-specific route control

Such route selection is enabled for all users' processes on the same host. Suppose there are two users, Alice and Bob on the host in AS4000. Obviously, they have to follow the same selection of the route towards Cindy on the peer host. However, Alice and Bob may have different understandings and different demands for the end-to-end routing. Therefore, they would like to leverage their paths independently.

Fig. 3 (b) describes the case where UOA assists the NIRA routing. We enable Alice with the address 1:2:4::1 while Bob with 3:3:4::2. As the result, the traffic from Alice to Cindy will take the route as the above case, but the traffic from Bob to Cindy will take another: AS4000 → AS300 → AS5000 → AS6000. This example demonstrates that UOA enhances the NIRA with enabling routing leverage towards different *user ids* on an end system.

V. Conclusions

We summarize the contributions of this paper.

First, based on the understanding that an IP address identifies a logical computer in the Internet, we propose assigning IP addresses directly to *user id*s within operating system in order to support slice computing environment.

Second, identifying the requirements in address uniqueness, manageability and privacy, we have designed the UOA architecture with answering the question as to how a user-oriented address should be defined, formatted, assigned, removed, transferred and shared among users of slice computing environment.

Moreover, UOA is not only an approach to facilitate slice computing, but also an enhancement for addressing and routing architecture. For new routing architecture enabling end *system* control over the end-to-end routes, UOA enhances it to the control of end *users*. More detailed analysis and experiment with routing leverage with user-oriented addresses are left as our future work.

VI. Acknowledgement

We thank following persons for their discussions and comments on the UOA problem statement, architecture design and system implementation: Prof. Xing Li from Tsinghua University, Damien Saucez and others from IP Networking Lab of UCL, Louvain-la-Neuve; Prof. Kurt Tutschku from Univerisität Wien; Prof. Xiaowei Yang from Duke University and Prof. Minghua Chen from CUHK.

References

[1] Freebsd architecture handbook. http://www.freebsd.org/doc/en/books/arch-handbook/jail.html.

[2] Grid'5000. http://www.grid5000.fr/.

[3] Linux vserver. http://www.linux-vserver.org/.

[4] Vmware. http://www.vmware.com/.

[5] Xen. http://www.xen.org/.

[6] T. Anderson, L. Peterson, S. Shenker, and J. Turner. Overcoming the internet impasse through virtualization. Technical Report GENI Design Document GDD-05-01, GENI Planning Group, Apr. 2005.

[7] T. Aura. Cryptographically Generated Addresses (CGA). RFC 3972 (Proposed Standard), Mar. 2005. Updated by RFCs 4581, 4982.

[8] M. Bagnulo. Hash based addresses. Internet-draft, draft-ietf-shim6-hba-06 (work in progress).

[9] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.

[10] B. Carpenter, J. Crowcroft, and Y. Rekhter. IPv4 Address Behaviour Today. RFC 2101 (Informational), Feb. 1997.

[11] M. Chen, A. Nakao, and T. Li. Isolating IP addresses among users in native OS for light-weight slicing in planetary scale testbed. Under submission.

[12] R. Droms. Dynamic Host Configuration Protocol. RFC 2131 (Draft Standard), Mar. 1997. Updated by RFCs 3396, 4361.

[13] D. Farinacci, V. Fuller, D. Oran, D. Mayer, and S. Brim. Locator/id separation protocol (lisp). Internet-draft, draft-farinacci-lisp-09 (work in progress).

[14] V. Fuller and T. Li. Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan. RFC 4632 (Best Current Practice), Aug. 2006.

[15] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau. Large-scale virtualization in the emulab network testbed. In *ATC'08: USENIX 2008 Annual Technical Conference*, pages 113–128, Berkeley, CA, USA, 2008. USENIX Association.

[16] R. Hinden and S. Deering. IP Version 6 Addressing Architecture. RFC 4291 (Draft Standard), Feb. 2006.

[17] E. Nordmark and M. Bagnulo. Shim6: Level 3 multihoming shim protocol for ipv6. Internet draft, draft-ietf-shim6-proto-12 (work in progress).

[18] C. Partridge, T. Mendez, and W. Milliken. Host Anycasting Service. RFC 1546 (Informational), Nov. 1993.

[19] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. In *Proceedings of the 1st Workshop on Hot Topics in Networks (HotNets–I)*, Princeton, New Jersey, October 2002.

[20] L. Peterson, S. Sevinc, S. Baker, T. Mack, R. Moran, and F. Ahmed. PlanetLab Implementation of the Slice-Based Facility Architecture. Technical Report http://svn.planet-lab.org/wiki/GeniWrapper#geniwrapper, February 2009.

[21] L. Peterson, S. Sevinc, J. Lepreau, R. Ricci, J. Wroclawski, T. Faber, S. Schwab, and S. Baker. Slice-Based Facility Architecture. Technical Report http://svn.planet-lab.org/attachment/wiki/GeniWrapper/sfa.pdf, February 2009.

[22] B. Quoitin, L. Iannone, C. de Launois, and O. Bonaventure. Evaluating the benefits of the locator/identifier separation. In *MobiArch '07: Proceedings of first ACM/IEEE international workshop on Mobility in the evolving internet architecture*, pages 1–6, New York, NY, USA, 2007. ACM.

[23] F. Sultan, K. Srinivasan, D. Iyer, and L. Iftode. Migratory tcp: Connection migration for service continuity in the internet. In *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 469, Washington, DC, USA, 2002. IEEE Computer Society.

[24] S. Thomson, T. Narten, and T. Jinmei. IPv6 Stateless Address Autoconfiguration. RFC 4862 (Draft Standard), Sept. 2007.

[25] X. Yang. Nira: a new internet routing architecture. In *FDNA '03: Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, pages 301–312, New York, NY, USA, 2003. ACM Press.