Project Description and Approach

1 Motivation, Background, and Summary

Virtual worlds are networked, simulated environments that bring geographically-diverse users together. They simulate physical interaction in three-dimensional spaces and decouple such interaction from geographic constraints. People participate in virtual worlds through their avatars, manipulating objects in the virtual world and interacting with other users. The virtual and physical (real) worlds may be coupled; objects in the virtual world can be linked to and can cause action in the physical world, while interfaces and sensors in the real world can control objects or provide data in the virtual world.

There are already multiple examples of successful, functioning virtual worlds. While hearkening to an earlier era that lacked the visual aspects of modern virtual worlds, the text-based MUDs of the 1980s and MOOs of the early 1990s demonstrated social and role-playing user interactions in online, real-time environments, where some users can create and program objects in the world. The first large-scale graphical systems were designed for military training [14], and the Army can now prepare a force for battle using a real-time networked three-dimensional simulation. Massive multi-user role-playing games like Blizzard's World of Warcraft, Sony's Everquest, and NCsoft's Lineage have millions of users and are commercial successes. Social virtual worlds like Second Life support the inclusion of user-generated content. Interest in Second Life has expanded from individual users to real-world organizations: universities and museums own regions of the world to explore science and education; corporations such as IBM and HP have created virtual office space to conduct business meetings, provide customer support, and support commerce.

Figure 1: Second Life, a popular virtual world environment. In January of 2008, 900,000 Second Life users logged in and spent 28 million hours online [13].

Virtual worlds have the potential to revolutionize the way that humans can interact with one another, with physical objects having some virtual presence and control, and with virtual objects that only exist in the virtual environment. Human interactions can cross national and cultural boundaries. New forms of education are possible, with students at different sites receiving instructions and engaging in impromptu interactions in virtual classrooms. Virtual worlds can benefit research, engineering, and businesses, enabling distributed groups to collaboratively share three-dimensional environments both synchronously and asynchronously. Virtual tools linked to physical devices can be operated remotely, both in commercial and home settings. The trend towards the networking of everyday devices—even something as mundane as air conditioners, security systems, or webcams—can lead to virtual presences that may be remotely and seamlessly accessed within virtual worlds. Indeed, humans place real value in virtual worlds and their virtual objects; an underground economy has evolved that supports the exchange of virtual-world objects to real-world money, including an exchange rate between virtual- and real-world currencies. Virtual worlds truly help entwine computer systems and networks into human life.

1.1 Towards federated virtual worlds

Yet for all their benefits and popularity, today's virtual worlds are characterized by centralized control and administration. All physical resources and administrative controls are managed by a single entity, *e.g.*, Blizzard for World of Warcraft or Linden Labs for Second Life. No interaction is possible between users in these walled garden virtual worlds, limiting the "network effect" of such technologies, and innovation is constrained by the system's central operators.

The success of the Internet as a disruptive technology—and all the subsequent technologies that use it as a communications platform—arguably arose from its ability to knock-down previously walled gardens by its embrace of interoperability and federation. The original ArpaNet's main task was connecting local networks running disjoint protocols. The World Wide Web allowed users to easily post, read, and organize

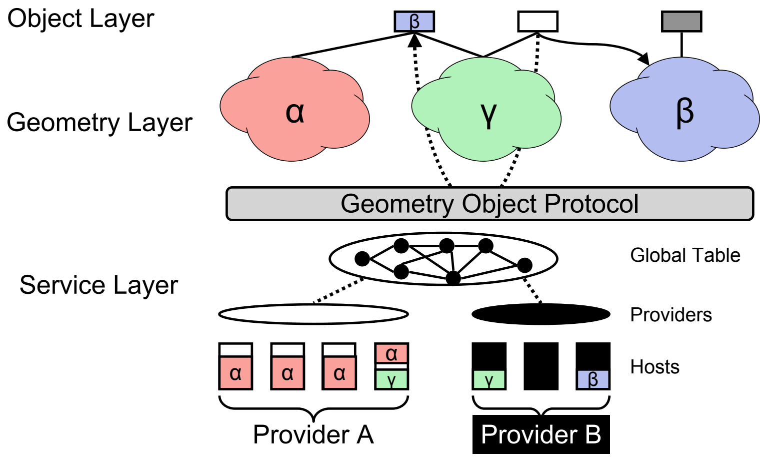

Figure 2: Overall networked system architecture.

content, linking content that may be physically hosted at distant servers. In the U.S. at least, the Internet finally became a wide-spread phenomenon when online portals such as CompuServe, Prodigy, and America Online finally provided access to the public Web, newsgroups, and IRC channels. More recently, the same phenomenon is happening with federating social networks (*e.g.*, through the OpenSocial directive [17]) and online authentication (*e.g.*, through OpenID [16]).

We believe that the same federation and interoperation that enabled the wide-spread growth of and innovation on the Internet is necessary for virtual worlds. Such a world could link together different managed environments, could enable end-users to host their own environments or use a platform provided by a service provider, and could provide a common programmatical environment for creating and developing virtual objects. In fact, just this past October, several providers—including Linden Labs (Second Life) and IBM—announced an intent to develop open standards for 3D virtual worlds [10]. This effort falls short of our goal: it serves mostly to provide a "single-sign-on" for the existing virtual worlds of today, rather than researching an infrastructure for the virtual worlds of tomorrow.

If virtual worlds were to become a dominant application platform, how might one design a network architecture to support them? This proposal addresses many of the most basic research questions around building global-scale, federated, seamless virtual worlds: What would the network stack look like? What interfaces would be exposed to a programming layer? How could this scale to huge numbers of virtual worlds, objects, and providers? What lower-layer services and infrastructure would be needed to support such a design? How could these support the secure migration of objects between providers? How could such a system prevent unwanted communication, as spam attempts would be inevitable given a loss of centralized control? To our knowledge, the goal of seamlessly federating virtual worlds is a new one, so the set of open research problems is both rich and varied.

1.2 P/NP: A network architecture for virtual worlds

This proposal seeks to answer our high-level research question by designing, implementing, and deploying such an architecture, named the Physical/Not-Physical (P/NP) architecture. The P/NP network architecture will focus on four important properties to scalably support large numbers of interacting virtual worlds: *expansibility*, *federation*, *migration*, and *security.*

The architecture must be able to support large numbers of virtual worlds with very different uses and administrative properties. At the same time, many service providers—from large commercial entities to single-server hosting sites—may contribute computational, networking, and storage resources to run these virtual worlds. To enable a seamless virtual universe rather than walled-garden networks, these different physical and administrative domains must be able to cooperate and interoperate while simultaneously maintaining internal control of their network operations. For such a network to provide ease-of-use, performance, availability, and reliability, objects and environments must not be physically tied to a single server or domain; users and objects must be able to move across worlds for seamless integration, with movements being controlled by a mix of provider and user requirements. At the same time, all involved parties—including users, application developers, world administrators, and service providers—need to be able to protect themselves from malicious entities and influence some system decisions and control.

The P/NP architecture is motivated by a small set of design principles. First, virtual world networks will be federated. Many independent service providers will contribute resources to host and run these environments. These service providers need to have flexibility in how they manage and maintain their own resources, while simultaneously having simple and clear mechanisms for cooperating and interacting with other providers. Second, application communication is grounded in three-dimensional coordinate spaces: objects can only communicate after being introduced through proximity. This geometric addressing decouples applications from their physical locations on hosts. Third, by using this communication model, the architecture can directly present and interface with the physical world.

The P/NP architecture, shown in Figure 2, uses a layered approach to achieve these principles. Each layer—geometry, object, and service—has its own research questions and challenges.

- The **geometry layer** forms the narrow communication waist of the architecture. The geometry layer is based on the observation that, in virtual worlds, objects need to interact mostly with other objects that are close by. It addresses how we should name and address objects across a universe of virtual worlds as well as how we can prevent unwanted interactions: objects may only communicate after they have been introduced through proximity in a world.
- The **service layer** sits below the geometry layer and implements its mechanisms. It addresses how to locate objects and route messages to them, how to enforce security and communication policies between objects, how to scalably distribute object execution across machines. how to support migration across hosts and providers, and how to support scalable content distribution for bandwidth-intensive virtual world content.
- The **object layer** sits above the geometry layer and constitutes the programming interfaces for virtual worlds. We are particularly interested in how one can support custom content creation by casual users.

2 Geometry: The Narrow Waist

This section describes the "narrow waist" of the P/NP architecture: the geometry layer, its mechanisms, and its communication abstractions. The key insight behind the geometry is that communication and naming in virtual worlds should follow their real-world analogues: proximity matters. More specifically, objects in discover what objects are around them by posing a geometric query to the virtual world they are currently in. Once they have discovered references to nearby objects, the two can communicate.

2.1 Why Geometry?

Making the basic networking primitives in the P/NP architecture explicitly geometric—at least from the perspective of the higher-level object layer—influences what application communication patterns will dominate. It also affects how communicating objects will organize themselves with respect to one another. For example, objects can remain private by remaining in private virtual worlds.

Furthermore, defining discovery in terms of geometry provides a clear and simple abstraction that can be implemented in many ways and optimized heavily. For applications and environments that are user-centric, one might optimize the networked system to respond with what objects are viewable. One can optimize queries exactly as they typically are in interactive systems, responding with closer objects first.

Take, in contrast, a more general querying system that operates on attribute/value pairs: query resolvers might have to understand value types, equality and inequality relations, and complexity would scale with the number of attributes. Geometric spaces provide a simple way to index values in terms of scalar values: queries on color, for example, can be geometric queries in a 3D space where object positions are based on their HSV or RGB values. Exploring the expressiveness and limitations of geometric networking is a key motivation for deploying the P/NP architecture to end users.

2.2 Virtual versus Physical Location

Naming, addressing, and lookup within the geometry layer are grounded in three-dimensional address spaces. Software objects are communication geometry object protocol endpoints and every object has an associated address space. This section describes the basic concepts of objects, references, and address spaces, as well as ownership chains, object zero (the physical world), logical object migration, bootstrapping, and the control plane. It also proposes a set of four simple communication abstractions that could support this functionality: packet delivery, querying, conditional communication, and control messages akin to ICMP.

The key principle behind the geometry design are that separating the location of an object within an address space from its location in the network allows the network to support federation, gives users control of distribution, can enact fine-grained, distributed access control, and can scale to support thousands of virtual worlds, with millions of users and billions of objects within each one. Furthermore, separating the network into many address spaces with geometric properties naturally lends to distributed and federated hosting. This enables the underlying service layer (Section 3), to migrate objects across physical resources.

2.3 Objects, Ownership, and Accounting

Before describing the communication primitives of the geometry, we must first describe geometry objects, the entities that communicate. Every object has a large, opaque, unique identifier. This identifier is by itself not sufficient for communication. An identifier merely provides a way to distinguish objects. We imagine identifiers being inexpensive to create and objects being objects in the software sense: they can have a wide range of functionality and capabilities.

All objects have an owner, the object that is responsible for it. Owners represent both administrative control as well as an accounting mechanism. If virtual worlds are a platform for full-fledged applications, there will be many active objects in worlds that act autonomously and without human interaction. Examples include objects that scan sales data to mine later, objects that display or animate visuals in the world, and objects that conduct financial transactions. All of these objects consume physical resources, such as CPU, networking bandwidth, or storage. Having a definition of ownership at the geometry provides a simple way to account for this consumption.

Ownership chains ultimately end in the identifier of the object economically responsible for the consumed resources: a service provider. For example, a data scanner might be owned by an object that coordinates several scanners, while this object is owned by a user's avatar, which is owned by the user's root object, which is owned by a service provider. Service providers with peering agreements can allow each other's objects to migrate onto and run on their infrastructure, with internal accounting and financial agreements. Service providers without peering agreements can still host objects, but cannot migrate them. For example, an end user who wants to run a vending object on his own computer can do so: by connecting a new machine to the network he beco .

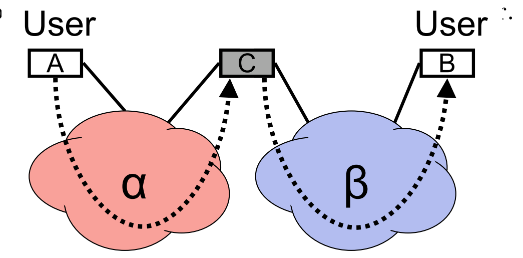

Figure 3: User A in address space α communicates with user B in address space β through B's communication object C (the grey object), which has a reference in both address spaces α and β. C forwards packets between A and B.

Only a service provider knows the binding between object identifiers and real-world people, in a manner similar to IP addresses today. However, unlike IP addresses, object identifiers are easily created and discarded. Correspondingly, a user who wishes anonymity can sign up with a service provider who promises such and frequently generate new top-level objects.

2.4 Address Spaces and Object References

An address space is a three-dimensional coordinate space. Objects themselves do not exist anywhere: instead, they maintain one or more *object references* in address spaces. An object reference R_X^A denotes that the object X is present in address space A. Address spaces are the ultimate authority on which objects have references in them, and where those references are, in terms of coordinates.

As Section 2.3 mentioned, object identifiers are not sufficient for communication. Instead, two objects can communicate if and only if they both share references in an address space. Practically speaking, this means data packets are sent and received in terms of references, rather than objects. For example, in Figure 3, objects A and B cannot communicate directly, as they exist in address spaces α and β respectively. Object C, however, can communicate with both A and B, as it has references in both address spaces.

An object can exist (have references) in multiple address spaces at once, but may only have one reference in any single address space. Existing in multiple address spaces allows communication to bridge address spaces. For example, a user can have a telephony object, which forwards audio streams. To let another user communicate through this object, the owner gives that user a reference to it. With this reference, the user can communicate with the telephony object, which then communicates with the owner through her own reference. Figure 3 shows this interaction; we discuss service-layer support (and optimizations) for such packet routing and communication enforcement in Section 3.

While a root identity represents a physical administrative domain consisting of hardware, address spaces represent logical administrative domains. An address space is, in essence, a self-contained virtual world. The owner of an address space can control who may enter it, and what goes on within it. The owner of an address space is responsible for resolving queries and being the final arbiter of object addresses: for virtual worlds, this is essentially deciding what objects can see and the physics model for their movement.

These responsibilities mean that address spaces are objects as well: they require computational resources. In the P/NP architecture, the two are synonymous. Every address space is an object, and every object is also an address space. This relationship will allow fine-grained administrative control. The house on a street in an address space is actually an object reference to the house object: stepping inside the house is a transition to a new address space, with possibly different security rules, or even rules of physics. An avatar carries a bag, which is an object: it is also an address space in which objects can be placed.

2.5 Querying

We have described objects, address spaces, and how the former communicate through references in the latter. Here we discuss querying, how objects can discover each other's references in an address space.

Fundamentally, queries within the P/NP architecture are geometric in nature. Figure 4 shows how objects can ask an address space, on behalf of a reference in that address space, what other objects are nearby. The address space can decide, based on its own policies, how to respond to the query. For example, it can scope the area the query can cover, the number of objects references returned, and which object references are returned.

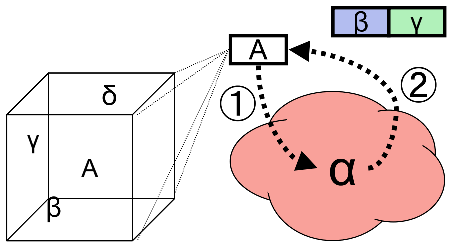

Figure 4: Object A in address space α queries α for what objects have references nearby (1). α responds with references for β and γ (2).

One early design question will be the form that these queries take, including numeric representation (fixed or floating point), coordinate system (Euclidean or polar), and centering (relative or absolute). Because these queries will be a significant computational load on hosts serving address spaces, and they will have a fundamental influence on the organization of objects and worlds, it will be one of the our critical early design decisions. One approach might be to allow all of them, and observe which are most useful in practice.

Designing the query resolvers themselves will itself be a significant research challenge. Work in the database community has deeply explored the problem of querying n-dimensional spaces, and has shown that as long as n is small, there are efficient querying structures (*e.g.*, R-Trees [9]). We plan to leverage this work, but with the understanding that the answers to these queries may inherently come from many servers, so the architecture will need to not only have efficient on-node query resolvers, but also fast resolvers for which nodes to query (*i.e.*, an anycast problem).

2.6 Object Zero

The P/NP architecture has one special, reserved object, object zero, whose identifier is 0. The address space of object zero is different than all other address spaces: its contents reflect the physical world. Placing the virtual representations of physical objects in object zero provides a simple and elegant way for users to query the physical world. The presence of such a physical address space is why we call our architecture P/NP architecture: it is the Physical/Not-Physical architecture.

Having a virtual world representation of the physical world has many uses and possibilities. This is Google Earth, except that there are active communication objects controlled by real-world entities. Real-world people can control their presence in this virtual reflection by whether their devices are connected to the network and advertise their presence. Owners of real-world establishments can establish an online presence by placing a store object, which allows objects to enter its address space. Of course, supporting such an object—which should seamlessly span the entire physical world—requires the design of very scalable address space, especially in how one can query for objects by three-dimensional locality. For example, when walking through the physical Times Square in New York, a query (say, originating from your physical cell phone) in the virtual world to find coupons advertised by nearby physical stores must be scoped to objects that are both physically and virtually nearby. Alternatively, "Times Square" might be a separate virtual address space that advertises a presence in object zero, but enforces its own admission control before users could enter its own address space—possibly based on physical locality—and subsequently issue queries for its virtual stores.

Unlike most other objects, object zero is governed by multiple administrative entities: the nations that constitute physical space. Therefore, while it might have a single logical administrative entity (the owner of object zero), this administrator may delegate responsibilities to the relevant governments. One can imagine that such a delegator might be a non-governmental organization, the virtual world equivalent of iCANN.

2.7 Persistent Communication

Limiting communication to objects within an address space might seem to be overly restrictive: it would mean, for example, that one object cannot easily communicate with another unless it gives it a reference to a communication object to own. However, address spaces are responsible for determining object reference addresses, and can evict them at any time. Objects in the world will need persistent and stable communication channels, and simply having an object reference nearby is not sufficient.

The fact that every object is also an address space saves us in this regard. A phone address book, for example, is an object in which a person stores references for voice communication objects. The person who owns the address book object controls its address space, so can arrange, add, and purge entries. In Figure 3, address space α could be such an object. Some people might arrange their address books as galleries, with paintings on the wall; others might arrange them as offices; others still might never view the address space, and just use the object as a communication channel. A conference call, or more complex communication sessions, can be address spaces in their own right, with entry controlling who may join.

The ease of creating objects and therefore new address spaces decentralizes controlling communication within virtual worlds. If an address space allows references that spam other references, or are otherwise noisome, then users will migrate to other address spaces that institute better admission policies. Alternatively, the address space can eject the problem reference. This richness and transience of possible communication channels creates an inherently competitive environment for designing good ones. For example, the administrator of an address space can decide what advertisements appear, can police appearances, and impose content-based restrictions. However, as anyone can create an address space, undue or insufficient exertion of that control will simply lead to objects leaving, or blocking communication from that address space.

2.8 Logical Migration and Bootstrapping

An object can only have a single reference within any given address space at any time. So far, we have talked about how references can be placed in address spaces, how address spaces control position (possibly based on object input), and how objects can use multiple references to bridge across address spaces. Here we describe how the network architecture might control how objects enter address spaces.

First, an object is the ultimate authority on what objects may have references in its address space, where those references reside, and the responses to their queries. For an object A to enter the address space of an object β, it must be able to communicate with β: both A and β must share references in the same address space γ. As an example, β is a virtual club: it has a reference on the street in the address space of a virtual city γ. A communicates with β, asking to create a reference within it. If β agrees, then A has a new reference within β's address space. Figure 5 shows such an exchange. Based on what sort of interaction is occurring, A might then destroy its original reference (it has moved into the club), or keep its original reference around.

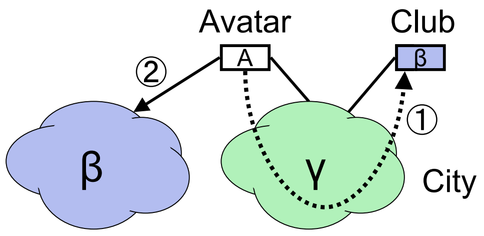

Figure 5: Avatar A enters address space β by sending a request to β's reference in address space γ (1). β approves the request and creates a reference for A (2).

As objects may only communicate through an address space, they can only ever communicate through introduction. For two objects A and B to communicate, they must both enter an address space, which acts as the introducer between the two. On one hand, communication by introduction is a powerful primitive for security. On the other, it raises basic problems in bootstrapping. The only way to get the object identifier for an address space β is to already be in an address space γ in which β has a reference: how does a brand new object enter its fist address space?

We imagine that this bootstrapping issue will be resolved much as the way in which it is in DNS today: there will be some set of well-known identifiers, describing giant hubs within the virtual world universe. Through an out-of-band mechanism, a user prompts one of these hubs to create a reference within an object she controls. This reference then allows her to communicate with the hub, and subsequently traverse into it and in turn be introduced to other objects. Much as Yahoo does with its portal pages, hub administrators have the power to control access, evict spammers and otherwise keep them as useful services.

These address spaces can apply policies for what objects they allow, either based on the objects' ownership or functionality. For example, an address space might restrict an objects ability to enter it unless its chain of ownership leads to a trusted provider (*e.g.*, trusted to use sufficient registration mechanisms to minimize spammers), limit the number of objects that may be owned by the same party (*e.g.*, to prevent denial-of-service attacks), or maintain blacklists based on an object's owner's prior behavior. These efficacy and expressibility of these security policies are bear further investigation.

2.9 The Communication Primitives

Distilling communication to object references within three-dimensional address spaces allows the P/NP architecture to define a simple narrow waist for the network. The architecture has four communication primitives: packet delivery, querying, conditional communication—which is a late-binding combination of the first two—and control messages akin to ICMP. Here, we describe each primitive in greater detail, sketching an idea of what fields these packets might require. Of course, as we develop and deploy the P/NP architecture, these definitions will change and evolve as we encounter unforeseen challenges and requirements: the descriptions here are solely intended to give a better sense of how we believe these primitives are quite simple in scope.

Packet Delivery: Section 2.4 stated that communication is always between object references within the same address space. Correspondingly, a packet between two object references has five fields: the address space object reference, the source object reference, the destination object reference, a protocol identifier, and the payload. The protocol identifier allows multiple higher-layer protocols to run on top of this unreliable datagram service. Just as transport protocols layer on top of IP, the network can support multiple transport protocols on top of basic packet delivery between virtual objects. We imagine a simple UDP-like transport will be one of these protocols, as will querying, described below.

Querying: Section 2.5 stated that querying is defined in terms of geometric coordinates. We leave it open as to whether such coordinates are Euclidean or polar, fixed or floating point, absolute or relative. Each of these will have different coordinate specifications, but we can generalize the packet format. A query packet's source is the querying object, and its destination is the address space itself: the address space and destination fields are the same. Its protocol identifier is the query protocol identifier, and its payload define the geometric space the query covers.

The response to a query has the same protocol identifier, but its source, rather than the destination, is the address space. Its payload contains a list of object references within that space. The references in the reply are a subset of those actually within the geometric region: an address space can choose to not include covered objects, but cannot add objects that the region does not cover. This ability to elide objects allows an address space to establish limits and policies on queries. For example, objects may constrain the geometric area that a query can cover in order to prevent denial-of-service attacks by huge queries, or use such functionality to implement visibility constraints given the virtual environment in the address space.

Conditional Communication: The third communication abstraction is a combination of the prior two: it enables objects to communicate with other, nearby objects without knowing their identities. This primitive is critical for basic virtual world operations such as shouting. Semantically, a conditional packet is a late-binding query that delivers a packet to references that satisfy the query. Without this primitive, objects must continuously query the address space and maintain identifier lists based on this query. Whether groupcasts involve state in the address space (*i.e.*, are long-running, rather than one-shot) is an open question we will need to examine. On one hand, allowing long-running groupcast operations will simplify applications; on the other, the resource requirements they place on the address space object may be large. Clearly, if allowed, some form of accounting for their cost might be warranted. A conditional packet is essentially a query with a payload.

Control Plane: In addition to the above three data abstractions, the P/NP architecture will have a control plane, with the equivalent to ICMP. One example of a piece of necessary control functionality will be querying an object for its ownership. Another is a request to enter an address space. As we develop the P/NP architecture and deploy applications, we expect that the control plane will evolve beyond these obvious two to include needed mechanisms, such as "object unreachable" errors due to network failures.

One primitive notably absent from the above list is a mechanism for an object to obtain the coordinates of another object, or even its own coordinates. While an address space answers queries in terms of geometric coordinates, actual communication of those coordinates occur above this protocol layer: they are data packets. We separate these two concerns for simplicity. Just as IP routing layers run on top of IP, we imagine coordinate communication will run on top of a coordinate lookup system.

We leave many of these higher-level abstractions to the object layer, the system that runs on top of the network architecture. If designed correctly, we can keep the two entirely separate and therefore able to evolve and advance separately. The computation and maintenance of coordinates, for example, is not tied to any of the mechanisms above.

2.10 Overview

This section has described network abstractions and primitives in the P/NP architecture. It described packets, their formats, and their semantics. These comprise the basic communication fabric of the architecture. The next section describes the service layer, which implements these abstractions. The service layer maintains address spaces, routes packets, and migrates objects while providing mechanisms for fine-grained accounting, inter-provider peering, and communication security.

3 The Service Layer

Objects, object references, and address spaces comprise a logical communication layer defined by geometry. The geometry layer ultimately runs on physical machines, connected through a physical network such as the Internet. The service layer implements this system substrate, providing five mechanisms: resolving objects to service providers, routing messages to objects, distributing objects across hosts, migrating objects between hosts, and enforcing communication policies from the geometry and object layers. We visit each in turn.

3.1 Resolving Messages to Providers

When an object sends a message to another object, the service layer needs to deliver the corresponding packet(s) to a host. Figure 6 shows how a data communication at the geometry layer appears at the service layer. Migration, however, means that the host or hosts an object runs on can change due to load-balancing, performance, or fault-tolerance considerations. Therefore, the service layer separates message delivery to an object into four steps. The first step resolves an object to one or more service providers. The second step is delivering the packet to a provider. The third step, conducted by the service provider, resolves the object to a specific host; the final step involves actually sending the message to this host. This section describes the process of the first step and second steps; the next section describes the latter two. Of course, this separation points out that the system is actually recursively performing the same two steps—resolution and routing—at increasingly fine granularity.

Viewed abstractly, object resolution creates a large global table that maps an object identifier to its current provider. This table may be implemented in a variety of ways, for example by partitioning the actual set of identifiers across multiple "root" providers (*e.g.*, via consistent hashing) or by replicating the complete table across a small set of highly-provisioned providers (as in the DNS root servers).

Upon receiving a message to an object reference, a server performs a *lookup* to translate this reference to a provider, fill in message headers with its destination address, and *route* the message accordingly. This separation allows providers to migrate objects without having to propagate state to other providers, giving them flexibility in resolution table maintenance and internal migration policies.

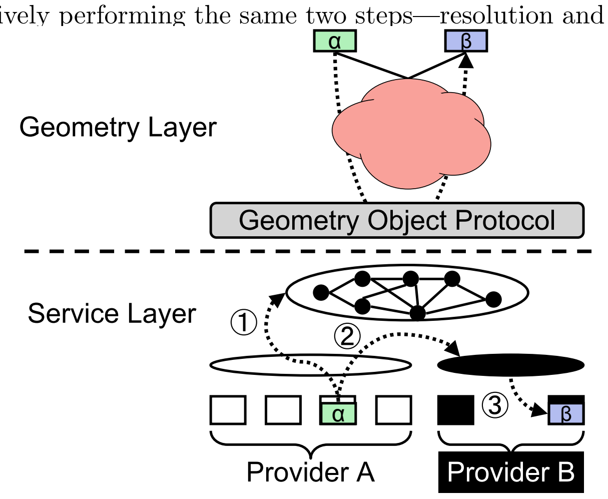

Figure 6: Object α sends a packet to object β at the geometry layer. At the service layer, this causes a lookup for a provider of β (1), α sending the packet to a gateway of Provider B's network (2), and B resolving and routing the packet to the internal host that β resides on (3).

Message delivery performance and the *lookup* service load would be a significant concern if the service layer had to resolve object identifiers on every packet. Instead, name-to-provider bindings can be cached in routing tables, much like TTLs are used in DNS resolution. However, the P/NP architecture need not suffer the same problems as cached DNS results when the binding is no longer valid—*e.g.*, because the object migrated or the destination is not reachable—because object identifiers are explicitly included in messages. When message routing fails, the control layer response will cause the P/NP network stack to invalidate its object resolution cache entry and perform a new *lookup*.

This is not possible with DNS, both because the network layer has no notion of names, only IP addresses, and because DNS resolvers are both logically and often physically separate from routing tables. But is feasible in the P/NP architecture, because messages contain both the intended object identifier endpoint as well as the service provider identifier. The equivalent would be if each IP packet contained the intended DNS name (*e.g.*, for content-aware routing [8]). Data communication constantly verifies routing tables.

After the object resolution phase returns the service provider P hosting an object, the service layer needs to route packets towards this destination P. Exactly how this routing works depends on how the P/NP architecture is deployed. For example, if the P/NP architecture is deployed directly over IP, then P may be one of a set of IP addresses that represent clusters or data centers, much as how large web sites today operate. This IP address would represent a gateway into the provider's network. In contrast, if P/NP architecture is deployed using an overlay on top of IP, then P would be an overlay identifier. Using an overlay could separate P/NP architecture from the underlying IP layer, provide instances of better performance and reliability than directly using IP routing, as demonstrated by Akamai's SureRoute [1] and RON [2].

3.2 Resolving Messages to Hosts

One a message reaches the provider that hosts an object, the provider must deliver it to one of the hosts the object resides on. A global table maps identifiers to providers; each provider then has a table that maps identifiers to hosts. Because this inner table is hidden from external hosts, providers have a great deal of flexibility in how they implement, distribute, and maintain it, especially in the face of object migration, and host addresses can be internal and remain hidden.

If providers are spread across multiple physical data centers that are geographically diverse, messages may enter their logical domain at various ingress points, much like border routers in inter-domain IP routing. A provider's internal routing needs to handle this behavior. Another level of hierarchy might be useful, first routing a message to a cluster, before finding the appropriate server. Such separation minimizes the amount of wide-area state that must be updated when objects are only migrated locally, *e.g.*, for load balancing.

3.3 Identifiers versus References

As an object creates a distinct reference for each address space that it occupies, the service layer must be able to easily map a reference to an object, and objects must be able to easily maintain, revoke, and control their references. Because routing is in terms of objects, rather than references, the *lookup* service only needs to maintain a single entry per object. This keeps routing state independent of the number of object references and enables objects to quickly and cheaply create or destroy references.

Messages in the geometry and service layers carry full object references, rather than object identifiers: we discuss some security reasons for including full object references later in §3.7. Therefore, overly large references would introduce overhead on all communication. We imagine references to initially be 128 bits. The top 96 bits identify the object and the bottom 32 bits identify the reference. Following a lesson from UTF-8, the first bit of all 128-bit references is 0; if a need for a larger namespace arises, a value of 1 can indicate a longer reference. Packet exchanges that are independent of the reference can simply ignore the reference bits.

3.4 Generating Identifiers and Updating Resolution

While much of this section has focused on object resolution, we have yet to discuss how these object-to-provider mappings can be set at the lookup service, both when an object is initially created and upon any subsequent migration. A secure, easy-to-use, and coherent *update* protocol is a very important aspect of P/NP architecture's service layer and perhaps its greatest research challenge. The *update* protocol should replace an object identifier's old provider with its new one; multi-homed objects will likely use separate commands to add and remove providers from an object identifier's set.

Simply leaving the *update* protocol undefined and outside the scope of the service layer is unsatisfactory. Such a situation exists with domain name registration and management with domain registrars, and it has led to registrar-specific mechanisms, mostly implemented via bewildering web pages and lacking machine-accessible APIs. Fully federating virtual worlds will require standardized update protocols.

We imagine global table maintenance having a single operation, *update*. Using *update* on an identifier that is not in the table inserts it. An *update* has three parameters: the object identifier, the service provider(s) hosting the object, and the identifier of the economically-responsible entity. The economically-responsible entity is typically a provider, or someone who has an existing financial relationship with the maintainers of the global table. This can either be a direct relationship (a personal account) or a proxy relationship (*e.g.*, your service provider). The former case allows users to host small virtual worlds on their own, while the latter case would likely be more common. Changing ownership of an object simply requires the current economic provider to send a message updating who is responsible for it; this message is itself signed by the newly responsible party.

The update message contains an economic identifier because, ultimately, introducing entries has a cost. Update messages are signed by their economically-responsible entities. Of course, actual update messages, when implemented, are likely to have additional fields and mechanisms, to protect from replay attacks, etc.

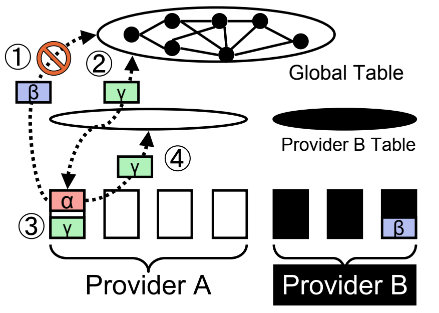

Figure 7: Object α tries to create a new object with identifier β by sending an update message, which the global table rejects due to an incorrect signature, as object β already exists at Provider B (1). α tries a new identifier, γ, which succeeds (2). Provider A creates the new object γ (3), and updates its local table to have a routing entry for γ.

These update semantics make it simple to generate a new object identifier: all one needs is a random number that is distinct from the owner's existing identifiers. Since the owner will need to maintain what identifiers it is responsible for regardless (*e.g.*, to prevent false billing), checking is a simple O(1) lookup. If the new identifier is already in use by another owner, then the update will fail due to an incorrect signature. Figure 7 shows such an example.

One other potential approach is to leverage self-certifying names [15], by extending object identifiers to include a fingerprint of its owner's public key. Then, an *update* request can be signed by this public key, and the lookup service can directly verify the *update* yet avoid any particular public key infrastructure. At the same time, self-certifying objects raise as many questions as they answer: Should each object have a separate key, or should users maintain a small number of keys for all their objects? Would the increased length of object identifiers pose scalability concerns? If we require signed updates, how can a provider migrate an object? Requiring a user to be online and approve each migration is infeasible in practice, yet might not a object's provider need access to the object's private key? Should we trust some providers to manage an object, but somehow only give them oracle access to a signing key hosted elsewhere? Managing updates in a secure manner should prove a rich area for research.

3.5 Multi-homing and Static Content

Our discussion so far has mostly dealt with objects that have a single service provider hosting the object. While multi-homed objects will significantly complicate consistency issues, our network substrate is at least amenable to objects concurrently residing at multiple locations. There are several ways to support multi-homing: *lookup* can return a set of providers managing an object, or simply one of these providers, much like anycast. If only one provider is returned, either the *lookup* process itself might have some intelligence in its selection of provider (*e.g.*, based on topological location), or it could just return a random provider and require that a subsequent request to the object learn a more desirable object instance if desired, *e.g.*, for persistent communication. Given that object-to-provider mappings may be cached in the network, this recursive (and potentially expensive) selection of an object instance need only occur once. Figure 8 shows two example multi-homing operations.

Multi-homing is also critical for address space management. Large and popular address space might have millions of object references that they must interact with and maintain. Many virtual worlds, such as Second Life, Asheron's Call, and World of Warcraft, provide the appearance of a continuous region of space, while actually being distributed across many different servers. Other virtual worlds, such as Eve and Dungeons and Dragons Online, have explicit boundaries (between solar systems and small geographic regions, respectively). Being able to scalably support the continuous model requires many references, maintained by different servers, to reside within a single address space.

Multi-homed objects may prove especially useful for *distributing static content*, as we can represent content just as any other object in our system. While most of this proposal has focused on dynamic objects in virtual worlds, content distribution remains an important service for virtual worlds. This is especially true when virtual environments includes dynamic content generation; one of Meru's strengths is its support for easy yet powerful user content creation. Indeed, if user objects include large, static components, one may choose to represent these as separate objects to simplify their dissemination to other objects in the system. For example, user-generated clothing may have relatively few dynamic properties, but may still require significant static data for its graphics rendering on a user's machine.

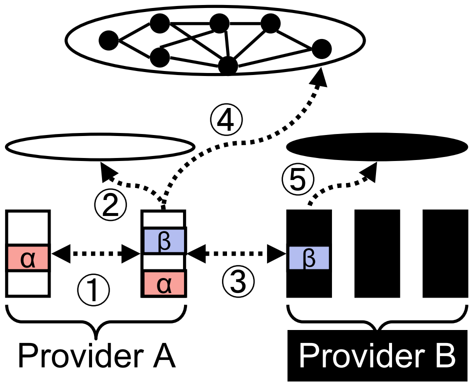

Figure 8: Provider A multi-homes two objects, α and β. It multi-homes α internally, by requesting another server to host it (1), then updating its provider table (2). It requests another provider to host β (3) and updates the global table (4). Provider B updates its provider table for β (5).

Object multi-homing places challenging requirements on the object layer. Objects running on multiple hosts must have state that can be distributed easily and avoid global variables. In the case of an address space, for example, the data structure storing references needs to be easily divisible, such that geometrically close references are likely to be on the same host. Global, single-value variables cannot be easily multi-homed, as they require consistency protocols between multiple copies, or message passing. One way to address this challenge is to encourage programming that separates per-interaction state from global state. An ATM object, for example, might be multi-homed, in that each user's transaction-in-progress is independent. As more objects interact with the ATM, it can multi-home to support the load. Of course, the ATM must also enact a transaction on a bank account, whose state cannot be as easily multi-homed. Exactly what such a programming model might look like is a very open question; it is part of our synergistic work on the Meru architecture.

3.6 Object Migration

Communication-by-introduction at the geometry layer makes physical object migration a critical requirement of the service layer. Simple communication between two objects may pass through several intermediaries that bridge address spaces. If each object in the chain requires a cross-continent packet transmission, latency will suffer terribly. Therefore, co-locating tightly-coupled objects on the same host is a key requirement for network-level efficiency. Thankfully, many of these intermediary relationships, such as the telephone example in Figure 3, can be reasonably static. Still, there will be many dynamic relationships, and thus the service layer needs to be able to migrate objects across hosts.

Load balancing is a second reason why the service layer must support object migration. As regions of the world become popular or more active, the service layer needs to be able to adapt its distribution of resources to maintain responsivity and performance.

At first, object migration might seem to be a sore point for system performance: process migration is well understood problem in operating systems, and typically is quite expensive. However, two factors can come to our rescue. First, the goal is to migrate software objects, many of which are lightweight scripts, rather than full-featured operating system execution contexts. Second, multi-homing simplifies object migration. Migrating an object from host A to host B involves multi-homing it on A and B then deleting the copy on A. It is more complex, of course, when the object being migrated cannot be easily multi-homed. But good solutions to the multi-homing problem will automatically improve migration efficiency.

3.7 Enforcing communication security

While we have shown how objects can communicate, we have yet to show how the service layer can *prevent* them from communicating, in order to restrict communication to objects within the same address space. The P/NP architecture is agnostic how such communication policies are actually enforced: filtering can be done at the sender, destination, or along the message's path, performed at the application or network layer.

At a high-level, the goal of our policy enforcement is related to Internet denial-of-service protection, for which there is a rich body of related work, from overlay-based filtering [3, 12], pushback of traffic filters into the network [4, 11], and capabilities to control network traffic to destinations [18, 19]. Much like some of the capability-based designs, our service layer seeks to enforce *default-off* communication [5, 7], where two objects can only communication if there exists an address space for which the sender and recipient both reside. However, this policy introduces a different twist on security enforcement, in that these policies are not set by the destination (as is with the DoS-limitation literature), but rather by the address space's state.

One simple design is a form of *waypoint* routing: a sender object S can first route its packet to the address space A in which communication is occurring ; this address space will subsequently delivers packets to the destination object D iff $\{S, D\} \in A$. On the destination side, the object will only accept packets from address spaces to which it belongs—but because messages are addressed with full object *references*, as opposed to object *identifiers*, a destination can simply check if the specified destination reference indeed exists in the specified address space. If it has no knowledge of membership in that address space, the packet is discarded (*e.g.*, as potential spam). Knowledge of address space membership can be pushed down from the geometry layer to the server layer at an object's provider, in order to perform fast filtering, *e.g.*, leveraging flow entries in hardware to restrict communication from approved sources [6].

One major open question is how much an address space needs to intercede in communication between objects, whether packets between objects in the same address space can avoid going physically through the address space's provider. One could envision leveraging ideas from network capabilities here, with the address space providing the control channel to establish direct communication, which avoids routing to the address space object as an intermediate waypoint. Once one party leaves an address space, this communication channel should be broken.

4 Summary

This proposal focuses on the next stage of interoperability and federation for network services: supporting virtual worlds that scale to world-wide numbers of service providers and users, as well as the objects they create. To our knowledge, our proposed work on the P/NP architecture is unique in its goal, and our planned work focuses on four important properties to scalably support large numbers of interacting virtual worlds: federation, migration, expansibility, and security.

Building such a virtual world ecosystem will entail providing a system layering model, much like one exists in basic network communication, that defines abstractions and interfaces between each layer. Ongoing research by the PIs on Meru, a distributed virtual world system, will provide the object layer and application driver for the proposed research. This proposal specifies the vision of federated worlds, and defines the geometry and service layers necessary to support them. Using the P/NP architecture, users will be able to build many different Meru-based virtual worlds which objects can interoperate across and migrate between.