**Abstract**

Networked Virtual Environments (NVEs) which are based on Peer-to-Peer technology aim to provide large-scale virtual worlds without bottlenecks coming from centralized instances. Current approaches focus on massively multiplayer online games, but little work has been done on designing NVEs that enable trading goods and services for real money. In this article, the specific threats on such networked virtual marketplace environments (NVMEs) are discussed. Consecutively, requirements for the underlying network structure are derived to avoid them. Finally, an efficient lookup protocol for persistent objects in NVMEs is presented, which can be set on top of distributed hashtables and allows scalable virtual environments of unlimited size.

**Keywords**: Distributed Systems, Networked Virtual Marketplace Environments, Peer-to-Peer

**1. Introduction**

Seamless 3D networked virtual environments (NVEs), in which users can interact with virtual objects that provide amongst others the functionality of today’s websites and web-applications, hold a remarkable potential to transfer today’s World Wide Web into an intuitively browsable World-Wide-SecondReality. To gain the necessary acceptance of both users and content providers, single points of failure and control as in client-server based architectures like e.g. SecondLife must be avoided. Hence, such an infrastructure must be built on an arbitrary and dynamically changing number of autonomous nodes that are operated by the respective content- and application providers, as it is in the current WWW. The resulting structure could grow and change dynamically without any centralized control instances.

With only few exceptions (e.g. [1]), current approaches to build such networked virtual environments in a peer-to-peer manner are based on an implicit agreement of trust and therefore neglect the existence of situations, where a single participant (peer) could gain personal benefit from disturbing the coherence or connectivity of the virtual environment. Such situations can especially occur, when the NVE in question provides capabilities to trade goods or services, be they virtual or real, in exchange for real money. Thus, the underlying structure of such trade-enabling NVEs, which can be denoted as networked virtual marketplace environments (NVMEs), must inherently avoid this kind of tempting situations.

However, in contrast to client-server architectures, the distributed character of P2P approaches inevitably requires a minimum amount of cooperation and trust between the participating peers to maintain connectivity and consistency of the virtual world. Since no single peer can maintain an omniscient consistent state of the whole virtual environment, a conflict of trust evolves, coercing every participant to rely on information given by potential competitors.

In addition to objects that purely serve as representation of contents (e.g. documents), NVMEs also consist of objects that represent virtual or real services or products, which may be subject to a charge. For a user who is willing to use such objects, the owner (provider) of such objects, and therefore the contracting party, must always be identifiable. Current NVE approaches either do not provide an object ownership model at all or dynamically change object ownership according to load-balancing or structure-management needs, which makes such approaches inappropriate for using them in NVMEs.
This article is thus dedicated to the question, how NVMEs can be built based on an implicit agreement of suspicion instead of an agreement of trust, allowing a user to crosscheck information given by single peers. After a short overview of related work, Section 3 therefore addresses challenges that specifically NVMEs have to meet and derives requirements on the underlying structure, showing that existing NVE approaches do not fulfill these requirements without undergoing substantial changes. Hence, in the forth section, a lookup protocol for persistent objects in NVMEs is proposed, which sufficiently reduces the necessary amount of trust between the participating peers. The last section concludes this article and gives an outlook on future work.

2. Related Work

In general, each approach for building NVEs in a P2P manner has to provide (i) an object lookup protocol, which allows querying for objects that are located at the user’s current position, and (ii) an update propagation mechanism to notify users on object state changes without flooding the underlying communication network. In almost all current peer-to-peer based NVEs, these problems are solved by partitioning the virtual world and therefore distribute lookup and management load over the participating peers. This segmentation can either be based on the virtual map (quasi-static segmentation) or on the participating avatar’s positions (dynamic segmentation).

Before the approaches are described in more detail, a short introduction to distributed hashtables is given, which play a major role for some of the existing approaches as well as for the concept discussed in this article.

2.1. Distributed Hashtables (DHTs)

In the context of NVEs, DHTs are in some cases used as substrate, on which lookup structures for objects or peers are built. They provide the functionality of hashtables, that is they can store and retrieve $(key, value)$ pairs, but they do it in a distributed, i.e. a peer-to-peer manner. Every peer in the DHT can retrieve the value associated with a given key either using its own local database, if that peer manages the respective value, or by forwarding the request to an appropriate neighbor. Commonly known examples for DHTs are the Content-Addressable-Network (CAN) [28] and Chord [33]. CAN provides a $d$-dimensional key space on a $d$-torus (see Figure 1a), i.e. the available key-range in each dimension is configurable but fixed and can therefore be normalized to a range of $[0..1]$. In Chord, the ID of each peer represents the lowest key value this peer manages. All peers are arranged in a circle, in which each peer maintains links to the peers with the next higher IDs. The size of the ring is configurable, but also limited and cannot be changed at runtime. Additionally to their direct neighbors, Chord peers store the addresses of more distant neighbors in a so-called finger table to improve the lookup performance (see Figure 1b).

![Figure 1. Example DHTs: a) CAN, b) Chord](source: [28] source: [33])

To retrieve an object for a given key, both CAN and Chord forward the request to the neighbor, which manages the keys closest to the requested key. This procedure is repeated, until the query reaches the peer that manages the requested key, which can now lookup the according value in its local database. Applications, which want to query the DHT, need an entry point to the structure and must
therefore know the address of at least one arbitrary peer that is part of the DHT. Often, links to several peers are maintained by the user to provide a higher level of fault tolerance.

2.2. Quasi-static Segmentation

Early examples of quasi-static segmentations are *MinMud* [4] and *MOPAR* [5], where the virtual world is divided into an arbitrary number of segments, each managed by one or more coordinators. Objects entering and leaving a segment are registered at the respective coordinator, which is also given notice when state changes of any object within the segment occur. The coordinator in turn multicasts state updates to all peers within the segment. If the segment size is well above the size of an avatar's area-of-interest (AoI), it is usually referred to as a *region* (as e.g. in *MinMud*), whereas smaller segments are called *cells* (as in *MOPAR*).

Because the location of segments in the virtual world does not change, the segment coordinator(s) can be seen as anchor points within the virtual world. DHTs can be used for coordinator assignment, so that a coordinator for a region is not necessarily located within that region (as e.g. in *MinMud* and the *Zoned Federation of Game Servers* [2]).

In contrast to its location, the size of a segment is not necessarily fixed. Instead, with changing numbers of managing peers, segments can be splitted and merged (see *LOADER* [6], *GROUP* [7], as well as Varvello et. al. [8]). With each split operation, some of the objects need to be transferred to the newly added coordinator.

In pure peer-to-peer approaches, the coordinators of neighboring segments are connected and can therefore provide a handover, when an object moves between two segments. In contrast, the *Zoned Federation of Game Servers* uses a central server, which manages such inter-region movements, so that region coordinators are not interconnected.

To avoid communication bottlenecks, the number of coordinators for a segment can be increased. For instance, in *MOPAR*, one master coordinator keeps an authoritative copy of the managed object, but in addition, several slave coordinators keep replicas of the object in weak consistency with the master. In *MinMud*, the replication serves only as backup, in case a master node unexpectedly disconnects from the network.

2.3. Dynamic Segmentation

The most established approach for avatar-oriented segmentation is the *Voronoi-based Overlay Network (VON)* as described in [9-12]. It dynamically creates Voronoi-tessellations around the position of each user's avatar and so defines a neighborhood relationship between all avatars as well as a dynamic partitioning for distributing the management load. In [14], Cavagna et. al. propose to use a hierarchical model for 3D objects to efficiently adapt to the level of detail needed to sufficiently display a peer's current view. The underlying partition scheme is *VON*-based.

The Delaunay triangulation, which corresponds to the dual graph of the Voronoi-diagram, is also used for dynamic partitioning, as described in [15], [1] and [16]. Voronoi and Delaunay triangulation could in principle be used for quasi-static segmentation, as e.g. proposed by the relaunched *Sollipsis* project [17-19], but they inherently produce a higher management overhead than quasi-static approaches.

In *pSense* [20], each peer defines a circular vision range. All peers within that range (near nodes) as well as the closest peers just outside the circle (sensor nodes) are tracked and receive state update messages from the current peer. When the peer moves, the sensor nodes can give information about other peers coming into the field of view. A similar concept is described in [3], where each peer is connected to its closest neighbors.

2.4. Segmentation-free Approaches

When the underlying distributed data structure allows efficient range-queries, i.e. queries that refer to a range of values instead of only a single one, the lookup process can be distributed without the need to artificially partition the virtual world. *NL-DHT* [21] provides a range-queriable DHT using Hilbert-
curves of adaptive order. The Colyseus NVE [22] can be built upon such range-queriable DHTs for object lookup.

The HyperVerse project [23-27] proposes a two-tier architecture, which consists of a structured CAN-like backbone of public servers (GP3 [29]) and a loosely-coupled, torrent-like client overlay. Although, the HyperVerse approach seems similar to our proposal at first glance, it does not provide an unlimited virtual world. Additionally, the backbone uses quasi-static segmentation as described in Subsection 2.1 above.

In Donnybrook [30], frequent state updates are only received for those objects which are currently important for a user, the so-called interest-set. State updates for all other objects are only sent infrequently to enable large-scale virtual worlds.

3. Challenges for NVMEs

In this section we address challenges that NVMEs have to meet in addition to NVEs and derive requirements on the underlying structure. Two scenarios are discussed which especially arise from the implicit agreement of trust in P2P-NVEs. Those are (i) the neighborhood lookup attack, where a malicious peer disturbs the coherence of the virtual map, and (ii) the object state manipulation attack, where peers disturb the consistency of object states. Finally, a third requirement is derived from scalability issues.

3.1. Neighborhood Lookup Attacks

When a user navigates to a specific coordinate in the virtual environment, the user’s NVE browser first of all needs to find out, which peer holds the data that is needed to display the surroundings at that coordinate properly. Therefore, each NVE approach necessarily has to provide a lookup protocol that maps coordinates to peer IDs. For performance reasons such protocols usually take into account, whether the queried location is directly neighbored to the user’s current position, or if it is an unrelated query. An unrelated query needs to be answered either by the underlying DHT or by forwarding the query hop-by-hop until the target location is reached, which leads to time and message complexities in the range from $O(\log N)$ (tree-based, Chord) to $O(\sqrt{N})$ (map-based, CAN). In contrast, the peer responsible for a neighboring coordinate, as for example $(x+1,y)$, can be queried directly from the already established connection to the peer responsible for the user’s current coordinate. Hence, the complexity for such a neighborhood lookup is $O(N)$. In some current approaches, as e.g. in VAST/VON, it is not even possible to place an unrelated query at all, because they provide no other infrastructure than the neighborhood relationship.

![Figure 2](image.png)

Figure 2. Sequence diagrams for a) regular neighborhood lookup, and b) honeypot scenario with malicious peer $P_{x,y}$

The downside of such neighborhood lookup mechanisms is that a user has to trust each peer giving correct information about other peers in its neighborhood. If a user wants to move from his current position to a neighboring coordinate, a handover from the current peer to the peer responsible for that
neighboring coordinate is necessary (see Figure 2a). However, in NVMEs, the current peer might for example be tempted to restrain the user from leaving its territory by giving false information about which peer is the responsible neighbor. In this honeyspot scenario (Figure 2b), the peer would make a handover to either itself or another peer under its control. Alternatively, the attacking peer could deny the existence of a rivaling neighbor with the objective of keeping customers from entering the competitor’s territory.

In map-oriented DHTs like CAN, the vulnerability for neighborhood lookup attacks is not limited to related lookups, but can affect all lookups which the malicious peer either should forward to one of its neighbors or answer itself. At worst, an attacking peer could isolate a complete zone, not only single peers, depending on its position in the routing network. The use of multiple CAN-realities, in which each peer is assigned to varying positions, attenuates the vulnerability at the cost of higher maintenance load.

Of course, every lookup structure can be disturbed by malicious participants. Nevertheless, a structure, in which each peer most likely is responsible for routing anonymous requests for anonymous peers, does inherently provide fewer incentives for malicious behavior and additionally makes it more difficult to attack a specific target (e.g. a competitor). Hence, the lookup structure of NVMEs as well as the lookup messages should not allow any inferences on the structure of the virtual world. Furthermore, an independent lookup structure provides more alternative routing paths to cross-check information given by a single peer. This leads to the first requirement on NVMEs:

Requirement 1: Maintain a lookup structure which is independent from the structure of the virtual environment.

3.2. Object State Manipulation Attacks

The general purpose of NVEs is to allow users to interact with virtual objects, like avatars of other users or objects that represent a specific service or application. Interactions with objects may lead to state changes, which in turn must be propagated to all users that might somehow be affected by them. Therefore, each object must be assigned to at least one peer that holds information about its current state and provides an interface to interact with it, if necessary. In most current NVE approaches, the peer assigned to the object is not necessarily the object’s owner or creator, but is selected by the underlying P2P system instead, e.g. depending on the object’s (current) position. When either the object’s position or the segmentation of the underlying map changes, it may be transferred to another peer, which is often denoted as delegate or coordinator. For load balancing purposes, it is also possible to create replicas of an object on multiple delegates, who then have to synchronize object state changes among each other to ensure consistency.

Usually, the object owner has no influence on the delegation process and therefore has no control over the availability of the object. This is a major issue for providers, whose business success strongly depends on the availability of their services. Furthermore, since the interaction with the provider’s object shall result in an exchange of real money for real or virtual goods, liability issues can arise, when a delegate shows malicious or unexpected behavior and so affects either users or providers or both.

While dynamic segmentation approaches (see Section 2.3) can in principle be designed to do without delegation and therefore let an object’s state be handled solely by its owner, approaches with static segmentation inherently have to use some kind of delegation mechanism. Those approaches are therefore always prone to attacks, in which a delegate peer manipulates an object’s state or availability. By using e.g. byzantine tolerant protocols with several delegates per object, executing a successful attack can be made more difficult, but not impossible. While such a residual risk still can be acceptable in online gaming environments, it is not tolerable in situations where object state changes result in an obligation to pay real money.

Nevertheless, delegation and/or replication to trusted peers should still be possible, but should be initiated by and kept under the control of the object owner, who should have the only authoritative copy of an object. From this, the second requirement on NVME structures is derived:
**Requirement 2:** Let object owners be authoritative, and leave delegation decisions to them.

With this requirement fulfilled, an object owner no longer has to implicitly trust anonymous and potentially malicious peers. Furthermore, additional business models like “rent-a-peer” are possible, in which a peer offers his computing and network resources for rent to overloaded providers, who then can delegate some of their object management load.

### 3.3. Scalability

Three scalability domains are of interest in networked virtual environments: (i) the number of objects and users, (ii), the number of state updates per time or per peer, and (iii) the size of the virtual world. The first and second domains have already been widely addressed by existing approaches. However, the size of the virtual world in static segmentation approaches is often limited due to the need for a globally known map in online gaming environments. However, in NVMEs, a limited map size automatically results in an artificial shortage of available virtual space. Hence, the number of providers is limited unnecessarily, which particularly influences the structure of applications and contents that are provided. Only scalable and therefore unlimited virtual worlds are truly open to arbitrary and dynamically changing providers, contents, and users. Therefore, the third, somewhat plain requirement for NVMEs reads as follows:

**Requirement 3:** Do not limit the size of the virtual environment.

### 3.4. Suitability of Existing Approaches for Use in NVMEs

Most existing approaches mentioned in Section 2 have in common that they aim to provide scalable virtual gaming environments. Therefore, it is clear that they do not necessarily fulfill the requirements for virtual marketplace environments, as Table 1 shows. Often, a limited and globally known size of the virtual world is expected (e.g. in MinMud, Colyseus, and HyperVerse). Half of the approaches do not provide any mechanism of authoritative ownership of virtual objects.

The concepts based on dynamic segmentation, i.e. those that use the NVE neighborhood to provide lookup functionality, are in the same way inherently prone to neighborhood lookup attacks as described in Section 3.1 as all static segmentation approaches which are not based on DHTs.

### Table 1. Comparing Properties of current NVE projects

<table>
<thead>
<tr>
<th>Project</th>
<th>Connectivity &amp; Lookup</th>
<th>Authoritative Owners</th>
<th>Scalable Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinMud</td>
<td>DHT</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>ZFGS a)</td>
<td>DHT &amp; neighborhood</td>
<td>yes</td>
<td>not specified</td>
</tr>
<tr>
<td>Colyseus</td>
<td>DHT</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>MOPAR</td>
<td>DHT &amp; neighborhood</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>pSense</td>
<td>neighborhood</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>KMA(2002) b)</td>
<td>neighborhood</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>LOADER</td>
<td>neighborhood</td>
<td>yes</td>
<td>not specified</td>
</tr>
<tr>
<td>HyperVerse</td>
<td>DHT &amp; neighborhood</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Sollipsis</td>
<td>neighborhood</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>VAST / VON</td>
<td>neighborhood</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Donnybrook</td>
<td>neighborhood</td>
<td>yes</td>
<td>not specified</td>
</tr>
</tbody>
</table>

a) Zoned Federation of Game Servers [2]; b) Work of Kawahara, Morikawa and Aoyama [3]

Even when DHTs are used, they either only serve to assign coordinators or delegate peers to resources or they provide application-level multicasts. However, none of the concepts uses DHTs to deliberately detach neighborhood in the NVE from neighborhood in the underlying lookup structure and still preserve $O(1)$ complexity for neighborhood lookups. Although the NL-DHT approach seems to solve this problem at first glance, taking a closer look reveals that it still preserves neighborhood-relationships within the proposed hash structures and therefore is also still prone to neighborhood...
lookup attacks. Summing up, none of the existing approaches to built NVEs fulfills all requirements for NVMEs as discussed in this section.

4. Persistent Object Lookup in P2Life

The P2Life project [31], the name being derived from P2P and SecondLife, aims to provide unlimited NVMEs without relying on centralized instances. In this section, we describe the P2Life lookup protocol for finding the peer managing persistent virtual objects on a given coordinate. Persistent objects are defined by their characteristic to be permanently visible in the virtual worlds and not to depend on a user to be online. The P2Life structure and the according lookup protocol are designed to fulfill all requirements for NVMEs mentioned in Section 3.

4.1. P2Life Basic Structure

The P2Life virtual world is based on a two dimensional infinite lattice, where each square, which we denote as parcel, can be uniquely identified by its coordinate and is either assigned to a provider or empty. Only the one peer, who owns a parcel, can place persistent virtual objects on it. The owner is responsible for providing information that is necessary for a user to render the persistent objects on the parcel. Additionally, the owner handles all interaction between his objects and the avatars himself or by designated delegates. A peer can occupy several contiguous parcels to build larger virtual buildings or coherent areas. Parcels which are not in use can be rendered as landscape by a user’s NVME browser.

The assignment of parcels to peers should be free of costs and controlled by the NVME community. Additionally, the system should provide mechanisms to revoke parcel ownership if a peer shows malicious behavior. Both the assignment- and the revoking strategy are not part of this article.

As discussed in Section 3.1, a lookup protocol is needed allowing a user to find out, which peer manages the objects that reside on a given parcel. For this purpose, all peers that own at least one parcel, i.e. all providers, form a lookup structure which is independent from the layout and structure of the virtual world. Each provider’s peer therefore acts both as NVME-peer, giving information about objects on a specific parcel, and as lookup-peer, handling lookup queries. Furthermore, the peers of all users that currently explore P2Life form another independent structure to manage lookups and state updates of non-persistent moving objects, like avatars. This additional lookup structure is not discussed in this article.

Figure 3. Unlimited NVE and underlying DHT (gray)

4.2. Lookup protocol for persistent objects

The proposed lookup protocol is based on a CAN as underlying structure (see Figure 3). In principle, any two-dimensional DHT that allows subdividing the hash-space into an arbitrary number of segments
can be used. For a higher degree of fault tolerance, multiple independent DHTs can be used, as e.g. in form of CAN realities. In each reality $i$, a unique and globally known coordinate-to-key mapping function generates key-pairs $(k_x, k_y)$ from coordinates $(x, y)$ with $0 \leq k_x, k_y \leq 1$. Thus, each key consists of $d$ decimal places, where $d$ can be changed dynamically and depends on the number of peers in the CAN and therefore on the number of providers in the NVME. Although the length of a key is variable, it still is limited and hence each key represents an unlimited set of coordinates. The position of a key-pair in the CAN is not related to the coordinate in the NVME, from which it was generated. Additionally, the keys for neighbored coordinates in the NVME usually will not be neighbored in the CAN. Thus, the structure of the NVME bears no reference to the structure of the lookup network.

The lookup structure is able to handle three types of messages, which serve the purposes of (i) registration, (ii) unrelated lookups and (iii) related lookups. When a provider obtains a parcel from the community, its NVME-peer $P_{NVME}$ initiates a registration process in the underlying CAN. The key-pair for the assigned coordinate as well as the NVME-peer’s (IP-) address are forwarded to the lookup-peer $P_{KEY}$ that is responsible for the respective key-pair. $P_{KEY}$ now queries the exact coordinate from $P_{NVME}$ and stores both the coordinate and the NVME-peer’s address in a local database. The unregistration process works the same way.

Similar to a registration query, an unrelated lookup query consists of the key-pair generated from the enquired coordinate and the (IP-) address of the querying peer $P_{QUERY}$. Again, the query is forwarded to the lookup-peer $P_{KEY}$ that is responsible for the key-pair in the CAN. $P_{KEY}$ queries the exact coordinate from $P_{QUERY}$ and looks it up locally. When an entry for the requested coordinate exists in $P_{KEY}$’s database, it returns the address of the assigned NVME-peer. Otherwise, the cell is empty and $P_{KEY}$ returns NIL. This way, no memory resources are required to manage empty parcels, of which each key-pair represents an unlimited number. Since the number of lookup-peers scales with the number of occupied parcels, the required amount of memory per peer is independent from the size of the virtual world.

To provide related queries, a newly added lookup-peer executes an unrelated query for those keys that represent neighboring NVME coordinates of the key-pair’s coordinates, which the peer is responsible for. Now, users have to execute an unrelated query only once to find the lookup-peer responsible for their current coordinate. All subsequent queries necessary to allow moving to neighboring parcels can directly be handled by that peer with $O(1)$ message and time complexity.

Because each key-pair represents an unlimited number of coordinates, and each lookup-peer can only store a limited number of neighbored keys, the functionality of this approach mainly relies on the coordinate-to-key mapping to preserve collisions of neighboring coordinates. This means that, if any arbitrary two coordinates $(x_1, y_1)$ and $(x_2, y_2)$ map onto the same key-pair $k_1$, their neighboring coordinates $(x_1+1, y_1)$ and $(x_2+1, y_2)$ are also represented by an identical key-pair $k_2$. This way, the lookup peer only has to store either 4 (von-Neumann neighborhood) or 8 (Moore neighborhood) addresses for each key-pair.

**Figure 4.** a) Neighbored coordinates in the NVME (left) and b) disjoint paths for the according related and unrelated lookups in a CAN

At first glance, this approach seems to have the same vulnerabilities as a static-segmentation approach. A malicious peer $P_M$ in the CAN could still try to disturb the coherence of the NVME by giving false information about its neighbors. But the main difference deriving from the key generation is that the user can check the correctness of the information at any time by performing an unrelated
query. Figure 4 illustrates that in contrast to approaches without key generation, the unrelated query, which is usually started from an arbitrary lookup-peer, will most likely not be routed through $P_M$, because as stated in requirement 1 for NVMEs, neighbored coordinates usually do not map to neighbored key-pairs. When a user’s peer randomly crosschecks the results from related and unrelated queries, malicious behavior of lookup-peers will most likely be detected.

Additionally, by using keys instead of coordinates, any lookup-peer forwarding a query to the peer managing the respective key-pair cannot directly determine the NVME coordinate from which the key-pair was generated. However, a malicious peer could pretend to be the lookup-peer responsible for the key-pair and then query the coordinate from the source of the request. In this case, the source peer still can crosscheck this information by querying the lookup peer managing the key-pair for the neighbored coordinate with an unrelated query, which will most likely not be routed through the malicious peer. From that lookup peer responsible for the neighbored position’s key-pair, a related query for the original coordinate will easily reveal the deceit. Again, this crosscheck needs not necessarily to be done for each single query, it is instead sufficient to do it in some random intervals.

4.3. Requirements for the Coordinate-to-Key Mapping Function

The central component of the proposed lookup structure for peers which manage persistent objects undoubtedly is the coordinate-to-key mapping, which we denote as $h(x,y,d)$. In this section, the requirements for this function are summarized using the symbols described in Table 2.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r \in \mathbb{N}^+$</td>
<td>The number of realities in the CAN</td>
</tr>
<tr>
<td>$i: 1 \leq i \leq r$</td>
<td>Index of an arbitrary reality in the CAN</td>
</tr>
<tr>
<td>$x, y \in \mathbb{Z}$</td>
<td>Coordinate of a parcel</td>
</tr>
<tr>
<td>$b \in \mathbb{N}^+, b \geq 2$</td>
<td>Number of fractional digits that the key-generation algorithm at least must compute</td>
</tr>
<tr>
<td>$d \in \mathbb{N}^+$</td>
<td>The coordinate-to-key mapping that assigns an arbitrary NVE coordinate $(x,y)$ to a key-pair (e.g. a position in the CAN) with $d$ digits</td>
</tr>
</tbody>
</table>

$h_i(x,y,d) \rightarrow [0,1] \times [0,1]$  

Table 2. Symbols to describe requirements for the coordinate-to-key mapping

The first two requirements for $h_i$ are derived from requirement 1 for NVMEs, which claims that the underlying lookup structure should not reflect the structure of the NVME itself. Therefore, $h_i$ shall not allow to identify the position of an NVME-peer in the virtual world through the position of the according key-pair in the CAN. Additionally, when two CAN realities $i$ and $j$ are used, the structure of $i$ should not be reflected in $j$ and therefore, the mappings $h_i$ and $h_j$ should be sufficiently different. This leads to following requirements:

**Requirement 1 (invertibility):** $h_i$ must not be invertible

**Requirement 2 (distinct realities):** $h_i(x,y,d) \neq h_j(x,y,d)$ for any $j \neq i, 1 \leq i, j \leq r$

The number of digits necessary to identify a lookup-peer depends on the granularity and therefore on the total number of peers in the CAN, which in turn depends on the number of providers in the NVME. The number of digits used to identify a lookup-peer must not have any influence on the structure of the CAN. So, if two keys as $k_1 = h_i(x,y,d_1)$ and $k_2 = h_i(x,y,d_2)$ are generated for the same coordinate with $d_1 \leq d_2$, $k_2$ must always completely contain $k_1$ at the first $d_1$ digits:

**Requirement 3 (granularity):** $h_i(x,y,d) = h_i(x,y,d-1) + a_d(x,y,d,h_i(x,y,d-1)) \cdot b^{-d}$
Finally, \( h \) must preserve collisions for neighboring coordinates. The related lookup described in Section 4.2 strongly depends on this property. This way, the necessary number of neighbors a lookup-peer has to store per key-pair, is constant.

**Requirement 4 (collision preservation):** Let \((x_1, y_1)\) and \((x_2, y_2)\) be any colliding coordinates for any given number of digits \(d\), so that

\[
h_1(x_1, y_1, d_i) = h_1(x_2, y_2, d_i)
\]

Furthermore, let \((x_1 + \Delta x, y_1 + \Delta y)\) and \((x_2 + \Delta x, y_2 + \Delta y)\) be neighboring coordinates of \((x_1, y_1)\) and \((x_1, y_1)\) with \(\Delta x, \Delta y \in [-1,0] \land (\Delta x, \Delta y) \neq (0,0)\). Then, the neighboring coordinates also must collide, i.e. they must map on the same key-pair, too:

\[
h_1(x_1 + \Delta x, y_1 + \Delta y, d_i) = h_1(x_2 + \Delta x, y_2 + \Delta y, d_i)
\]

Of course, this does not imply that the key-pairs for original and neighboring coordinates must be equal. Additionally, two coordinates colliding for a specific number of digits \(d\), do not necessarily collide for a higher number of digits, which therefore also applies for the neighboring coordinate.

### 4.4. Example Coordinate-to-key Mapping

To round off this article, an example coordinate-to-key mapping is presented, which serves as proof that the requirements discussed in Section 4.3 can be easily fulfilled. As a matter of simplicity, only the mapping for one-dimensional coordinates is discussed, from which the two-dimensional case can directly be derived. The proposed mapping is in principle based on a modulo operation in \(b^d\), which additionally maps each digit of the coordinate to another value. In a number system on base \(b\), where a coordinate is denoted as \(x = \sum_j a_j \cdot b^j\) with \(a_j \in \mathbb{N} \land 0 \leq a_j < b\), the key \(k = h(x,d)\) with \(d\) digits is generated as follows:

\[
k = h_{i,0}(a_0) \cdot b^{-1} + h_{i,1}(a_1) \cdot b^{-2} + \ldots + h_{i,d-1}(a_{d-1}) \cdot b^{-d}
\]

Because the NVME uses unlimited coordinates, while the CAN is based on a normalized coordinate space of \([0..1] \times [0..1]\), \(x\) is processed starting at the least significant digit. The value at the \(j^{th}\) place before the decimal point in \(x\) is mapped to the \(j^{th}\) place after the decimal point in \(k\) by using a “step-by-step” mapping \(h_{i,j}\). Table 3 shows precomputed mappings for the first 4 places in a hexadecimal coordinate-system \((b = 16)\) in CAN-reality \(i = 1\).

<table>
<thead>
<tr>
<th>(j)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>(h_{i,0}(j))</td>
<td>7</td>
<td>F</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>A</td>
<td>3</td>
<td>B</td>
<td>4</td>
<td>C</td>
<td>5</td>
<td>D</td>
<td>6</td>
</tr>
<tr>
<td>(h_{i,1}(j))</td>
<td>8</td>
<td>0</td>
<td>9</td>
<td>1</td>
<td>A</td>
<td>2</td>
<td>B</td>
<td>3</td>
<td>C</td>
<td>4</td>
<td>D</td>
<td>5</td>
<td>6</td>
<td>E</td>
<td>7</td>
</tr>
<tr>
<td>(h_{i,2}(j))</td>
<td>5</td>
<td>D</td>
<td>4</td>
<td>C</td>
<td>7</td>
<td>0</td>
<td>8</td>
<td>1</td>
<td>9</td>
<td>B</td>
<td>2</td>
<td>A</td>
<td>F</td>
<td>6</td>
<td>E</td>
</tr>
<tr>
<td>(h_{i,3}(j))</td>
<td>E</td>
<td>7</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>A</td>
<td>3</td>
<td>D</td>
<td>5</td>
<td>C</td>
<td>4</td>
<td>B</td>
<td>0</td>
<td>8</td>
<td>F</td>
</tr>
</tbody>
</table>

...  

For example, the NVME-coordinates \(x_1 = 1BF16\) and \(x_2 = A01BF16\) are both mapped to the following key with \(d = 4\) decimal digits:

\[
h_1(1BF_{16},4) = h_{i,0}(F) \cdot 16^{-1} + h_{i,1}(B) \cdot 16^{-2} + h_{i,2}(1) \cdot 16^{-3} + h_{i,3}(0) \cdot 16^{-4}
\]

\[
= 0.E5DE_{16} = h_1(A01BF_{16},4)
\]
The mappings in Table 3 have been selected in such a way that neighbored values for a single digit are distributed uniformly over the target space. For example, in each mapping, each possible value occurs exactly once. Additionally, the number of occurrences where a digit is mapped to a close-by value (e.g. \( B \mapsto C \) in \( h_{i,j}(j) \)) is similar or the same as the number of mappings to more distant values (e.g. \( 2 \mapsto 8 \) in the same mapping). This way, the distribution of keys in the CAN does not allow any inference on the structure of the NVME. To store the pre-defined mappings \( h_{i,j} \), 4 bytes of memory per digit are needed on each peer. The number of digits necessary to identify a peer in the CAN has \( O(\log N) \) complexity. So, using 32 pre-defined mappings would be sufficient to provide as many distinguishable keys as IPv6 addresses exist. If more mappings are needed during runtime, a distributed leader-election algorithm like e.g. [32] can be executed to dynamically agree on additional mappings and therefore maintain scalability.

It is obvious that the proposed coordinate to key mapping is not invertible and therefore fulfills the first requirement in Section 4.3. The second requirement (distinct realities) can easily be fulfilled for a feasible number of realities by either using different place-wise mappings or by rotating the existing mappings by one or more digits. The third requirement (granularity) is also fulfilled, because each digit is processed independently. Finally, the fourth requirement (collision preservation) is fulfilled, because two coordinates mapping on the same key in \( d \) necessarily must be equal in their \( d \) least significant digits and therefore also their neighbored coordinates have to.

5. Conclusion and Future Work

In this article, we have introduced networked virtual marketplace environments as a subset of virtual environments based on the peer-to-peer paradigm. We have pointed out that in contrast to gaming environments, the necessary amount of trust among the participating NVME peers to maintain a consistent virtual world, should be reduced as much as possible. From this general statement, the specific requirements of NVMEs have been analyzed and were checked against current NVE approaches. We have shown that the existing approaches for NVEs cannot be used to provide NVMEs, at least not without profound structural changes. Therefore, an NVME has been proposed that detaches the lookup structure from the structure of the virtual world and so allows a user to crosscheck structural information given by single peers. Additionally, the proposed structure allows building NVMEs of unlimited size. Because the functionality of the proposed concept strongly bases on specific properties of the coordinate to key mapping used, we have discussed both the requirements on the mapping as well as an example mapping fulfilling them.

Although NVMEs do not primarily aim to provide avatar-to-avatar interaction, still a scalable structure is needed to manage lookup and state-updates of non-persistent and mobile objects as avatars. Additionally, the system should be able to build a dynamically changing global map of the current state of the NVME to give users an orientation of what contents are available and where to reach them. This map could also contain automatically generated objects like roads, which represent areas of the NVME which were frequently visited by users in the past. Finally, both a prototypical web-server and browser plugin shall be developed, which allows combining surfing the current WWW as well as P2Life virtual worlds.

6. References


[31] P2Life project homepage, online at http://www.fernuni-hagen.de/kn/en/P2Life_EN.shtml
