Electronic Communications of the EASST
Volume 17 (2009)

Workshops der
Wissenschaftlichen Konferenz
Kommunikation in Verteilten Systemen 2009
(WowKiVS 2009)

Fighting Cheating in P2P-based MMVEs with Disjoint Path Routing

Sebastian Schuster, Arno Wacker and Torben Weis

12 pages

Guest Editors: M. Wagner, D. Hogrefe, K. Geihs, K. David
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Fighting Cheating in P2P-based MMVEs with Disjoint Path Routing

Sebastian Schuster, Arno Wacker and Torben Weis

University of Duisburg-Essen

Abstract: In a P2P-based Massively Multiuser Virtual Environment (MMVE) where nodes cannot be trusted, replicating data on multiple nodes is a possibility to increase the reliability to obtain correct data. Current structured P2P networks mostly place replicas in such a way that queries for the replicas travel along similar paths. A malicious node in the common part of all paths can nullify the security gain of replicated data. We therefore propose to combine radix-based prefix routing with a symmetric replication scheme to gain disjoint paths to each of the replicas.

Keywords: MMVE, Distributed Hash Table, Peer-to-Peer, Routing, Cheating

1 Introduction

Massively Multiplayer Online Role-Playing Games (MMORPGs) are probably the most popular type of MMVE today. They allow players to be play a role in a virtual fantasy world, fighting monsters in a party with other players, solving epic quests, advancing their characters to higher levels and acquiring virtual wealth. Blizzards World of Warcraft [Bli05] is the most prominent example of this kind of MMVE, having more than nine million users worldwide as of 2008 [Woo08].

When a user has left the virtual world, his progress must be stored persistently, so he can continue his adventures next time he joins the world. This applies to information about player characters, their levels, equipment, inventory and so on. This information must always be available, as players must be able join the game with their avatar whenever they want. In client/server-based MMORPGs, this information is stored on the server.

On the other hand, client/server-based MMORPGs have inherent scalability limitations and require expensive operation of powerful servers. Realizing MMORPGs based on peer-to-peer networks might overcome these problems. In a P2P-based MMORPG, data has to be stored in a distributed fashion. Well-known structured P2P overlays like Chord [SMK+01], Pastry [RD01] or Tapestry [ZHS+04] implement a distributed hash table (DHT) allowing this kind of storage. Using their built-in replication mechanisms might deliver the high reliability we need for this kind of information.

However, there is a serious threat to all MMORPGs that must be dealt with: Cheating. These games are highly competitive environments. Character advancement, equipment and accumulation of virtual wealth play a major role as they can be used to demonstrate the social status in the world and the skills of the player. Players have to invest a lot of time and effort to reach a high status. Consequently, there will always be some players trying to advance without obeying the rules of the game. If honest players think they are competing with cheaters, they will simply quit the game and look for a better one.

Modifying persistent character data is a straight-forward type of cheat. If cheaters could modify that data, they could easily advance to the highest level or get hold of some very rare and valuable equipment. In a well-designed client/server-based MMVE, this is impossible because persistent data is stored on the server and only modified by the server according to the game rules. In a P2P-based MMVE with distributed storage, mechanisms to select and authorize nodes for storing data are necessary. Clearly, storing this data only locally on the player's computer is not a good idea.

Again, replication might be useful. Storing r replicated copies of persistent data in the network allows to collect multiple replicas and increases chances to obtain a correct result. However, replication schemes that distribute replicas over the neighbourhood or leaf-set of the original node, like Chord or Pastry, have a decisive shortcoming. Queries and replies for the original data or its replicas travel along similar paths to a large extent. Thus, an attacker only has to attain a position in the common part of the paths to prevent any correct reply from reaching its destination.

We will show that a replication scheme where replicas are evenly distributed in the keyspace in combination with radix-based prefix routing can guarantee r disjoint paths to the r replicas. This ensures that an attacker needs more than $r/2$ nodes at specific positions in the keyspace to successfully corrupt data. Furthermore, we will sketch an approach to fight cheating with a distributed voting mechanism based on this scheme.

The paper is structured as follows. First, we give some background information about structured P2P networks and replication in those networks. Then, we discuss attacks and security considerations in these networks. Afterwards, we describe how a voting-based mechanism can be used to fight cheating. Then, we present our approach to guarantee r disjoint paths necessary for the voting mechanism to work. Finally, we offer a short conclusion and some thoughts about future work.

2 Background

2.1 Structured P2P Networks

A structured P2P network consists of a large number of nodes. Each node has a unique identifier from some identifier space, for example the unsigned numbers of a given length. Furthermore, each node has an address that can be used to send messages to that node (e.g. IP address). Each node only knows identifiers and addresses of a subset of all the nodes in the network. This information is stored in a local routing table. When a node sends a message to a node with a certain ID, it first checks wether this node is in its local routing table. If this is the case, it sends the message directly to the address of the target node. If not, the message will be routed indirectly by forwarding it to a node from the routing table which is closer to the target node. Usually, with numeric identifiers, the numerically closest node is chosen as the next hop.

To implement a distributed hash table on top of such a network, we need a function mapping data items to nodes of the network. These nodes are responsible for storing the data and answering queries for that data. Usually, the keys of data items and the IDs of nodes are simply taken from the same space and each node is responsible for numerically close data items. Using hash functions for calculating keys of data items (e.g. from a unique name) will result in a uniform distribution of data items across the keyspace. Also using a hash function to compute the IDs of nodes (e.g. from IP addresses) will also result in a uniform distribution of the nodes across the same keyspace. By mapping data keys to close nodes, this scheme will on average yield an equal load on nodes (data items to store per node).

Assuming some data has already been stored under a certain key in such a network, querying for that data works as follows. A node issues a query with the key of the data. This query is routed through multiple nodes. Each node has a local routing table. It uses this table to forward the query towards the node responsible for that key, that will finally answer the query.

A malicious node does not have to follow the sketched protocol. It can misroute the message, it can modify the message or simply drop it. However, existing structured networks in their basic scheme generally assume their nodes to behave well. This will most likely not be the case in a P2P-based MMORPG.

2.2 Replication in Structured P2P Networks

Existing structured networks can ensure reliable storage despite failing nodes. They store r replicated copies of the data on multiple nodes of the network. As nodes join or leave, the networks tries to maintain the replication factor r. When a node fails and the replication factor of some data falls below r, data is automatically copied from nodes still holding replicas. As long as not all r nodes fail simultaneously, data will always be available.

Replicas are usually placed in the immediate neighbourhood of the original responsible node (e.g. in Pastry, Chord). When a query is routed towards this node and it fails, the query will arrive at a neighbour unable to forward the query. But since the neighbour is holding a replica, it can answer the query and will automatically become the new responsible node. While this simple scheme ensures availability of data without any modification to the routing mechanism, it is highly vulnerable to attacks by malicious nodes.

Only one malicious node in the routing path is enough to prevent a query from completing successfully. Even if there is a mechanism to retrieve different replicas, the standard routing mechanism will always generate similar paths when all replicas are in the same area of the keyspace.

2.3 Security Considerations in Structured P2P Overlays

First structured P2P networks generally assumed their nodes to behave well and showed reasonable performance under these circumstances. However, in our setting they also have to perform well in the presence of attackers. An attacker is someone controlling a certain number of nodes behaving maliciously. Malicious nodes do not follow the protocol of the network. Furthermore, they may work together in a coordinated fashion at the discretion of the attacker. According to [SM02] there are three main types of attacks on structured P2P overlays. The first are *routing attacks*, where malicious nodes forward messages incorrectly, drop them or try to corrupt the routing tables of honest nodes by sending wrong routing updates. The second are *storage and retrieval attacks*, where nodes deny the responsibility for storing data assigned to them. The third type subsumes all other kind of attacks, including denial of service attacks [DKK+05], eclipse [SCDR04] or sybil attacks [Dou02].

To prevent routing attacks, the authors of [CDG+02] identify three building blocks: secure node ID assignment, secure routing table maintenance and secure message forwarding. Since node IDs are used for positioning nodes in the routing overlay, a node able to choose its own ID can easily take over control of a selected part of the identifier space with all the data stored there. If an attacker controls multiple nodes he can place them in a way that all routing entries of a victim node point to his malicious nodes. Access to the network of that node is then completely controlled by the attacker, who does not have to control a large fraction of nodes.

Node ID assignment must therefore be verifiable by all nodes, so a node cannot claim any position in the keyspace. Hashing of IP addresses is a straight-forward solution, assuming nodes cannot choose their IP addresses freely. However, the authors of [CDG+02] also noted that IP addresses are only a weak source for IDs. Often, they are assigned dynamically via DHCP or by Internet providers and an attacker can simply wait till he gets an address that hashes to the desired part of the keyspace. With the introduction of IPv6, an attacker will have much more addresses to choose from. The authors argue that certified identities assigned by an external certification authority (CA) are the only viable solution. Thus, every node can check the identity of every other node using the public key of the CA while the CA does not have to be present all the time. At the same time, confidentiality and integrity can be ensured using the public key infrastructure.

Making sure nodes cannot choose their identities is not enough to ensure secure operation of the network. It is also necessary to make sure that malicious nodes do not acquire a large number of entries in the routing tables of honest nodes. In the ideal case the fraction of entries pointing to malicious nodes ("‘bad entries"’) should not exceed the fraction of malicious nodes in the whole network. However, since nodes usually update one another with their routing information on a regular basis, bad entries might be propagated to other nodes and get a disproportionate share over time. According to [CDG+02] a solution to this problem is placing constraints on the node IDs that may show up in a certain place in the routing table. If only keys from a certain key range are allowed in a specific place, it is hard for an attacker to have a malicious node present in this place, since he cannot choose his node ID.

Despite securing the network structure by protecting the routing table through constraints and preventing nodes from choosing their positions by secure ID assignment, malicious nodes may still have a disproportionate influence on network operation. One malicious node in a routing path is always enough to prevent successful message delivery.

Nodes are supposed to forward messages according to some routing scheme. In all schemes, messages are supposed to get closer to the target node in each routing step. If nodes were able to observe how routing of their messages progresses they could check wether a node obeys the routing rules and reroute if necessary [SM02]. If a node violates the rules e.g. by forwarding the message into the wrong direction or by not forwarding the message at all, the original node can ask the node before the suspicious node for an alternative node to forward the message to.

They key question is how to ensure that redundant routes do not have many nodes in common to actually have a security benefit. Cyclone [ALG05] introduced the notion of independent paths, having no intermediate node in common. However, the destination node might still be malicious and can launch a storage and retrieval attack. Multiple replicas with paths to the replicas being completely disjoint would solve that problem.

With replicas placed in the neighbourhood of the responsible node, redundant routes will always converge at the responsible node and it can successfully prevent forwarding queries to the replicas. Therefore, some schemes have been developed distributing replicas in the keyspace in a uniform way. With dynamic replication [LDH06], the ID of the object will be hashed together with the number of the replica. Thus, each replica can be queried individually while the hash function ensures that replicates will be distributed evenly across the keyspace. However, there are no guarantees that routes will be disjoint in the end. Multiple replicas can even end up on the same node. Using a different hash function for each replica as proposed for CAN and Tapestry has similar properties.

The authors of [GAH05] proposed a scheme called symmetric replication. In this scheme, they define an equivalence relation on IDs with each class containing as many IDs as the desired replication factor. Then, all nodes responsible for one ID are also responsible for all equivalent IDs. If equivalent IDs are uniformly distributed across the keyspace replicas will also be distributed uniformly. This can easily be achieved by using congruence classes modulo N/r, N being the number of IDs and r being the desired replication factor. This way, equivalent IDs will be distributed with a distance of N/r. The authors argue this scheme can be used to improve security because a majority voting can be done. It can also be used to decrease lookup latencies, as queries for different replicas can be executed concurrently and the first answer arriving can be used. However, the authors did not consider the routing of messages and that paths should be disjoint for higher security.

The potential for using such a replication scheme to generate disjoint paths has been investigated for Chord in [HB06]. The authors call their scheme equally-spaced replication, which is essentially the same as symmetric replication with congruence classes modulo N/r [GAH05]. They were able to prove that equally-spaced replication with a replication factor of r yields $\log_2 r$ disjoint paths in Chord. Our work is a direct improvement on this, yielding r disjoint paths when using radix-based prefix routing and symmetric replication.

3 Basic Principles for Secure Storage

We assume all nodes to have a unique identity certified by some trusted certification authority. Public key cryptography with generated session keys ensures confidentiality and integrity of communication. Nodes in the message path cannot read the messages and modifications of messages will be detected.

However, the key problem for storing persistent information in a pure P2P network is the lack of trust. Having certified identities does not mean that we can trust the nodes, they may still behave maliciously. It only means that we can detect misbehaviour, identify the originator and punish him (e.g. by exclusion from the network). For this to work, obtaining an identity must be expensive – otherwise the same user may join with a different identity next time. The best way would be to have a one-time idenity e.g. by securely binding the online identity to the real world identity. This way, one-time misbehaviour could lead to all-time exclusion from the game. However, usually new identities can be obtained by paying money forming a moderate barrier for attackers.

We envision using the established secure identities to mitigate the lack of trust in any individual node by trusting a majority of nodes. Computations will be performed redundantly, nodes will check each other and results will be stored according to the votes of multiple nodes. We assume that at least a majority of nodes is honest and follows our protocol. After all, if there is a coalition of a majority of nodes they could change the game rules in any way they wish.

As long as our assumption holds, a successful attacker needs multiple expensive identities taking part in the decision process and forming a majority there. A minority of malicious nodes can be detected and punished accordingly. A participation of all nodes in the network in the voting process would deliver the best result but is infeasible for scalability reasons. Therefore, we need a process for selecting a representative subsets of all nodes. We must ensure that this process cannot be tampered with by malicious nodes – otherwise an attacker can gain majorities by selecting malicious nodes. Clearly, the decision to take part in a voting should not be up to the node itself. It should be decided by a majority again and every node should be able to verify the selection process.

A possible approach would be to use a global function to compute responsible nodes verifiable by all honest nodes. A hash function would be a straight-forward solution. The identifier of the information that should be agreed upon could be hashed to generate multiple keys (by hashing multiple times, appending a serial number or using the scheme we present later). Looking up these keys will deliver nodes to take part in the voting. The uniform distribution of the hash function guarantees the forming of representative subsets with a high probability.

This subset will be used for storing persistent information. This information must be stored redundantly to ensure it is available and will not be tampered with. Having each node of the selected subset store a copy of that information is a straight-forward approach. A node trying to retrieve the stored information can generate the keys for the replicas using the hash function and look them up. As long as the majority of selected nodes is honest, the correct result will be obtained.

However, a malicious node may influence voting outcomes at the routing layer. If some of the queries sent out to the replicates are routed via the same malicious node, it may simply drop all queries that would be answered by honest nodes. The remaining queries might be answered by colluding malicious nodes, leading to an incorrect result. To get the maximum benefit from a system of mutual checking and majority voting, we have to ensure paths to be completely disjoint.

4 Routing with Disjoint Paths

Our approach is based on the observation that disjoint routes to r replicas can be obtained if these routes are contained in different non-overlapping segments of the keyspace. Thus, a node querying for some data must send r messages directly to nodes in these segments. These nodes are responsible for forwarding the messages to the final recipient. Meanwhile, we must assure that the messages stay within their segments.

4.1 Replica Placement

We obtain these segments $S_1, ..., S_r$ by dividing the keyspace into r equally-sized intervals:

$$S_1 = [0, \tfrac{N}{r}), S_2 = [\tfrac{N}{r}, 2 * \tfrac{N}{r}), ..., S_{r-1} = [(r-2) * \tfrac{N}{r}, (r-1) * \tfrac{N}{r}), S_r = [(r-1) * \tfrac{N}{r}, N).$$

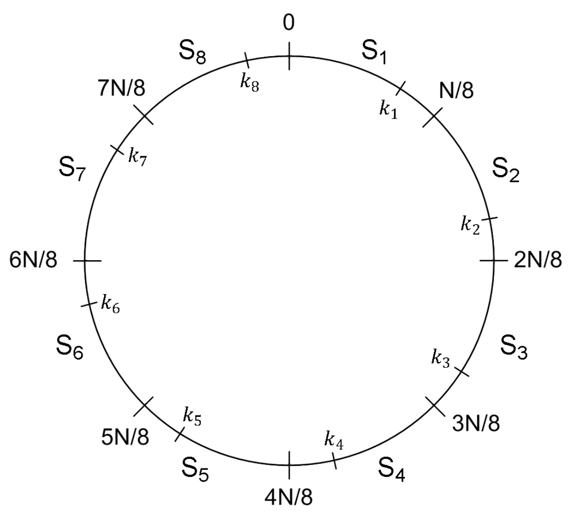


Figure 1: Symmetric Replication with Replication Factor 8

as shown in figure 1. Keys for replicas will be calculated similarly starting from the original key k_1: the $r-1$ copies will be mapped to the keys

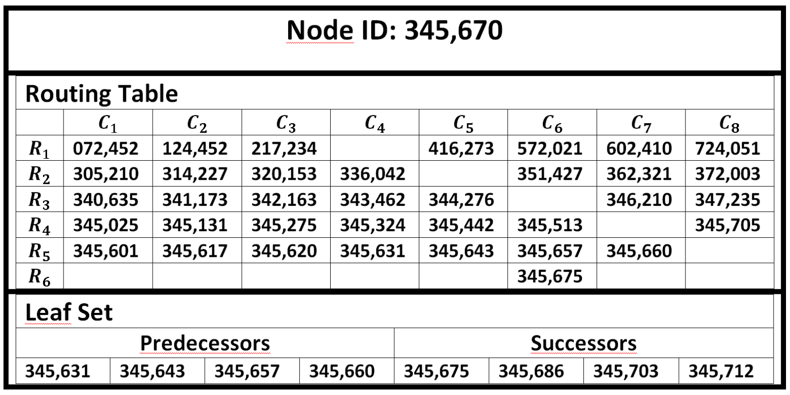
$$k_2 = (k_1 + \tfrac{N}{r}) \mod N, k_3 = (k_1 + 2 * \tfrac{N}{r}) \mod N, ..., k_r = (k_1 + (r-1) * \tfrac{N}{r}) \mod N$$

in a wrap-around fashion. This guarantees that all keys are in different segments. Furthermore, each node is able to calculate the $r-1$ keys for all replicas from the original key. By sending out r queries for the different keys and collecting the result, a node can do a majority voting on the results obtained. We only need to ensure paths to be disjoint so a malicious node cannot drop correct votes to gain a majority. Without loss of generality, we will denote keys with indices equal to the segment they are in, so k_i is in S_i from now on.

4.2 Radix-based Prefix Routing

We adopt the radix-based prefix routing of Pastry to reach our goal. Similar to Pastry, each node maintains a routing table called *leaf set* containing IDs and addresses of the closest nodes in the keyspace in both directions. With $|L|$ being the size of the leaf set, there will be $|L|/2$ nodes with lower IDs and $|L|/2$ nodes with larger ID in the leaf set. This knowledge of the local neighbourhood is necessary to guarantee that messages will be routed closer to their destinations. A node is either responsible for the key a message should be routed to or it knows another node from its neighbourhood closer to the key.

The routing table is used to provide the necessary shortcuts in the network to reduce the average number of routing steps to $O(\log N)$. In each step, a message is forwarded to a node from the routing table with the longest matching key prefix. The goal is to forward the message to a node with an ID matching at least one more digit than the current node, if the current node is not responsible and the closest node is not in the leaf set. Figure 2 shows an example routing table and leaf-set of a node with ID $345,670$. The routing table has a number of columns $C_1, ..., C_b$



Node ID: 345,670

Routing Table

	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
R_1	072,452	124,452	217,234		416,273	572,021	602,410	724,051
R_2	305,210	314,227	320,153	336,042		351,427	362,321	372,003
R_3	340,635	341,173	342,163	343,462	344,276		346,210	347,235
R_4	345,025	345,131	345,275	345,324	345,442	345,513		345,705
R_5	345,601	345,617	345,620	345,631	345,643	345,657	345,660	
R_6						345,675		

Leaf Set

Predecessors				Successors			
345,631	345,643	345,657	345,660	345,675	345,686	345,703	345,712

Figure 2: Routing Table and Leaf Set

equal to the radix b and rows $R_1, \ldots, R_l$ equal to the number of digits per key l. For a radix $b = 8$ and keys from 0 to $777,777o$ (octal) the number of columns will be 8 and the number of rows will be 6. A row $R_i, i \geq 1$ contains nodes with IDs matching the current node's ID in the first $i-1$ positions and having all possible $i-1$ different digits in position i (counted from left to right). Column C_j will contain the digit $j-1$ at position i of the entries in that row. One entry will always be empty: the column C_j with $j-1$ matchig the current node's ID also in position i. An entry might also be empty if a node has not found another node with ID matching the requirements of the routing table for an entry.

Routing of messages takes advantage of the routing table structure as follows: If a message arrives at a node matching this node in the first x digits, in row R_{x+1} there will always be a node matching the key in one more digit because entries with all possible different digits will be there. If an entry is missing or the node is unreachable, the message will be forwarded to another node closer to the target key instead. Even if there is no match between the current node and the target key, the first row contains one node matching in the first digit. Thus, there is always a closer node in the routing table unless it is in the leaf-set or the current node is responsible. For a detailed explanation of Pastry's routing mechanism we refer to [RD01].

The radix b can be chosen freely in theory. The internal binary representation of numbers favors a radix that is a power of two. We choose a radix b equal to the desired replication factor r. As we will show, this will guarantee disjoint paths with the symmetric replication scheme presented before.

The relation between radix-based prefix routing and symmetric replication is straight-forward: keys in different segments have different digits in the first position. More precisely, for segments $S_1, .., S_r$ the first digit d_{l-1} of key k_i will always be equal to $i-1$. For a key k_1 from segment S_1, the first digit will be 0, for k_2 from segment S_2 the first digit will be 1 and so on.

When a query is started from node A, only one of the r messages sent out for keys k_1 to k_r will match A's ID in the first position. For this message, there will be a closer node somewhere in the rows $R_2, \ldots, R_l$ of A's routing table, if the closest node is not in the leaf-set and A is not responsible itself. The message will be sent to this node, which is in the same segment as A because both have the same digit in the first position. All other messages will have no common

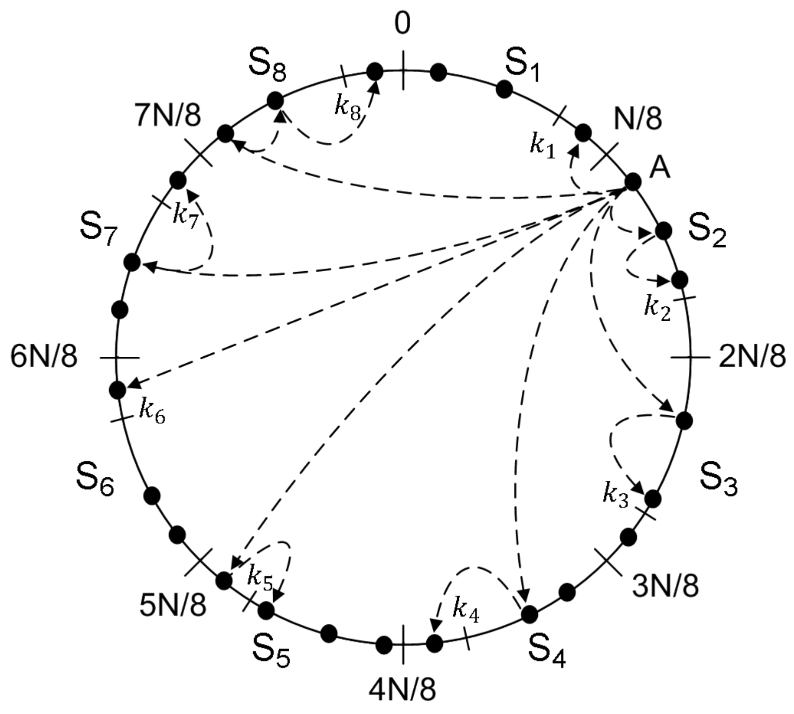


Figure 3: Disjoint Paths with Prefix Routing and Symmetric Replication

prefix with A's ID. These messages will each have a matching entry with the same digit in the first position in the first row R_1. Since all digits in the first position of these entries are different, all of the nodes are in different segments. Consequently, the r messages will be sent to r different nodes in r different segments as shown in figure 3 for a node A sending messages to the replicas.

Now we only need to ensure that a message does not leave the segment it has been sent to. When a node B receives a message with key k, three cases can arise. First, B does not find any closer node in its routing table or leaf set. Then B is the closest node and responsible for k. Thus the message does not leave the segment. Second, the closest node C might be in the routing table. B will look into its routing table to find nodes matching k in more positions than it matches B's ID or being at least closer to k than B. Since k matches B's ID at least in the first digit, C can only be in rows $R_2, ..., R_l$ where all nodes match B's ID in the first digit, too. Thus, the next hop C will also match k at least in the first digit. It will stay in the same segment in this case, too.

In the third case, the closest node is in the leaf set. With the closest node being responsible for a key, it could happen that a node B near a segment border actually finds a node C in its leaf set from the other side of the border to be closest to k. In that case it would forward the message to C in another segment. In the worst case, C could actually be responsible for two keys: the one from C's segment and the one from B's segment and paths would not be disjoint. However, this would only happen if B was the only node in its segment. It can be shown that as soon as there is another node D in the same segment as C, it will automatically be responsible for the key in that segment. Consequently, both keys cannot end up on the same node. We will skip the proof for space reasons.

The case of only one node in a segment is quite unlikely to actually show up when a certain amount of nodes are present. As node IDs are evenly distributed, every segment should have its proportional share of all nodes in the network.

4.3 Discussion

The increased security of disjoint path routing comes at a cost. When distributing replicas in the neighbourhood of the responsible node, only one query is necessary taking $\log N$ routing steps and $\log N$ messages. In our scheme, r independent queries are issued resulting in $r \log_r N$ exchanged messages. Since queries will be processed concurrently in the network after they have been initiated sequentially at the first node, the time to complete a query does not increase as strong as the number of messages. However, since we usually wait for the last returning query to do the voting, the slowest returning query dictates time of completion of the whole query. It will therefore be slower than running only a single query.

However, we can introduce a performance tradeoff here as we can use the f fastest returning queries to do the voting for some $f < r$. This could actually lead to faster query completion at the cost of security. However, malicious nodes could get a higher influence having fast network links.

A larger r gives more confidence to obtain the correct result. As node IDs are uniformly distributed and nodes cannot choose their IDs, picking r nodes creates a random subset of all nodes. The larger this subset the less likely malicious nodes have a majority in the subset when they are a minority in the whole network.

As r is also the radix of our prefix routing, increasing r also lowers the number of routing steps. The number of routing steps in prefix routing is $O(\log_r N)$ [RD01]. At the same time, the size of the routing table increases. While the additional memory required might not have a large impact, the amount of state to be maintained might. The larger the routing table, the higher the probability that routing entries are out of date when churn is the same. This could lead to messages taking longer routes and more overhead for requesting new entries from other nodes.

Probably the biggest problem is the vulnerability of prefix routing to routing attacks. As discussed before, malicious nodes can be prevented from getting a disproportionate large share of entries in a victim node's routing table by putting very tight constraints on the nodes that can show up in such an entry. The bounds for entries in the first rows of routing tables are very loose in prefix routing. In the first row, only the first digit of the node ID must have a specific value to enter a specific place. Thus, in the worst case an attacker only needs one node in every of the r segments to poison the routing tables of other nodes successfully by providing false routing updates. Every query to other segments would then be answered by a malicious node. This example shows that loose bounds can totally offset the security benefit of disjoint path routing.

There are approaches proposing to place bounds on the degree of nodes – the number of times one node may show up in other nodes' routing tables [SNDW06]. Considering the example from above, the r malicious nodes would show up in virtually every node's routing table. A reasonable bound would be to restrict the number of occurences of one node to its share in the whole network. However, there will be increased overhead to control the compliance of bounds.

5 Conclusion and Future Work

The combination of radix-based prefix routing and symmetric replica placement allows a r-disjoint path routing of messages in P2P-based distributed hash tables. This feature is especially useful when we cannot trust individual nodes but assume a majority of nodes to be honest. In such a setting, a majority voting performed by a representative subset is the only way to get the correct results with a high probability. However, in standard neighbourhood-based replication schemes messages to different replicas are routed along similar paths. A single malicious node in these paths would be able to corrupt the votes of multiple nodes. Our disjoint path routing scheme prevents this.

However, the security gain comes at a cost as the number of messages increases by the replication factor r. It remains to be evaluated wether the disjoint path routing delivers the necessary performance for the planned voting mechanism in P2P-based MMVEs. Mechanisms to deal with the vulnerability of prefix routing to routing attacks like the on presented in [SNDW06] will further decrease performance. However, since storing and checking persistent information is less time critical than fast-paced realtime actions in virtual worlds, our routing scheme is still feasible.

Consequently, we are currently implementing our routing scheme and we will evaluate it with varying replication factors. On top of that, we will also implement a voting algorithm to be used within the distributed hash table. Evaluations will show the real-life overhead induced by the additional security.

Bibliography

[ALG05] M. S. Artigas, P. G. López, A. F. Gómez-Skarmeta. A Novel Methodology for Constructing Secure Multipath Overlays. *IEEE Internet Computing* 9(6):50–57, 2005. http://doi.ieeecomputersociety.org/10.1109/MIC.2005.117

[Bli05] Blizzard Entertainment Inc. World of Warcraft. http://www.worldofwarcraft.com, 2005.

[CDG+02] M. Castro, P. Druschel, A. J. Ganesh, A. I. T. Rowstron, D. S. Wallach. Secure Routing for Structured Peer-to-Peer Overlay Networks. In *Proceedings of the 5th ACM Symposium on Operating System Design and Implementation (OSDI-02)*. Operating Systems Review, pp. 299–314. ACM Press, New York, Dec. 9–11 2002.

[DKK+05] D. Dumitriu, E. W. Knightly, A. Kuzmanovic, I. Stoica, W. Zwaenepoel. Denial-of-service resilience in peer-to-peer file sharing systems. In Eager et al. (eds.), *SIGMETRICS*. Pp. 38–49. ACM, 2005. http://doi.acm.org/10.1145/1064212.1064218

[Dou02] J. R. Douceur. The Sybil Attack. In Druschel et al. (eds.), *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*. Lecture Notes in Computer Science 2429, pp. 251–260. Springer, London, UK, 2002.

[GAH05] A. Ghodsi, L. O. Alima, S. Haridi. Symmetric Replication for Structured Peer-to-Peer Systems. In Moro et al. (eds.), *DBISP2P*. Lecture Notes in Computer Science 4125, pp. 74–85. Springer, 2005.

[HB06] C. Harvesf, D. M. Blough. The Effect of Replica Placement on Routing Robustness in Distributed Hash Tables. In Montresor et al. (eds.), *Peer-to-Peer Computing*. Pp. 57–66. IEEE Computer Society, 2006.
http://doi.ieeecomputersociety.org/10.1109/P2P.2006.44

[LDH06] M. Leslie, J. Davies, T. Huffman. Replication Strategies for Reliable Decentralised Storage. In *ARES*. Pp. 740–747. IEEE Computer Society, 2006.
http://doi.ieeecomputersociety.org/10.1109/ARES.2006.108

[RD01] A. Rowstron, P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Lecture Notes in Computer Science* 2218:329–350, 2001.
http://link.springer-ny.com/link/service/series/0558/papers/2218/22180329.pdf

[SCDR04] A. Singh, M. Castro, P. Druschel, A. I. T. Rowstron. Defending against eclipse attacks on overlay networks. In Berbers and Castro (eds.), *ACM SIGOPS European Workshop*. P. 21. ACM, 2004.
http://doi.acm.org/10.1145/1133572.1133613

[SM02] E. Sit, R. Morris. Security Considerations for Peer-to-Peer Distributed Hash Tables. In Druschel et al. (eds.), *IPTPS*. Lecture Notes in Computer Science 2429, pp. 261–269. Springer, 2002.
http://link.springer.de/link/service/series/0558/bibs/2429/24290261.htm

[SMK$^+$01] I. Stoica, R. Morris, D. Karger, F. Kaashoek, H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*. Pp. 149–160. ACM, San Diego, California, United States, 2001.
citeseer.ist.psu.edu/stoica01chord.html

[SNDW06] A. Singh, T.-W. Ngan, P. Druschel, D. S. Wallach. Eclipse Attacks on Overlay Networks: Threats and Defenses. In *INFOCOM*. IEEE, 2006.
http://dx.doi.org/10.1109/INFOCOM.2006.231

[Woo08] B. S. Woodcock. An Analysis of MMOG Subscription Growth Version 23.0. http://www.mmogchart.com, April 2008.

[ZHS$^+$04] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, J. Kubiatowicz. Tapestry: a resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications* 22(1):41–53, 2004.
http://doi.ieeecomputersociety.org/10.1109/JSAC.2003.818784