Malicious Behaviour in Content-Addressable Peer-to-Peer Networks

Thomas Reidemeister
University of Magdeburg, Germany
treideme@student.uni-magdeburg.de

Klemens Böhm
University of Karlsruhe, Germany
boehm@ipd.uka.de

Paul A.S. Ward
University of Waterloo, Canada
pasward@ccng.uwaterloo.ca

Erik Buchmann
University of Magdeburg, Germany
buchmann@iti.cs.uni-magdeburg.de

Abstract

Distributed Hash Tables (DHTs) promise to manage huge sets of key-value pairs in a Peer-to-Peer manner. The Content-Addressable Network (CAN) is a prominent variant of DHT. A critical challenge when designing a CAN, or indeed any DHT, is ensuring that all data items are accessible despite the presence of malicious and faulty peers. Such peers may hinder other peers in accessing the keys in various ways. In this paper we identify various types of attacks and propose, where possible, some countermeasures. To counter man-in-the-middle attacks we have developed a dynamically-adjustable multi-path routing algorithm. We evaluate the efficacy of our method both analytically and by simulation. For networks with less than 1% *malicious peers we were able to reduce the effect of such attacks by* 80%.

Keywords: networking, malicious behaviour, CAN, attacks, DHT

1. Introduction

Distributed Hash Tables (DHTs) are designed to manage huge sets of key-value pairs under high workloads. They form a self-organizing overlay network on top of large physical networks that consist of autonomous and anonymous peers. The Content-Addressable Network (CAN) [12] is a prominent kind of DHT. While other DHT variants are available, notably Chord [16] and Pastry [13], this article focuses on CANs. At present, DHT design in general, and the CAN in particular, presumes that there is no malicious behaviour, and the will to carry out operations is established among the peers. For a closed group of users these assumptions are reasonable. For an open network, with anyone free to join, these assumptions are not realistic. In this situation, the CAN must remain operational in the presence of malicious, faulty, or uncooperative peers.

We define *malicious peers* as those peers which intentionally harm the operation of the CAN. It is not necessary for the peer(s) causing the harm to obtain any direct advantage by their attack. For example, a peer that commits an attack may not, by its attack, be able to issue queries more easily. Motive is not relevant to our work. That said, it may typically be the case that the intention behind being malicious is to make information unavailable to other peers, possibly generating and disseminating false information.

We define *faulty peers* as peers that manifest faults, but that do not harm the network intentionally. Their behaviour can be the result of poor protocol implementations, software or hardware failures, *etc.* Given their lack of malicious intention, and assuming independent failure, it is reasonable to posit that faulty peers do not intentionally co-operate in harming the network. Malicious peers, by contrast, might co-operate according to their purposes. Since byzantine failure is indistinguishable from malice, we will not, in general, make a distinction between faulty and malicious peers. We define any action by either of these peer types that harms the CAN as an *attack* on the CAN.

Uncooperative peers wish to benefit from the network without carrying out operations related to requests issued by other peers. Research in countering such free riding, in the absence of malice, has already been done by various researchers (*e.g.*, [1, 2, 3, 7]), and is not our focus here.

In this paper we identify various attacks on the CAN that are the result of malicious or faulty peers. We describe possible countermeasures to some of these attacks and present simulation results that demonstrate the efficacy of our countermeasures.

To counter man-in-the-middle attacks we developed a dynamically-adjustable multi-path routing algorithm. In Section 4, we evaluate the efficacy of this algorithm analytically and by simulation. We show that for networks with less than 1% malicious peers we were able to reduce the effect of man-in-the-middle attacks by 80%.

2. Content-Addressable Networks

The Content-Addressable Network (CAN) is a large-scale Distributed Hash Table (DHT). It implements the usual hash-table operations, namely, the *insertion*, *lookup*, and *deletion* of values that are mapped to keys. The key space is mapped to an n-dimensional torus. Each peer of the network manages a *zone* of the key space and stores additional contact information of nodes[1] that manage the adjacent zones. We refer to a CAN as *regular* if all peers manage equally-sized zones. In general, a CAN will not be regular. However, regular CANs are more amenable to mathematical analysis, and are thus a useful concept.

The peers of the CAN issue and perform the operations. We define *operations* as performing a hash-table operation on the local zone of a peer. A *request* is generated for each operation that cannot be satisfied from the zone of the current peer. This request contains the *destination-key*, the *sender*, and a *value* in the case of *insert* requests. The request is forwarded to the neighbour that is closest to the destination key. We refer to this as *forwarding* or *routing*. This peer in turn forwards the request to its neighbour closest to the destination key. We refer to the peers forwarding a request as *forwarders*. Forwarding is performed until the request arrives at the peer that manages the zone containing the destination key. We refer to this peer as the *receiver*. The receiver performs the operation and sends the requested information *directly* back to the original sender of the request.

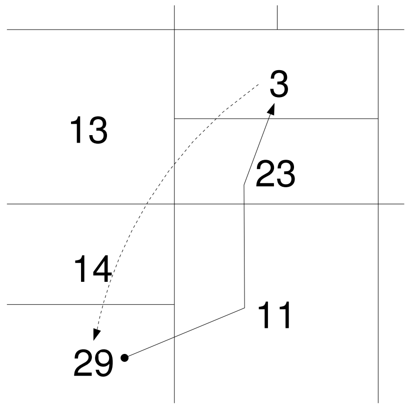

Figure 1. Sample routing in a 2d CAN

Figure 1 shows an example routing path for a request generated by node 29 for a key that is managed by node 3. The request contains node 29 as sender and the key. Node 29 forwards the request to node 11, because node 11 has the least *distance* to node 3 among its neighbours. Node 11 forwards it to node 23, which in turn forwards the request to node 3. Node 3 performs the operation and sends the result directly back to node 29. For completeness we mention here that the *neighbourhood* of a zone can be defined in different ways. Ratnasamy *et al.* [12] define two peers as neighbours if their zones overlap in $d-1$ dimensions, where d is the dimension of the CAN. Thus, each peer has $2d$ neighbours if the CAN is regular. We refer to this as *city-block* neighbourhood. Alternately, two peers can be defined as neighbours if their zones have one point in common. In this approach a peer has $3^d - 1$ neighbours in a regular CAN. We refer to this as *point* neighbourhood. In this paper we use point neighbourhood if not stated otherwise.

[1]We will use the terms "node" and "peer" interchangeably.

3. Attacks

There are a number of attacks to which peer-to-peer (P2P) protocols in general, and the CAN system in particular, are vulnerable. Sit and Morris [14] have enumerated a number of attacks on generic P2P protocols. In this section we present vulnerabilities we have identified on the CAN system specifically. We provide, where possible, countermeasures to such attacks. The reader should note that we do not deal with attacks at the application level (*e.g.*, the insertion of bad content into the CAN) or on the underlying network (*e.g.*, TCP/IP-level attacks). In the latter case, we presume that the core network is relatively secure. In the former case, we leave it to application-level semantics to deal with the problem.

In Section 3.1 we describe man-in-the-middle attacks. We analyze denial of service (DOS) attacks in Section 3.2. We describe attacks that can be committed by a malicious zone owner in Section 3.3. In Section 3.4 we describe attacks that harm the structure of the CAN. In Section 4 we will focus exclusively on countermeasures to the man-in-the-middle attack, providing an algorithm to address it, analysis of that algorithm, and simulation data to confirm our analysis of the efficacy of our approach.

3.1. Man-in-the-middle attacks

Man-in-the-middle attacks in the Content-Addressable Network may occur while routing a request. A malicious or faulty peer on the route to the destination may modify or drop requests. These attacks can either be committed intentionally, to prevent the sender from gathering or inserting specific information into the CAN, or they can be the result of peer error. Our simulation results (see Section 4.2) show that this type of attack is a serious problem, with just a small number of malicious nodes needed to corrupt a very large large number of requests. For example, in a 10,000-peer CAN, less than 1.5% of nodes need to be malicious for one in ten requests to be corrupted. Further, this problem grows with the size of the CAN, as the route lengths will grow.

As described in Section 1, the request of a message contains contact information of the sender, the destination key, and further information related to the operation to be performed. These are properties that are accessible to every forwarder. The effect of modifying one of these properties is a corrupted request, leading to three basic attacks:

1. If the sender of the request is modified, then the response for this request will not reach the original sender.
2. If the destination key of a lookup request is modified, then the sender of the request will receive a false result. If this happens to an insert request, then the consistency of the CAN is harmed.
3. Modifying additional information attached to a request also harms the consistency of the CAN. For example, if the value to be inserted by an insert request is modified, an illegal value is inserted at the destination key.

The occurrence of these characteristics of the man-in-the-middle attacks can be detected. Attack one can be detected by the sender using a timeout for each request. To detect modifications of additional information by malicious peers, the receiver can directly ask the sender to confirm the information. However, a combination of attacks one and three cannot be ruled out by only using such a handshake. A malicious node could replace the sender of a request with its own contact details, modify the additional information, and perform the handshake. From the perspective of the receiver the modified request seems to be correct. Our approach to deal with such attacks, as we will describe in detail in Section 4, is to use multiple paths to the destination, thus precluding all but colluding man-in-the-middle attacks.

We note here that an alternate approach to counter man-in-the-middle attacks is to use a public-key infrastructure [15]. The requests are signed by the sender and the receiver can check the authenticity of the signature by obtaining the public key from a central authority. However, the use of a central authority is not consistent with the idea of peer-to-peer DHTs. Other attacks using distributed algorithms [6] are too expensive in a setting such as ours.

While man-in-the-middle attacks can be detected, as we shall see in Section 4, the desired operations of the request may not be carried out properly without further countermeasures. There are several approaches to address this problem.

Ratnasamy *et al.* [12] suggest *data replication*. This approach replicates the key space into multiple realities. In every reality, the same partitions of the key space are owned by different peers. For example, each reality is described by another hash function. If the CAN is a read-only structure, then issuing the request in all realities reduces the probability of the request being corrupted. A problem with this approach is the potentially large data overhead by replicating the CAN. Naively issuing the request in every reality also produces a significant message overhead.

Sit and Morris [14] propose to let the sender observe the routing path. Because greedy-routing is used, the distance from the forwarders zone to the destination key will decrease from hop to hop. If an anomalous hop occurs the sender could ask the last correct forwarder to use a different routing path. For example, if the distance to the destination key increases after the hop, the sender asks the previous forwarder to provide a different routing path.

This could solve the problem of modifying the destination. However, checking each message hop is likely to increase the bandwidth and time overhead substantially. The sender of a request is also in a guessing situation which peer has modified the message. Further, a number of colluding peers could pretend to form a correct routing path and one could claim to be the destination by sending a spoof reply [8].

In Section 4 we present our solution to the man-in-the-middle problem, which addresses the various concerns described above.

3.2. Denial of service attacks

Denial-of-Service (DOS) attacks have received a lot of attention in the past years. The idea behind this attack is to generate a sufficient workload on a particular peer, by one or more other peers, to bring that peer down. If a denial-of-service attack is committed by a number of nodes it is typically referred to as a *distributed denial of service attack*. We do not make such a distinction, or consider it relevant to our work here. The motivation behind this attack is to make information unavailable to other peers by consuming the resources of the peer that provides this information, or to prevent the attacked node from obtaining services/data from the CAN.

We begin our analysis of this attack on the CAN by identifying types of this attack. In Section 3.2.1 we describe how the classic Denial-of-Service (DOS) attacks can harm the operation of the CAN. An advanced DOS attack, the bogus-request attack, is outlined in Section 3.2.2.

3.2.1 Classic DOS attack

The classic DOS attack is committed by a number of peers. They have chosen a particular key, or a number of keys, that point at data items they wish to make unavailable. At a certain time they start their attack by issuing a large number of requests targeting their chosen keys. Because the attack consumes the resources of the target peers, those peers might become unavailable. The difficulty with dealing with this problem is that it is strictly impossible to differentiate between a valid, but large, workload and a denial-of-service attack.

Countermeasures Given the indistinguishableness of denial-of-service attacks from a large, but valid, workload,

Key	last forwarders
(234,234)	18
(234,234)	6
(234,234)	18
(234,234)	6
(123,123)	13
(234,234)	18
(234,234)	6
...	

Table 1. Example History of Recent Requests

our basic approach to this problem is to replicate data items in the CAN. Research on replication in P2P environments has been done by various researchers [5, 11, 12], and we draw on their work. We first address the problem in the context of a *read-only* CAN. In this instance *hop-replication* could reduce the impact of this attack.

The idea behind hop-replication is that a peer replicates frequently-requested data items to its neighbours. For example, considering Figure 1, imagine that the key $(234, 234)$ is held by peer 3, and is the target of an attack. Assume that peer 3 keeps a history of the recent queries and their last forwarders, with the most recent data in this history shown in Table 1. In this case, a lot of requests for key $(234, 234)$ came from peers 6 and 18. Peer 3 could therefore forward a copy of this data item to those peers. In this event, we refer to these peers as *replica keepers*. The result of this replication is that peers 6 and 18 can now satisfy requests on key $(234, 234)$ too. This improves the situation for peer 3. Now the attackers must not only consume the bandwidth of one peer, they must consume the bandwidth of multiple peers. Further, peer 3 is now available to process other requests. A further improvement is to allow the replica keepers to recursively forward the replicas to their frequent last forwarders.

In principle, this approach could also be used to detect the origin of the DOS-attack. If a peer refuses to hold a copy of the data item that is frequently accessed from its direction, then this peer is likely to be an origin of the DOS attack. The peers detecting this case can ignore any future queries from the peer identified.

We now consider the case where the CAN is modifiable, and thus we must deal with updates. One approach is to timestamp all replica data. If a replica times out, a new copy is obtained. Insert requests are forwarded to the peer containing the key in its key space, in a write-through caching style. This procedure is sometimes referred to as *lazy replication* [10].

Unfortunately, this approach offers a new possibility for denial-of-service attacks. The attacker now generates insert and update requests, which are sent directly to the owning peer, thus overloading it. To avoid this, the targeted peer forwards advice to its neighbours to ignore, or accept only a small fraction of, any incoming insert and update requests for a period of time for the attacked key. This approach has been inspired by Vig [17], who suggests blocking DOS attacks that follow certain patterns on web servers behind a gateway at the gateway, as depicted in Figure 2.

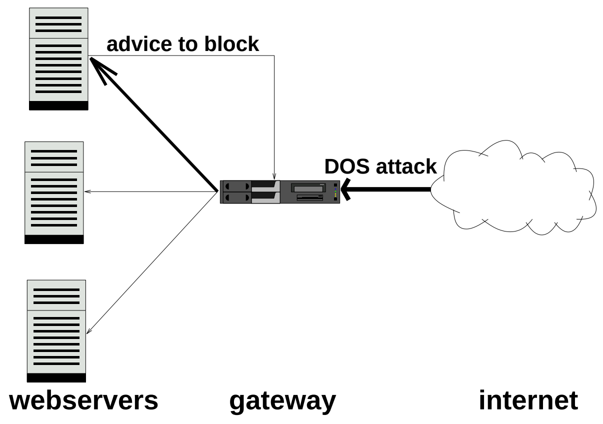

Figure 2. Blocking DOS attacks

In this approach, a web server is exposed to an extraordinary load. It assumes that the respective requests are part of a DOS attack, and thus advises its gateway to ignore any further incoming packets that follow this pattern.

Another approach to get rid of ”hot spots” is to use zone overloading, as described by Ratnasamy *et al.* [12]. This approach allows a peer to dynamically adjust the number of copies of its zone. It assigns joining peers copies of its zone. A DOS attack must then target all replicas, reducing its chance of success.

In this section we described countermeasures against DOS attacks. Replicating the attacked data items at the neighbours of the victim may be an effective way to make it available to other requests, although this approach is limited to query attacks. The efficacy of these approaches needs to be evaluated.

3.2.2 Bogus request attacks

The bogus-request attack is the inverse of the classic DOS attack. Instead of sending multiple requests to the target, multiple replies are transmitted. This attack is committed by a number of peers that generate bogus requests to random destination keys. The malicious peers do not include their contact information as sender. Instead they name the victim as sender and claim to be the forwarders of this request. As a result, the replies will primarily consume the bandwidth of the victim and make it unavailable. This attack is similar to the Smurf IP DOS attack [4], though that attack uses ICMP echo requests.

Countermeasure A possible countermeasure against this attack is handshaking. Before the destination peer of a *request* performs an operation, it contacts the sender included in the request and requests a *confirmation* for the desired operation. The efficacy of this approach depends on the size of the messages used in the CAN. For example, in a simple lookup service the message size is presumably the same as the size of a confirmation message. On the other hand, if the CAN is used as a backend for a distributed file system, the size of a confirmation message is small relative to the files. Instead of the data requested, the confirmation request only contains the key and not the requested data.

3.3. Attacks by a malicious zone owner

The previous attacks are committed by the senders and the forwarders of a request. The receiver can also behave maliciously, since it is the owner of the zone that contains the desired key. A malicious destination peer could drop the request or answer with a spoof result. Data replication [12] is required to prevent a single malicious zone owner from making the information unavailable. Such replication is the subject of much dependable distributed-systems research [9].

3.4. Attacks on the structure of the CAN

The CAN is a dynamic system. Peers can enter and leave the network frequently. A peer that joins takes over a part of a zone of a peer that is already part of the network. To join the CAN, the joining peer has to know a peer that is part of the network. We refer to this peer as the *contact peer*. The contact peer will then issue a query to a random destination key. The peer containing the destination is referred to as the *destination peer*. The destination peer splits its zone into two equal parts and assigns one part to the joining peer. The destination peer also informs all of its neighbours about the joined peer and removes the now non-neighbours from its neighbour set. After joining the CAN, the joining peer sends out periodic updates to its neighbours, as suggested by [12], so that node departures will be detected.

3.4.1 Modified split distribution

The destination peer is chosen by the contact peer. It is supposed to select the destination randomly. Instead, it could always choose the same destination key. This results in a highly fragmented area around the destination key. The intention behind this attack is to have a small zone and therefore a smaller number of queries. On the other hand, peers that have joined the CAN the way they are supposed to tend to have a larger zone. Hence, they will have to process more requests than the malicious peers.

3.4.2 Multiple joins

Because the CAN has no central authority that keeps track of joining peers, and the peer identifiers are not hashed onto the key space, as for example in Chord [16], it is possible for a peer to join multiple times. The intention behind this attack is to take over a larger area of the CAN. Having multiple zones in the CAN involves it in more routing paths. Thus, the peer can commit more man-in-the-middle attacks. The peer can also make these zones unavailable by acting as a malicious destination as described in Section 3.3.

3.4.3 Invalid splits

If the destination peer has to perform a split, it could use the opportunity to reduce its zone to a size smaller than its half. As a result, it reduces its future workload, because it does not have to carry out as many requests.

3.4.4 Countermeasures

Because we do not have a central authority that keeps track of joining and departing peers, it is hard to counter this type of attacks. If we did allow a central authority, it could assign zones of the key space to peers. This would prevent those attacks. However, this solution is not in line with the peer-to-peer paradigm.

4 Multi-path routing

In this section we give a detailed description of our multi-path routing algorithm that counters man-in-the-middle attacks. Our approach is to use *message replication*. Given that the key space is a Cartesian coordinate system, as in Figure 1, we can wrap around the edges. In general, each dimension of the CAN can be wrapped. Thus, we propose a multi-path routing algorithm that offers 2^d almost peer-disjoint paths, where d is the dimensionality of the CAN, to reach the destination peer from the source.

The CAN uses greedy routing to forward messages from the sender to the receiver: a forwarder calculates the distance to the destination key from all of its neighbours and selects the neighbour with the smallest distance to send the message to. Our algorithm redefines the distance calculation based on a bit mask. The number of bits of the bit mask in our context is the dimensionality of the key space. A bit mask describes a possible path from the sender to the destination: If (and only if) the bit for a certain dimension is set, the path wraps around this dimension. Figure 3 serves as an illustration for the two-dimensional key space.

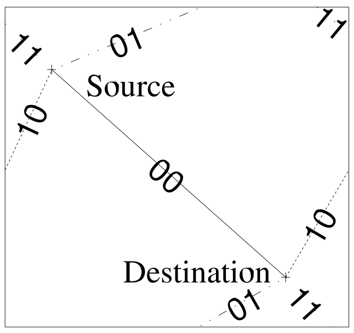

Figure 3. Bit-mask paths in $2d$-CAN

If we imagine a sender that is about to send out a request, instead of sending out one message, it sends a number of messages, which we term the *request messages*. For the time being, there is one message for each bit mask. But there can be less messages, as we will explain later in this section. We refer to the bit mask corresponding to a request message as its *request bit mask*. It specifies the direction where the sender is supposed to send the message, and it is attached to the message. Furthermore, all request messages contain the same sender, key and additional information. Sending several messages along disjoint paths reduces the probability that the request is corrupted. If at least one unmodified request message reaches the destination, the operation is performed. The sender must obtain at least two identical replies to its messages to identify correct answers. Two identical messages are sufficient if we assume that malicious peers do not collude. If collusion occurs, more than half identical replies would be needed, if we presume that malice is relatively rare.

When crossing the edge of a dimension, the bit mask must be changed. If not, this algorithm would end up in a loop. The message would be passed over the edge again. The neighbour on the other site would perform the distance calculation using the old bit mask. As a result, this peer would send the request back to its last forwarder and so on.

We use *situation checking* to modify the request bit mask. When a peer calculates the distance of one of its neighbours to the destination key, it creates a separate bit mask for its relation to the neighbour. We refer to this bit mask as the *situation bit mask*.

Figure 4 illustrates situation bit masks. The thick lines mark the edge of each dimension. The top edge of peer 17 is the top horizontal edge. Its right edge is the vertical edge. The peers 6, 18, 8, 1, 28, 15, 26, 11, 23 and 3 are the peers managing the adjacent zones. Because this is an example for the two-dimensional case, we have bit masks consisting of two bits. To reach the peers 1, 28 and 15 from peer 17 the horizontal dimension needs to be wrapped. This requires that the second bit is set to 1. For peers 6 and 18 the vertical

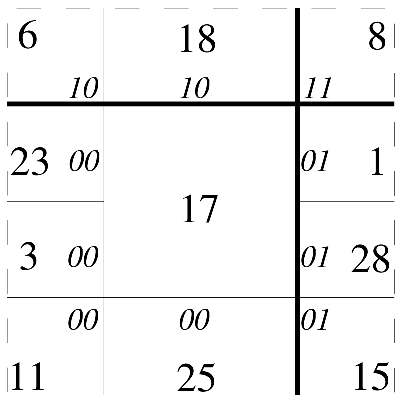

Figure 4. Sample situation bit masks

dimension needs to be wrapped. This requires that the first bit is set to 1. To reach peer 8 both dimensions need to be wrapped, thus both bits are set to 1. The bit masks of the other peers are set to 00, because they can be reached from peer 17 without wrapping any dimension.

The forwarders have to decide whether to send the request over the edge of the dimension or not. This decision depends on the situation bit mask $B_p(i)$ for each of its neighbours i and the request bit mask B_r. We use bitwise logical operators to make this decision. The masks are checked in the following order:

1. Neighbours that satisfy $(\neg B_r) \wedge B_p(i) = 1$ are not considered as possible forwarders.

2. If the peer can only reach one of its neighbours by wrapping a dimension, the request bit mask is modified as follows: $B_r := B_r \wedge (\neg B_p(i))$.

Case 1 excludes all neighbours that can only be reached by going over an edge that is not allowed by the request bit mask. If an edge is to be crossed, then case 2 sets all corresponding bits of the request bit mask to 0. For example, consider a request with a bit mask of 01 that is to be forwarded from peer 17. The first case prevents forwarding this request to the peers 6, 18, 8. If the request is to be forwarded over the horizontal edge, the bit mask is set to 00 according to case two.

4.1. Attack probability

In this section we compute the probability for a request to be subject to a man-in-the-middle attack. Sending out 2^d requests per operation is inefficient. A peer can adjust multi-path routing if it knows the probability of hitting a malicious node. If the probability is low, single-path routing can be used instead of multi-path routing, to save the data bandwidth overhead. We will at first describe the attack probabilities for single-path routing and multi-path routing.

Then we describe how to adjust the multi-path routing. For our formal analysis we make several assumptions:

- Malicious peers do not collude.
- A malicious peer is always malicious and does not change its behaviour.
- The total number of peers in the network is known to each peer or can be estimated.
- The destination peer of a request is never malicious.
- A request is only attacked once. Thus, an attacked path relates to one malicious node. Because we regard malicious behaviour as an exception, this assumption is reasonable.
- The malicious peers are uniformly distributed in the key space.
- The CAN is regular.

Single-path routing The attack probability for an attacked path for single-path routing can be constructed on top of the probability for an attack of one message hop p_h. We define one message hop as forwarding the request from a peer to its neighbour. We calculate the probability for one hop to be attacked from m, the number of malicious peers, and n, the total number of peers.

$$p_h = m/n \tag{1}$$

While n can be estimated, m is not known to the peer. We will later show how a peer can estimate p_h by using multi-path routing, enabling it to estimate m.

We can apply a binomial distribution with the parameters $p = p_h$ and $q = 1 - p_h$. The average path length is denoted by l. Since we assume that a request is only attacked once, we have to cumulate all possible permutations of at least one attack to happen.

$$P(X \geq 1) \quad = \quad 1 - P(X = 0) \tag{2}$$

$$P(X = 0) \quad = \quad \binom{l}{0} p^0 q^{\lceil l \rceil} \tag{3}$$

By simplifying and substituting our values, we get

$$P(X \geq 1) = 1 - (1 - p_h)^{\lceil l \rceil} \tag{4}$$

If two peers are defined as neighbours using the city-block definition, then the average path length l as number of hops can be calculated as

$$l = \frac{d}{4} n^{\frac{1}{d}} \tag{5}$$

When two peers are defined as neighbours using point neighbourhood, then

$$l = \frac{1}{4} (dn)^{\frac{1}{d}} \tag{6}$$

Equation 5 has been proposed by Ratnasamy *et al.* [12]. If we use point neighbourhood and assume that zones are very small, the Euclidean distance is a good approximation. Because of the torus property, the longest path is $\frac{1}{2}(dn)^{\frac{1}{d}}$. Thus, the average path length can be calculated by using Equation 6.

Estimating the percentage of malicious peers When using multi-path routing, a peer can estimate the percentage of malicious peers and the probability for a message to be attacked at one message hop. The receiver of a request appends the number of message hops $h_r(i)$ for each request message i to the result. The sender of the request can sum up the number of message hops of messages resulting in correct results. It also keeps a count of the corrupted messages c_f and the total count of correct messages c_t. For simplicity, we assume that the corrupted messages have been attacked in the middle of the path. We define C_t as the set of identifiers of all correct paths. Thus, we calculate the probability p_h for a message to be attacked at one particular message hop as follows:

$$p_h = \frac{c_f}{c_f \frac{l}{2} + \sum_{j \epsilon C_t} h_r(j) + 1} \tag{7}$$

Using this formula, we can estimate the attack probability for single-path routing. This in turn allows us to adjust the number of messages for multi-path routing. At a fairly low attack probability, the peer sends out a low number of messages per request. At a higher attack probability, the peer uses more messages per request. The peers can send out multi-path requests with a large number of messages periodically to update their estimation for p_h. However, adjusting the number of messages is non-trivial. Note that the relationship between the attack probability and the number of paths in multi-path routing is not linear. Deriving a comprehensive model is future work.

4.2. Simulation

We simulated the effects of man-in-the-middle attacks on the CAN. To enable comparison with our analytical results, we used a regular CAN with no peers entering or leaving during the simulation, $10,000$ peers, and $1,000,000$ uniformly-distributed queries. The number of malicious peers was varied from 0 to 500, uniformly distributed in the CAN. Figure 5 shows the impact of man-in-the-middle attacks using the classic three-dimensional CAN without any countermeasures and the CAN using multi-path routing. The horizontal axis shows the percentage of malicious peers. The vertical axis shows the percentage of requests corrupted. We examined up to 5% malicious nodes, since we regard malicious behaviour as an exception. In particular, if there are effective measures against malice, it will

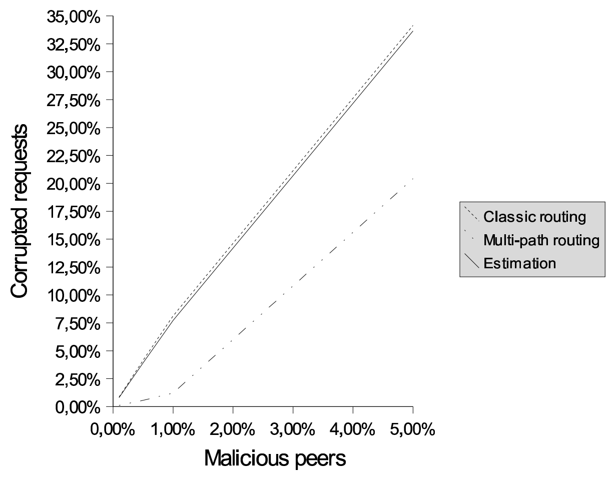

Figure 5. Impact of man-in-the-middle attacks

become rare, as it will be unproductive. We found a significant benefit in using multi-path routing in such cases. For example, when 1% of the peers are malicious our algorithm reduces the impact from 8.12% of queries being corrupted to 1.21%, an 85% reduction. Also as can be seen in Figure 5, the estimates for single-path routing obtained from our analytical model approximately match the simulation.

5 Conclusion

Content-Addressable Networks (CAN) are P2P overlay networks that allow storage and querying of huge volumes of data. However, their design thus far assumes that trust is given and there are neither malicious nor faulty peers in the CAN. This limits applicability to closed networks executing correct software on flawless hardware. Keeping a CAN operational in the presence of malicious and faulty peers makes it applicable for an Internet-scale open network. In this paper we presented various types of attacks on the CAN and possible countermeasures. We have demonstrated the efficacy of multi-path routing to eliminate the effect of man-in-the-middle attacks. Our future work is to evaluate the efficacy of other proposed countermeasures. The list of attacks identified is unlikely to be complete. Further, there are still open questions in how to deal with attacks on the application layer. In particular, the problem of peer inserting unwanted or unpopular data into the CAN remains open.

References

[1] K. Aberer and Z. Despotovic. Managing trust in a peer-2-peer information system. In *CIKM '01: Proceedings of the Tenth International Conference on Information and Knowledge Management*, pages 310–317. ACM Press, 2001.

[2] E. Adar and B. A. Huberman. Free riding on gnutella. *First Monday*, Sept. 2000.

[3] E. Buchmann and K. Böhm. Fairnet - How to counter free riding in peer-to-peer data structures. In *Proc. of the International Conference on Cooperative Information Systems 2004, Agia Napa, Cyprus*, Oct. 2004.

[4] CERT Coordination Center. CERT advisory CA-1998-01 smurf IP denial-of-service attacks. Technical report, CERT Coordination Center, Pittsburgh, PA, 5. Jan. 1998.

[5] E. Cohen and H. Kaplan. Balanced-replication algorithms for distribution trees. In *ESA '02: Proceedings of the 10th Annual European Symposium on Algorithms*, pages 297–309. Springer-Verlag, 2002.

[6] A. Datta, M. Hauswirth, and K. Aberer. Beyond "web of trust": Enabling p2p e-commerce. *Proceedings of the IEEE Conference on E-Commerce, USA*, 2003.

[7] P. Dewan and P. Dasgupta. Pride: Peer-to-peer reputation infrastructure for decentralized environments. In *WWW Alt. '04: Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters*, pages 480–481. ACM Press, 2004.

[8] J. R. Douceur. The sybil attack. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 251–260. Springer-Verlag, 2002.

[9] P. Jalote. *Fault Tolerance in Ditsributed Systems.* Prentice Hall, Englewood Cliffs, New Jersey, 1994.

[10] R. Ladin, B. Liskov, L. Shrira, and S. Ghemawat. Providing high availability using lazy replication. *ACM Trans. Comput. Syst.*, 10(4):360–391, 1992.

[11] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *SPAA '97: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*, pages 311–320. ACM Press, 1997.

[12] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 161–172. ACM Press, 2001.

[13] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350. Springer-Verlag, 2001.

[14] E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash tables. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 261–269. Springer-Verlag, 2002.

[15] W. Stallings. *Cryptography and Network Security: Principles and Practice, 3rd edition.* Pearson Education, 2002.

[16] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, 2003.

[17] A. Vig. Preventing denial of service attacks. Online article available at: http://www.onlamp.com/pub/a/bsd/2004/-06/24/anti_dos.html, June 2004.