Exercise 7

18.5/20

H#7.1 KAD

a) Summary

The paper covers new attacks and improvements of existing attacks on KAD. To compare the attacks' impacts, experimental measurements were taken. It is shown how an attacker with low resources can successfully harm a KAD network and which countermeasures can be taken.

→ which Vulnerabilities? 2.5/3

b)

paragraph 2.1

Kademlia ⟷ KAD

distance is measured with XOR

- replica are stored at replica roots
  ↳ nearest nodes according to XOR distance
- replica roots are within a search tolerance, depending on the number of nodes in the tolerance a variable number of replica is possible

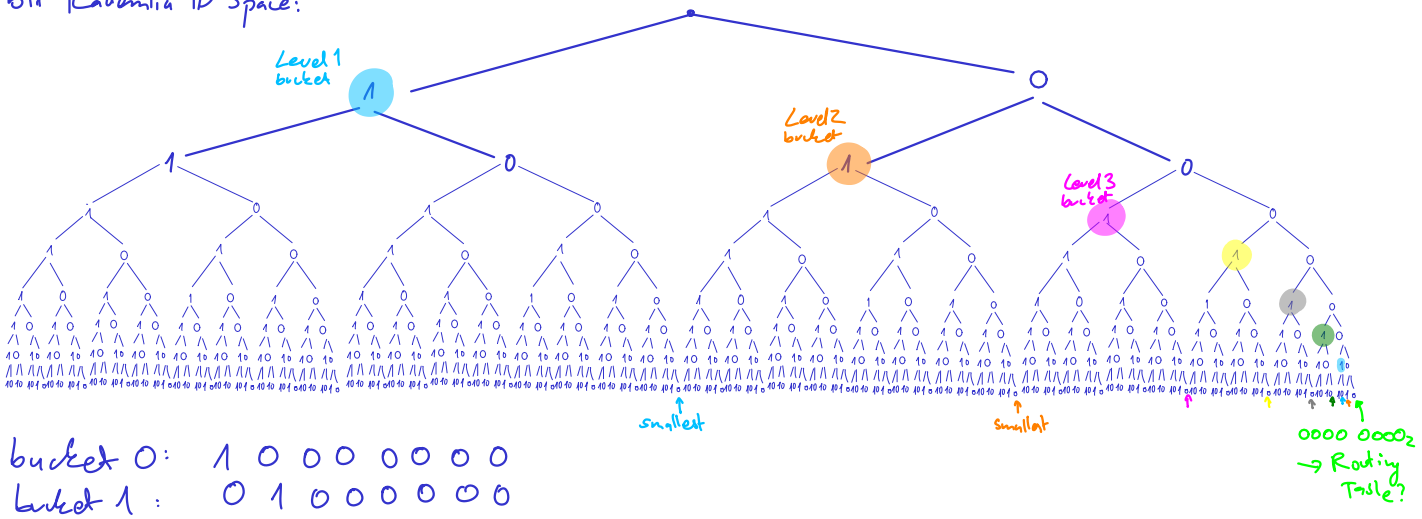
routing tables of a node are built via the longest prefix match of its ID. At the i^{th} level in the routing table, the first i bits have to match. If there are multiple matches, only k nodes are placed into the routing table; k is the bucket size.

- splitting of buckets is only possible at index 0
- if the level is ≥ 4, splitting is possible for index 0..4
  → shorter Routing paths,
  neighbourship between arbitrary nodes is possible

In KAD, less steps are required for routing than in Kademlia, because the splitting for index 0 to 4 makes routing tables wider which leads to shorter paths. Because routing in KAD is iterative as in Kademlia, a shorter routing paths results in less contacted nodes.

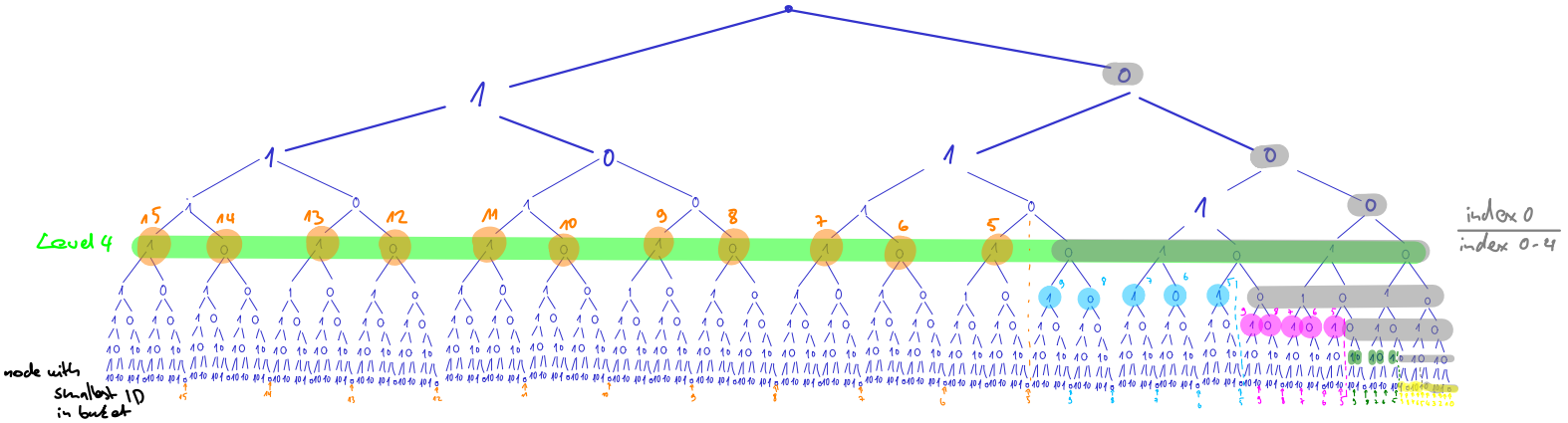
Kademlia Routing table

8 bit Kademlia ID Space:



bucket 0: 1 0 0 0 0 0 0 0
bucket 1: 0 1 0 0 0 0 0 0
bucket 2: 0 0 1 0 0 0 0 0
bucket 3: 0 0 0 1 0 0 0 0
bucket 4: 0 0 0 0 1 0 0 0
bucket 5: 0 0 0 0 0 1 0 0
bucket 6: 0 0 0 0 0 0 1 0
bucket 7: 0 0 0 0 0 0 0 1
(bucket 8: 0 0 0 0 0 0 0 0 (node itself))

KAD Routing table



Level 4:
bucket 15: 1 1 1 1 0 0 0 0
bucket 14: 1 1 1 0 0 0 0 0
bucket 13: 1 1 0 1 0 0 0 0
bucket 12: 1 1 0 0 0 0 0 0
bucket 11: 1 0 1 1 0 0 0 0
bucket 10: 1 0 1 0 0 0 0 0
bucket 9: 1 0 0 1 0 0 0 0
bucket 8: 1 0 0 0 0 0 0 0
bucket 7: 0 1 1 1 0 0 0 0
bucket 6: 0 1 1 0 0 0 0 0
bucket 5: 0 1 0 1 0 0 0 0

Level 5:
bucket 9: 0 1 0 0 1 0 0 0
bucket 8: 0 1 0 0 0 0 0 0
bucket 7: 0 0 1 1 1 0 0 0
bucket 6: 0 0 1 1 0 0 0 0
bucket 5: 0 0 1 0 1 0 0 0

Level 6:
bucket 9: 0 0 1 0 0 1 0 0
bucket 8: 0 0 1 0 0 0 0 0
bucket 7: 0 0 0 1 1 1 0 0
bucket 6: 0 0 0 1 1 0 0 0
bucket 5: 0 0 0 1 0 1 0 0

Level 7:
bucket 9: 0 0 0 1 0 0 1 0
bucket 8: 0 0 0 1 0 0 0 0
bucket 7: 0 0 0 0 1 1 1 0
bucket 6: 0 0 0 0 1 1 0 0
bucket 5: 0 0 0 0 1 0 1 0

Level 8:
bucket 9: 0 0 0 0 1 0 0 1
bucket 8: 0 0 0 0 1 0 0 0
bucket 7: 0 0 0 0 0 1 1 1
bucket 6: 0 0 0 0 0 1 1 0
bucket 5: 0 0 0 0 0 1 0 1

bucket 4: 0 0 0 0 0 1 0 0
bucket 3: 0 0 0 0 0 0 1 1
bucket 2: 0 0 0 0 0 0 1 0
bucket 1: 0 0 0 0 0 0 0 1 node itself
bucket 0: 0 0 0 0 0 0 0 0

c)

Paragraph 2.1

HELLO_REQ: A "hello request" is a request similar to a ping. It is used to test if a contact is still alive or went offline. The hello requests are only sent, if no search request was sent during the last 2 hours, because search requests renew the contact's liveness state, too.

SEARCH_REQ: A "search request" looks for keyword matches on replica roots. They are sent during a search process when a replica root was found with KADEMLIA_REQ. The search process stops when more than 300 keyword matches were received. (→ no more SEARCH_REQ and KADEMLIA_REQ messages will be sent)

KADEMLIA_REQ: A "kademlia request" looks for replica roots. The search key for the replica is the MD4 Hash of the keyword. KAD keeps a list of contacts. "kademlia requests" are sent to the 3 closest contacts in each step. Answers to these requests (KADEMLIA_RES) contain contacts of contacts, that are added to the list. Again, "kademlia requests" are sent to the 3 closest nodes and so on...
If the 3 closest nodes don't respond, the contact list contains backup contacts where the "kademlia requests" can be sent instead.

d) Sorting buckets

Lecture 3, slide 83

Kademlia avoids to rearrange contacts because they lead to routing tables with old entries. contacts which are seen get moved to bottom of the list.

paragraph 2.1

Kademlia only splits buckets at index 0. That leads to fewer possibilities for splits which means fewer possibilities for routing table entries.

If new nodes want to join, the rearrangement probability is kept low by prefering existing nodes, because a node that already stayed for a certain time has a high probability of staying longer than the new node.

e) Attacks

Attacks of Wang et al

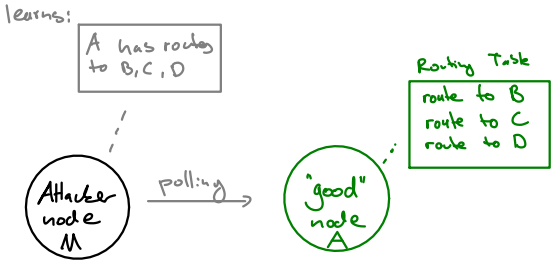
paragraph 3

The attack has a preparation and an execution phase, but it's possible starting them at once.

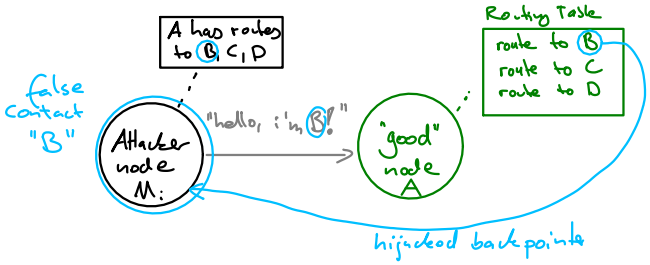
paragraph 3-1

During the preparation phase the attacker...

① starts nodes to discover the network. Node IDs are chosen in a way that divides the ID space in as many pieces as possible in order to start a fast discovery via polling routing tables from separated entry points.



② The attacker's nodes hijack backpointers from other nodes.



By hijacking additional routes, the attacker can prevent that the old (correct) route to B is discovered by A again.

paragraph 3-2

During the execution phase makes queries fail. KAD queries multiple closest contacts to a target, so multiple answers containing malicious nodes closer to the target are required. The paper covers the following 2 possibilities to do so:

- Terminate the search by sending more than 300 keyword matches.
- Terminate the search by pretending all contacts close to the target were stale. This works, because the possibility of attacker nodes to be contacted is high because they are close to the key.

paragraph 6.1

Sybil Attack

The goal of a Sybil Attack is to receive many query messages which is archieved by gathering back-pointers. All P2P networks like KAD are vulnerable because they don't have any query control.

The longer a Sybil node stays in KAD, the more back-pointers it gets. Back-pointers of nodes with a small common prefix length are hard to get because they are higher in the hirarchy where buckets are usually full in a highly populated network.

paragraph 6.2

Index Poisoning Attack

The goal of an index poisoning attack is to link keywords to nonexistant files. If the index is poisoned, user queries will mainly contain those nonexistant files. In KAD, this attack is performed by recording all "publish"-messages to get existing (keyword, file)-pairs and then poisoning them.

The effects of this attack are only small because the user can simply query more nodes to get a correct answer of the file's location and the main parts of the routing mechanism (joining, leaving, maintaining routing tables) aren't affected.