Decentralized Content Trust for Docker Images

Quanqing Xu[1], Chao Jin[1], Mohamed Faruq Bin Mohamed Rasid[2], Bharadwaj Veeravalli[2] and Khin Mi Mi Aung[1]

[1]*Data Storage Institute, A*STAR, Singapore*

[2]*Department of Electrical and Computer Engineering, National University of Singapore, Singapore*

Keywords: Trust, Docker, Blockchain, BitTorrent, Decentralized System.

Abstract: Default Docker installation does not verify an image authenticity. Authentication is vital for users to trust that the image is not malicious or tampered with. As Docker is currently a popular choice for developers, tightening its security is a priority for system administrators and DevOps engineers. Docker recently deployed Notary that is a solution to verify authenticity of their images. Notary is a viable solution, but it has some drawbacks. This paper specifically addresses its vulnerability towards Denial-of-Service (DoS) attacks, the repercussions, and discuss two potential solutions. The proposed solutions involve decentralising the trust via either a BitTorrent-like protocol or a modified blockchain. The solutions greatly reduce the risk of DoS and at the same time provide a trustless signature verification service for Docker. The solutions could also possibly be repackaged for similar use cases on other technologies. We demonstrate the proposed blockchain-based solution's scalability and efficiency by conducting performance evaluation.

1 INTRODUCTION

Docker is an emerging container technology with the goal to provide a solution to deploy an application along with the dependencies automatically in a self-contained environment called containers. The containers are isolated from each other and are given individual network interface. The isolation provides a solution similar to virtual machines (VM) (Arumugam et al., 2014) but without the overhead of running the guest operating system on top of the host operating system. Docker minimal overhead is achieved by sharing the kernel with the host operating system, unlike VMs that run the guest operating system's kernel on a hypervisor. The containerization is realized through Linux kernel namespaces and control groups allowing processes to be isolated and resources to be limited with negligible overhead. Processes running in containers are made to believe that it is running in a separate operating system than the host with its own allocated resources, such as CPU and memory.

While Docker wins in terms of performance, VMs are inherently more secure than Dockers since VMs can provide true isolation (Bui, 2015). Shared kernel translates to possibility of a container escaping the isolation and gains privileged access through kernel vulnerabilities. Since Docker and VM have some overlapping use cases, both technologies are competing directly for market share when system administrators and DevOps evaluate solutions for their deployments. Interest around Docker started increasing around Q4 2013. In fact, Docker popularity has been growing steadily since its 1.0 release, and Dockers are replacing VMs in many use cases. In addition, developers can rapidly build applications for the Internet of Things (IoT) (Xu et al., 2016) by using Docker containers to help them in many different ways.

Docker growth in popularity has helped it gain wider visibility and receive more adoptions and supports from developer community and large organizations alike. Datadog reported that between May 2015 and May 2016, Docker market share grew by almost 30% (Datadog, 2016). As adoptions increased, finding vulnerabilities and exploiting them became increasingly lucrative. Since Docker is used to deploy production server applications, successfully exploiting Docker containers would be synonymous to getting full access to the organization's web services that can be used for malicious purposes like siphoning data or impersonation. In the worst case scenario, an attacker could possibly break out of the container and gain access to the host OS via exploits like Dirty COW (CVE-2016-5195) (Khandelwal, 2016). Such an attack may cause catastrophic damage to both the servers and the organization itself.

Besides namespaces and cgroups, Docker containers filesystem is provided by Advanced Multi-layered Unification Filesystem (AuFS). AuFS, as a

Xu, Q., Jin, C., Rasid, M., Veeravalli, B. and Aung, K.
Decentralized Content Trust for Docker Images.
DOI: 10.5220/0006379404310437
In *Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security (IoTBDS 2017)*, pages 431-437
ISBN: 978-989-758-245-5

layered filesystem over one or more existing filesystems, creates a file copy when a process needs to modify the file. This process is called copy-on-write (Merkel, 2014). This feature allows an image to depend on another image (i.e., a base image), e.g., an nginx application depending on a CentOS 5.11 base image. Due to this dependency mechanism, if the base image is not updated with the latest patched binaries, any image depending on it will inherit the vulnerabilities if the publisher is not proactive at ensuring the image is up to date. It is also theoretically possible for an attacker to plant malwares and/or backdoors to a base image. A compromised image might even be used to gain privileged access to the host operating system via exploits like the CVE mentioned above. These issues are quite alarming given the worsening security climate in recent years. Its effect could be catastrophic to an organization who is being hacked.

This paper analyses Notary, a recently released software that is integrated into Docker as the feature called Docker Content Trust (DCT), to counter these issues. We investigate two areas: 1) how DCT is a viable solution, and 2) drawbacks of DCT. We propose two alternative solutions to Notary: 1) BitTorrent-like, and 2) blockchain-based decentralized distributed systems. As Docker is a new technology that is still going through rapid changes, up-to-date literature is usually in the form of online articles. To the best of our knowledge, this paper is the first one to propose decentralized content trust for Docker images.

The rest of the paper is organized as follows. Section 2 introduces background and motivation. We describe the drawbacks of Docker Content Trust in Section 3. We propose two potential solutions in Section 4. In Section 5 we present performance evaluation results. Section 6 describes related work. In Section 7 we conclude this paper.

2 BACKGROUND AND MOTIVATION

2.1 Decentralized Distributed System

The issues mentioned in the previous section are actually not unique to Docker Content Trust only. The issues stem from the limitations of a centralized system. In most simple centralized system designs, and in the case of Notary, it has a single point of failure. The solution to this issue is to build a decentralized distributed system.

In a decentralized system, there is no central authority. The decisions are either made independently by each node, or the nodes exists in a swarm with a swarm leader. A decentralized system is resistant to fault since if you take down a node, or a swarm, the system will still function normally. In order to take down the system, all the nodes have to be taken down. Distributed systems on the other hand is designed to withstand high workloads. Usually, there is still a central authority in a distributed system making decisions. For a DoS attack to be successful on such systems, the attacker would usually need to match the number of resources on the server side. Usually, the attacker would employ Distributed Denial of Service (DDoS) attacks in such scenarios.

2.2 Docker Content Trust

Docker is usually used together with a registry server where images are stored. Docker Hub and Quay.io are popular public registries. Although while pulling images the user would need to specify the repository to pull from, there is no built-in method to verify the integrity and the publisher of the received data. As the Internet is an untrusted medium of communication, it is critical to check if the received data have been tampered with. Responding to requests by community to have strong cryptographic guarantees of the code and versions being run in their infrastructure, Docker introduced Docker Content Trust (DCT) in version 1.8 of the Docker Engine (Mnica, 2015). DCT allows users to use the same Docker commands by seamlessly integrating signature verification to the workflow.

When a user interacts with an image through Docker commands in the first time, trust is established automatically with the image publisher. Subsequent interactions with the same publisher requires a valid signature verification. This model is known as trust on the first use, similar to SSH trust verification model. The implementation of DCT is actually done by integrating Notary into Docker Client and Engine. Notary is a separate project by Docker security team with the aim to make the Internet more secure by making it easy for people to publish and verify content. While the Notary project was started to add image integrity protection for Docker, it was designed to be used and integrated into other softwares.

2.3 The Update Framework (TUF)

Internally, Notary uses The Update Framework (TUF), a flexible security framework that developers can integrate with any software update system (TUF-spec, 2017). TUF provides protection against forgery, serving of data over untrusted mirrors, protection against replay and downgrade attacks. However, its most notable one would be survivable key compromise. This is in contrast to the design of most software update systems where a compromise of a single trusted key is fatal (Samuel et al., 2010). As survivable key compromise is central to the design of the TUF specifications, instead of a single key for signing, multiple keys are used instead. To be more specific, the keys belongs to a single hierarchy under a single root key which is kept offline. The other main keys are the snapshot key, targets key and timestamp key.

When used with Docker, the terminologies used for the keys are slightly different although the usage is conceptually the same. With Docker Content Trust, the documentation identifies only two main keys: 1) tagging key, which is used to sign data for individual repositories, and 2) offline key, which is the root key for all the tagging keys. These tagging key is synonymous to the targets/delegates key while the offline key is the root key. Using TUF and the keys, DCT is able to provide protection against some common forms of attacks where an attacker is at a privileged network position once the users permit opt-in DCT on their Docker installations.

3 DRAWBACKS OF DCT

Docker Content Trust provides adequate protection against some common attacks. However, there are some limitations to its security design decisions. The design decisions are influenced by the goals set by the development team and the different beliefs on what is to be prioritised when securing the system.

3.1 Limited Protection against Server Compromise

When Docker pulls a signed image, it checks with a Notary server to ensure the image's integrity and to verify the image's publisher. In the event of failure to verify these, the Docker command yields a hard fail. However, it is important to note that by default DCT adopts trust on the first use (mrled, 2017). Hence, if the user pulls the image in the first time and it is tampered and pointing to a different root metadata, the command still succeeds. When an attacker cannot tamper the file, another alternative attack he/she may try out is to send a previously valid payload to trick the clients not to update to the latest image that has a patch for the known vulnerabilities. Likewise, he/she may also attempt to downgrade the image to an older version that contains the vulnerabilities. It is mitigated with the timestamp key.

In Notary's implementation, the Notary server is in charge of signing the timestamp metadata file generated every time. When a Docker image is pulled, the verification of files' integrity is requested. The timestamp metadata contains an expiry field that indicates till when is the metadata valid. As a result, the attacker cannot replay a previously valid payload as the data freshness is also verified on top of the integrity of the data files. Although key rotations are possible, it is a mechanism for recovering from an attack in the case of a server compromise as oppose to protection against one. If the server is compromised, while the users are protected if they already have the root public key locally, new users attempting to establish connection to the server will successfully connect, assuming the user did not turn off trust on the first use, believing it to be trustable. The window between a successful attack and the recovery might still cause a large number of users to blindly trust and accept the attacker's rogue key and metadata. A properly coordinated targeted attack could still be fatal.

3.2 Potential DoS Attack

Docker yields a hard fail if the signature of an image could not be verified (i.e., failed to contact the Notary server). This could also be leveraged by attackers who are at a privileged position in the network to block the Notary server or in the event of a DoS attack on the Notary server itself. As a result, if the verification fails, a system administrator might forego the signature verification if he/she is under the pressure of time. Although this is against the intended behavior of the design (i.e. hard fail means do not continue at all costs), organizational requirements and/or human conditions would override this control.

4 DECENTRALISING THE TRUST

In this section, we propose two potential decentralized solutions: BitTorrent-like and Blockchain-based to Docker Content Trust. Ideally, we need to build a decentralized distributed trust system with security features similar to Notary's use of multiple keys in a hierarchy.

4.1 System Architecture

We present a system architecture as shown in Figure 1. Each user publishes his/her Docker image to a DHT (Distributed Hash Table) network, similar to content distribution (publishing) in the DHT network (Xu et al., 2009). Each Docker image is a key-value store, where key is a hash of Docker image, and value is a combination of TUF metadata, tagging key and offline key. The published Docker image is verified by its TUF, tagging key and offline key. If a Docker image is not verified by its value, it cannot be published to the DHT network. In addition, for a Docker image without its TUF, tagging key and offline key, it is valid if it is found in the DHT network.

Based on the system architecture, two decentralized approaches are proposed as follows.

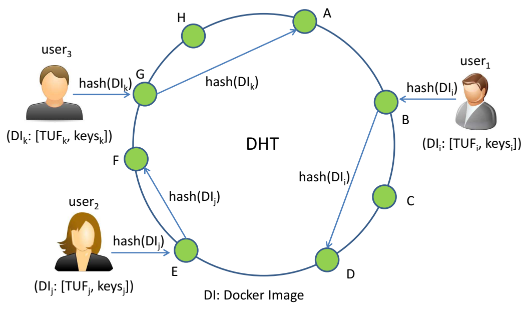

Figure 1: System Architecture.

4.2 BitTorrent-like Solution

The first approach uses a technology similar to BitTorrent called InterPlanetary File System (IPFS), a distributed file system that synthesizes successful ideas from previous P2P systems, such as DHTs and BitTorrent (Benet, 2014). IPFS can be viewed as a single swarm BitTorrent or a global FTP server running on P2P network. It can protect data integrity and availability by breaking it into pieces and share it among peers. To achieve this, Merkle directed acyclic graph (DAG) is employed to ensure the pieces are resistant to tampering (Benet, 2014). With IPFS, we can publish a file or a whole folder with subdirectories and files. However, the file or folder is only mirrored when any peers request for data. The published file or directory is given a hash that is its unique address for users to access the data. When users want to access this data, they can either issue "ipfs get $< hash >$" command or access it through an IPFS gateway on their browsers.

The gateway acts like a HTTP proxy that retrieves the files via IPFS protocol and serves it over HTTP. While the gateway is usually launched locally when IPFS is ran as a daemon, IPFS official website hosts a gateway for usability. It is important to note that files on IPFS are immutable. There is no method to delete or replace a file. We can only add files that will generate a new unique hash. This immutability property is valuable in security context as files cannot be replaced silently by the publisher. With this protocol, we can push the TUF metadata of a Docker image onto the network with the exception of timestamp metadata as we no longer have an actor to sign the timestamp since the files are decentralized. In other words, we do not have a solution to guarantee the data freshness. However, we can guarantee that the metadata are unchanged ever since it was published, and the metadata to verify the integrity and identity of the Docker images is resistant to DoS.

A typical workflow after integrating this solution would be: 1) Docker pull command is issued, 2) image is downloaded to local storage, 3) download metadata files for image through IPFS, and 4) verify downloaded image with the metadata files. Docker needs to be modified to automate steps 1) to 4). However, for a proof-of-concept, we could build a separate tool to perform steps 3) and 4) and leave steps 1) and 2) to built-in Docker functions. Currently there are two issues with this approach. Firstly, Docker images are uniquely identified by name and tag but IPFS data are identified by hashes. Therefore, we need to figure out a solution to map these two unique IDs so that on the user's end, he/she only needs the Docker image unique identifier to verify the image. Secondly, as already mentioned, we need to figure out an alternative solution to guaranteeing the data freshness.

4.3 Blockchain-based Solution

The second approach explores how to publish the metadata files to an immutable ledger, the blockchain. Currently, two most popular blockchains are Bitcoin's and Ethereum's. While it is possible to store non-transactional data on these blockchains, it is highly not recommended for many reasons (Matzutt et al., 2016). Therefore, this narrows the possibility of implementing metadata storage on blockchain to two approaches: 1) start an entirely new blockchain that supports data storage, and 2) use blockchain to store the pointer to data stored off-blockchain.

As mentioned in previous section, the strength of blockchain comes from its "democratic" system. Implementing a new blockchain, especially a private one, dilutes the value proposition of a blockchain being decentralized and distributed if it is controlled by a single entity. The value proposition of a blockchain grows with the number of nodes as this defines how immutable the ledger is. Hence, implementing a new blockchain is not a viable solution as without the nodes, the miners, the financial transactions (providing incentive for miners) the blockchain is just a complex implementation of a database. While it is possible, at this point, implementing a new blockchain as a solution is just not practical.

In the second approach, we explore using existing blockchain as a Notary to back our off-blockchain storage. For such implementation to work, the off-blockchain storage must be designed in such a way that the data are stored in a hash pointer data structure in order to produce a "chaining" effect. The resultant benefit is resistant to mutability as to change any data in the middle of the chain would require rewriting the chain following the item being changed. A Merkle-based tree or list is commonly employed to achieve this. It is no coincidence that both blockchains and IPFS all use Merkle-based data structure

Table 1: Parameters used in experiments.

Parameters	Setting
# of Docker images	$10^5, 2\times10^5, \cdots, 10^6$
# of servers	10, 20, $\cdots$,100
# of bits of hash function	384

it is the reason the data is immutable. In our case, we can store the metadata in the off-blockchain storage and use an existing blockchain as a stamp or proof-of-authenticity.

More research needs to be carried out in order to design an architecture of the off-blockchain storage possibly implementing a Merkle-based data structure. For the solution to be complete, the off-blockchain storage needs to also be decentralized. It might be viable to use the BitTorrent-like solution as the off-blockchain storage. Similar to the solution mentioned in previous subsection, an alternative way would also be needed to guarantee the freshness of data.

5 PERFORMANCE EVALUATION

In this section, we present performance evaluation using detailed simulations based on synthetic dataset. We have developed a detailed event-driven blockchain simulator to validate and evaluate our design decisions and choices. Note that due to space limit, we do not present experimental results for the BitTorrent-like solution. Parameters used in experiments are shown in Table 1. All simulation experiments are conducted on a Linux server with a Intel(R) Core(TM) 2 Duo E6750 2.66GHz processor and 32GB of RAM, running 64-bit Ubuntu 14.04.

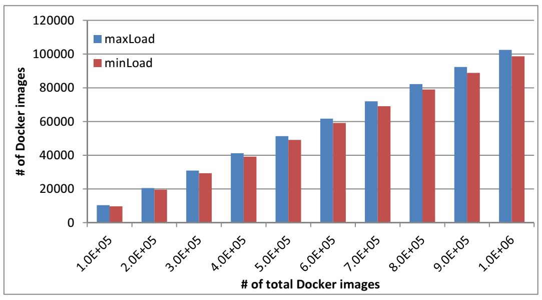

Figure 2: 10 servers with varying 10^5 to 10^6 Docker images.

Firstly, we present an experimental result for 10 servers with varying 10^5 to 10^6 Docker images, as shown in Figure 2. We give a definition of *Load Factor* as follows: $LoadFactor = \frac{maxLoad}{minLoad}$. From Figure 2, we can see that the average *Load Factor* is 1.05. The minimum *Load Factor* is only 1.04 when there are 10^6 Docker images while the maximum *Load Factor* is 1.06 when there are 10^5 Docker images. Our blockchain-based solution is scalable for 10 servers with different scales of Docker images.

We second illustrate load distribution for 10 servers with 10^6 Docker images, as shown in Figure 3. The *Load Factor* is only 1.04. We illustrate an experimental result for 10 - 100 servers with 10^6 Docker images in Figure 4. As shown in Figure 4, the maximum *Load Factor* is only 1.08 when there are 70 servers, while the minimum *Load Factor* is 1.03 when there are 20 servers. The average *Load Factor* is only 1.06. The blockchain-based approach is scalable for 10 - 100 servers with 10^6 Docker images.

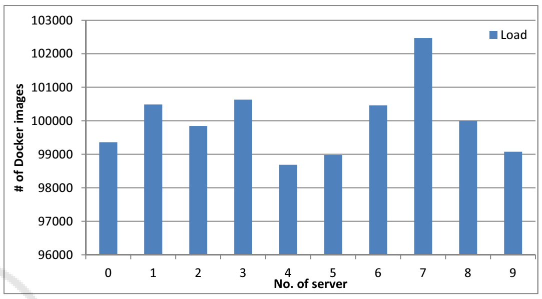

Figure 3: Load distribution for 10 servers with 10^6 Docker images.

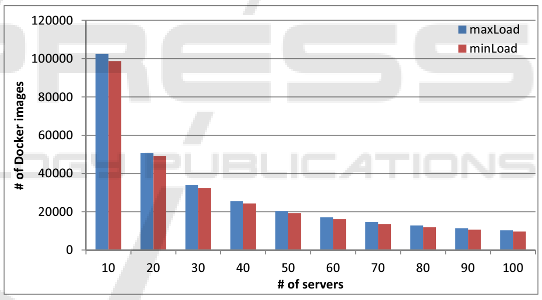

Figure 4: 10 - 100 servers with 10^6 Docker images.

6 RELATED WORK

6.1 Decentralized Distributed Systems

BitTorrent (BT), a P2P file sharing platform is one example of such a system. Traditionally, BT requires a tracker to function. The tracker acts as a central address book listing who have the file. However, in "trackerless" BT, a DHT is employed to allow each peer to act as a tracker. A DHT node is used to contact another node to retrieve info of peers who has the file a client requests. This effectively remove the need for a central tracker as long as the client trying to download the file can find peers who can provide information to populate the DHT.

Another popular example of such a system is Bitcoin. Bitcoin is a P2P platform to allow spending of the crypto-currency (BTC) without requiring a central authority (e.g., a financial institution) to oversee or validate the transaction (Nakamoto, 2008). In the Bitcoin platform, a user can have one or more wallets storing the BTC. Although in practical, there is no actual storage of coins involved. This is due to how Bitcoin is implemented.

6.2 Blockchain

Bitcoin is powered by the blockchain technology, a public ledger of every transaction made on the platform. A transaction is actually a transfer of coins signed with the recipients public key. Each coin is associated with an address and a transaction is simply a trade of coins from one address to another (Pilkington, 2016). A wallet on the other hand is actually the cryptographic key-pair (private-public key). Therefore, this key-pair can be used to trace how much spending a wallet has made or how much money it has received to derive the balance left in a wallet.

A blockchain with one node is basically a simple linked list data structure. The ingenuity of blockchain is when multiple nodes are involved to form a decentralized distributed system. Every node in the system has a copy of the entire blockchain. No central authority is needed to verify the authenticity of a copy of the blockchain (Brito and Castillo, 2013). When a transaction is made, it is broadcasted to the network, where the mining nodes add them to the block they are creating. The completed block will then be broadcasted to the network where the network will agree to add it to their copy of blockchain based on consensus. Blockchains can be utilized as smart contracts, which facilitate and enforce the negotiation of a contract in the IoT (Xu et al., 2017).

7 CONCLUSION AND FUTURE WORK

A TUF-based decentralized implementation carries the benefit of protection against a single point of failure and DoS attacks. To a certain extent, it also removes the need of a central authority to manage trust and therefore the user would not need to trust a single central authority but leave to a decentralized trustless system in place to manage the trust. However, such a solution as mentioned in this paper is not without its own issues and more research is needed to strengthen the idea and mitigate these issues. A final solution is still in the process of being explored that could either be BitTorrent-like or with combination of a blockchain. The decentralized trustless system if successfully implemented, could be used for more than just for verifying Docker images, but on any other data types. By conducting performance evaluation through extensive trace-driven simulations, experimental results illustrate the scalability and efficiency of the blockchain-based solution.

REFERENCES

Arumugam, R. V., Xu, Q., Shi, H., Cai, Q., and Wen, Y. (2014). Virt cache: Managing virtual disk performance variation in distributed file systems for the cloud. In *CloudCom*, pages 210–217.

Benet, J. (2014). Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*.

Brito, J. and Castillo, A. (2013). *Bitcoin: A primer for policymakers*. Mercatus Center at George Mason University.

Bui, T. (2015). Analysis of docker security. *arXiv preprint arXiv:1501.02967*.

Datadog (2016). 8 surprising facts about real docker adoption - datadog. Retrieved from https://www.datadoghq.com/dockeradoption/.

Khandelwal, S. (2016). Dirty cow critical linux kernel flaw being exploited in the wild. Retrieved from http://thehackernews.com/2016/10/linux-kernel-exploit.html.

Matzutt, R., Hohlfeld, O., Henze, M., Rawiel, R., Ziegeldorf, J. H., and Wehrle, K. (2016). Poster: I don't want that content! on the risks of exploiting bitcoin's blockchain as a content store. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1769–1771.

Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2.

Mnica, D. (2015). Introducing docker content trust. Retrieved from https://blog.docker.com/2015/08/content-trust-docker-1-8/.

mrled (2017). No way to disable trust-on-first-use for 'docker pull' with content trust #342. Retrieved from https://github.com/docker/notary/issues/342.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *URL: http://www.bitcoin.org/bitcoin.pdf*.

Pilkington, M. (2016). Blockchain technology: principles and applications. *Research Handbook on Digital Transformations*.

Samuel, J., Mathewson, N., Cappos, J., and Dingledine, R. (2010). Survivable key compromise in software update systems. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 61–72.

TUF-spec (2017). The update framework specification. Retrieved from https://raw.githubusercontent.com/theupdateframework-/tuf/develop/docs/tuf-spec.txt.

Xu, Q., Aung, K. M. M., Zhu, Y., and Yong, K. L. (2016). Building a large-scale object-based active storage platform for data analytics in the internet of things. *The Journal of Supercomputing*, 72(7):2796–2814.

Xu, Q., Aung, K. M. M., Zhu, Y., and Yong, K. L. (2017). *A Blockchain-based Storage System for Data Analytics in the Internet of Things*. To appear in "New Advances in the Internet of Things".

Xu, Q., Shen, H. T., Cui, B., Hou, X., and Dai, Y. (2009). A novel content distribution mechanism in dht networks. In *International Conference on Research in Networking*, pages 742–755. Springer Berlin Heidelberg.