DOI: 10.3969/j.issn.1673-5188.2015.03.009
http://www.cnki.net/kcms/detail/34.1294.TN.20150522.0920.001.html, Published online May 22, 2015

Crawler for Nodes in the Internet of Things

Xuemeng Li, Yongyi Wang, Fan Shi, and Wenchao Jia
(Department of Computer Science, Electronic Engineering Institute, Hefei 230037, China)

Abstract

Determining the application and version of nodes in the Internet of Things (IoT) is very important for warning about and managing vulnerabilities in the IoT. This article defines the attributes for determining the application and version of nodes in the IoT. By improving the structure of the Internet web crawler, which obtains raw data from nodes, we can obtain data from nodes in the IoT. We improve on the existing strategy, in which only determinations are stored, by also storing downloaded raw data locally in MongoDB. This stored raw data can be conveniently used to determine application type and node version when a new determination method emerges or when there is a new application type or node version. In such instances, the crawler does not have to scan the Internet again. We show through experimentation that our crawler can crawl the IoT and obtain data necessary for determining the application type and node version.

Keywords

crawler; local storage; nodes; Internet of Things

1 Introduction

With the fast development and increasing popularity of the Internet of Things (IoT), more and more devices are being used in everyday life and are being incorporated into the Internet. Such devices are also called nodes. Because these nodes are exposed to the Internet, they are not as safe many of the owners and users of these nodes think. The ability to connect to, communicate with, and remotely manage an incalculable number of networked, automated devices via the Internet has become pervasive. As we become increasingly reliant on intelligent, interconnected devices in everyday life, protecting billions of these devices from intrusion and interference has become a serious issue. Unauthorized intrusions can compromise personal privacy or even public safety [1]. Vulnerability is related to the application and many of the affected devices as possible in order to give warnings and manage the problem. In this paper, we describe a crawler for nodes in the IoT. This crawler is based on local storage. By collecting and storing information about the nodes' features, we can determine the application and version of individual nodes locally and conveniently.

ZoomEye [2] and Shandon [3] are two mature search engines that enable web users to search for application type and version. Using a distributed web crawler, ZoomEye collects information from Internet nodes all over the world. The ZoomEye interface enables the user to search for application, version, location, open port, and so on. The nodes from which this information is collected include websites and devices. Shodan, by contrast, is only focused on IoT device. It is used to expose vulnerabilities in routers, switches, and industrial control systems [2] and is often seen as a valuable tool for hackers. Shodan can be used to detect just about anything on the Internet, such as printers that can be controlled remotely; open, accessible web cameras; and other unsecured devices.

A popular strategy for determining the node application and version is first to crawl the Internet and then directly use the downloaded information. Then, the determinations are stored, and the raw information is abandoned. If there is a new strategy for determining the application and node version or if there is a new node application or version, the only thing that can be done it to scan the Internet a second time in order to retrieve the raw information again. This significantly increases scanning costs. We improved the existing web crawler by putting the NoSQL database to use. Our proposed distributed web crawler can retrieve fingerprint information in the IoT. The raw information returned by crawler is stored in the NoSQL database and used to determine the node application and version. The NoSQL database has high storage capacity, which the crawler demands, but does not occupy much of the system. Because it can search big data efficiently, the NoSQL database might be helpful for searching through the innumerable results returned by the web crawler [3], [4].

In section 2, we introduce MongoDB and traditional web crawler. In section 3, we describe the overall system structure, crawler design, data structure, local storage, and design of the database. In section 4, we experiment on our proposed crawler and show that it is efficient enough to satisfy the system requirements.

2 Related Works

2.1 MongoDB

The company 10gen develops the MongoDB, which is a doc-

This research work is supported by the ZTE Corporation and University Joint Research Project under Grant No. CON1307100001, and the National High Technology Research and Development Program of China under Grant No. 2013AA013602.

ument-oriented NoSQL database, not a traditional relational database. Although it is non-relational, MongoDB is faster, more expansible, and has more useful than a relational database [5]. MongoDB has many more functions than a relational database, including sorting, secondary indexing, and range searching [6]. Mongo DB has the following features:

- binary JavaScript object notation (BSON) for data storage. This increases the speed of index page traversal.
- non-relational storage and support for sharding. With sharding, big data is automatically divided into small data blocks that are then stored in an appropriate server. Although separate, user searching and combining of results can be done in a highly efficient way, and servers can linearly increase expandability and performance.
- flag_id as the only flag of the document. The value of this flag can be automatically assigned by the database or assigned by users themselves.
- a combination of key and value as the means of storing data. This is a loose-storage solution that makes inserting data into MongoDB easier than inserting data into a relational database. In this way, a user does not need to define the detailed table structure in advance, which is necessary in a relational database.
- no support for transactions. MongoDB sacrifices transactional support for easy use, high speed, and expandability.

2.2 Principle of the Traditional Web Crawler

A web crawler is a computer program used to downloading the source page of a website. Web crawlers are widely used in Internet search engines or for managing web page caches [7]. Crawling usually starts from one URL or a collection of URLs. Crawlers store these URLs permanently in a queue. Guided by a priority principle, a web crawler selects one URL from the URL queue and downloads the corresponding web page. If there are any other URLs in the web page, crawler extracts these URLs from the page after it has been downloaded. These extracted URLs are also stored in the queue. The downloading and extracting process loops until the web crawler is turned off, and downloaded pages are stored in the server database.

3 System Design

3.1 Background Requirements

The system must have a web crawler and database storage. As much as possible, the web crawler must crawl nodes on the IoT and grab information that is as complete as possible. Information stored locally in the database must be easy to understand, convenient to search, and useful for determining the application type and node version. At the same time, the crawling depth and width of the web crawler must be guaranteed.

A distributed web crawler is necessary to ensure high crawling speed and in-depth crawling. Web crawlers are now very cooperative with each other, and a scheduling relationship can be established between multiple web crawlers.

Traditional web crawling technology must also be improved so that traditional web crawlers can be used to grab specific information from nodes in the IoT. The information from these nodes can be used to determine the application type and node version in the future work.

MongoDB is necessary to store the collected information locally. Because the web crawler is distributed and MongoDB does not support transactions, independent MongoDB interfaces must to be designed for database reading and writing. In this way, MongoDB remains consistent.

Local storage is necessary for storing the information downloaded by the web crawler. Local storage enables the application type and node version to be determined locally, and scan time can be saved.

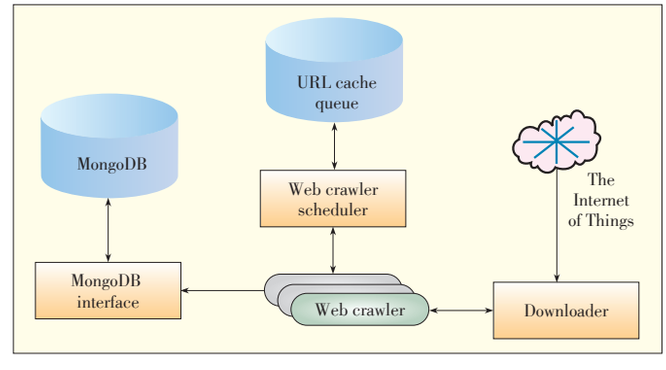
3.2 Overall Design

The system structure that best satisfies the previously mentioned requirements is shown in **Fig. 1**.

Within this system, the web crawler scheduler selects a URL from the URL cache queue, which has been sorted, and assigns the URL to a free web crawler. The web crawler crawls the web and downloads information from web pages or pages returned by devices. At the same time, the downloader also deletes duplicate useless webpages. Finally, the web crawler delivers the downloaded information to the MongoDB interface, which converts the information to BSON format and stores it. The web crawler scheduler, URL cache queue, and the web crawler can inherit the ones in mature crawler, so they are not covered in this paper.

3.3 Downloader Design

The IoT can be divided into public IoT and special IoT. In the former, all devices are connected. Public IoT covers one whole area of administration and connects to the public Internet. Public IoT is regarded as a true social information infrastructure [8]. Each sensor and entity connected to the IoT corresponds to one web source defined by a web page URL and can be accessed by HTTP protocol. All these web sources can



▲Figure 1. Overall system structure.

be expressed as HTML pages and are usually regarded as sensor pages or entity pages. These pages contains the type of the nodes, reading of the nodes, and some unstructured information [9]. Devices that can only be accessed with administrator privileges will return the administration web pages or error pages. These pages can also be used as the fingerprint of the device. By downloading the information included in these web pages, we can determine the application type and node version.

Our proposed downloader is different from a traditional downloader in three ways:

- Our proposed downloader scans more ports. The downloader of the traditional web crawler mainly scans ports 80 and 8080 whereas the downloader of our web crawler aimed at the IoT also scans ports such as 21, 22, and 23.
- The content downloaded by our proposed downloader is different. The downloader of a traditional web crawler downloads texts, pictures, files, and so on whereas the content downloaded by the web crawler of our proposed system only includes information useful for determining the application type and node version.
- Before our downloader downloads information, it has to determine whether the node belongs to the IoT or not.

The content downloaded by our web crawler aimed at the IoT must be useful for determining the application type and node version. After analyzing existing fingerprint-judging software, we realized that there was much more information that could be used to determine the application type and node version than we initially thought. We need to find out as much as possible about the node's fingerprint as accurately as we can and consider the load of the web crawler and capacity of MongoDB. We also need to download the URL address, header of the HTML page, body of the HTML page, banner information, and the geographic location. The downloader should also remove duplicates of the pages and complete cleaning of the content. The system needs to be capable of incremental crawling from time to time to obtain complete and accurate node data.

3.4 Local Storage of Node Data

Raw data from nodes is commonly used to determine the application type and node version but is not stored. The downloaded data is used as soon as it is downloaded, and after a determination has been made, only the application type and node version are stored in the database. However, a problem emerges when a new method is introduced for determining application type and node version or if there is a new application type or node version. To maintain the integrity of the database, devices should be scanned a second or even third time to obtain data, then the new approach is used to judge these devices. This "directly using without storing" method wastes scanned data and creates more scanning workload.

In terms of storing data, the system must avoid repetitive work. Our system stores the raw information downloaded by the downloader but stores it separately from the determinations. While the crawler is crawling devices, the data returned is directly passed to MongoDB and stored locally. The application type and node version are determined locally and independently. In the future, when derminations are being made about devices, regardless of whether or not the determination is made successfully, the raw information will be kept in MongoDB. If a new method for determining application type or node version emerges or if there is a new application type or node version, the only solution is to iterate through the database and obtain the fingerprint data of scanned devices.

3.5 Data Structure and Design of MongoDB

3.5.1 Data Structure

The data downloaded by the downloader must be stored in MongoDB in a certain format. After removing duplicates, the data is converted into BSON format by MongoDB interface and then stored in the database. The data to be downloaded includes URL address, header of the HTML page and so on.

3.5.2 Design of MongoDB

The collection obtained by MongoDB is similar to the table of a relational database. However, the definition of the collection is not as strict as that of a table. The MongoDB collection only comprises similar elements. Each data item in the collection is called a document. In our system, we create one document for every IP address. The fixed data structure of every document is shown in **Table 1**. Because MongoDB does not strictly define the collection, we can improve the downloader according to the background requirements. The structure of MongoDB does not need to change obviously. The design of MongoDB is shown in **Fig. 2**.

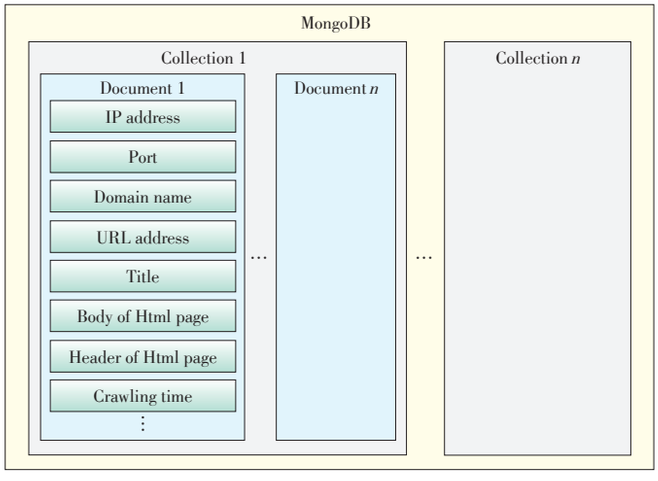
4 Realization of the Key Technology

4.1 Dividing Web

For *N* web pages, by using some certain dividing rule, the

▼Table 1. Data structure

Field	Data type	Description
ipAddr	string	IP address
Port	string	Port
Domain	string	Domain name
UrlAddr	string	URL address
Title	string	Title
htmlBody	string	Body of HTML page
htmlHead	string	Header of HTML page
HTTPheader	string	HTTP header
UpdateTime	datetime.utc	Time of crawling converted into utc timestamp
Location	string	Geographic location of nodes



▲Figure 2. Design of MongoDB.

collection G of web pages to be downloaded is divided into M (where $M > N$) subsets: $\{G_1, G_2, \ldots, G_M\}$. If $G_1 \cup G_2 \cup \ldots \cup G_M = G$, $G_1 \cap G_2 \cap \ldots G_M = \varphi$, and $|G_i| \approx |G_j|$, then G is called a division of N. Reasonable division of web pages prevents agents belonging to the system from crawling the same pages and increases the overall efficiency of web page crawling. The system uses dynamic distribution. At the start of a crawling job, the system distributes 1 to N subsets of web pages to N agents for crawling. Once an agent has finished crawling, the system distributes the next subset of web pages to the crawler. This distribution lasts until the end of the crawling job. To increase the efficiency of web crawling, the crawler should crawl web pages as close to it in network as possible. Therefore, a threshold timeout value is set for every agent. During crawling, if the time to connect to the web server exceeds the timeout threshold, the crawler will give up trying to connect. The agent will keep these URL addresses in its own list, and after a specified period, will pass the list to the system scheduler. This scheduler will re-schedule the nodes that the crawler could not connect to.

The advantage of dynamic distribution is that the load can be dynamically balanced and the agent is not turned down mistakenly [10].

4.2 Agent Synergy

Division of web pages requires the synergy of every agent in the system. To adapt to the strategy of web page division, our system synergizes with a scheduler. Each agent passes URL addresses that have been newly crawled to the scheduler, which distributes them in a uniform way. At the same time, the scheduler re-schedules nodes that the crawler could not connect to. Agent synergy is useful for controlling the division of web pages but overloads individual nodes and causes single-node failure [10].

An exchange approach is used to deal with links that cross the subset of the division. If an agent finds a URL that does not belong to its division subset, it passes the URL address to the crawler responsible for it.

4.3 Incremental Crawling

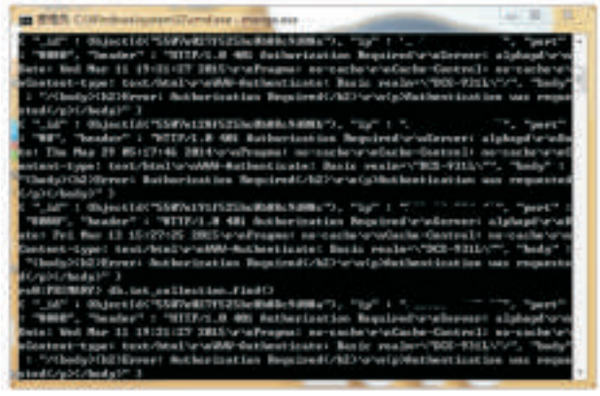
To ensure the integrity of the fingerprint database, incremental crawling is required. The easiest method of incremental crawling is to crawl pages again every specified period. However, doing this requires huge effort. Web pages are updated at different times, and some pages may not be updated for a long time. Therefore, the update period for web pages must be considered in the incremental crawling strategy. Among nodes in the IoT, there are important nodes whose accuracy must be guaranteed. As in [11], we categorize the nodes in the IoT as very important, important, general, or not important. The attribute of incremental crawling is the weighted average of the quantized importance and the update frequency. The incremental crawling attribute of all the nodes in the IoT is calculated to obtain the average. If a node has a higher-than-average incremental crawling attribute, the system crawls that node three times faster than average. If a node has a lower-than-average incremental crawling attribute, the system crawls that node at a speed 0.8 times the average.

5 Experiments and Conclusion

The performance of our web crawler under experimentation was affected to an extent by network bandwidth and hardware. We used Windows 7; internal storage was 2 GB; hard drive capacity was 250 GB; and network bandwidth was 100 Mbps. If the thread count of the distributed web crawler for the IoT is set to 10, 1085 URL address can be accessed in ten minutes. The downloaded information from 114 of these URL addresses belonged to nodes in the IoT. We used mature software to determine the application type and node version from this downloaded information. Fifty-six nodes were determined correctly, 49 of which included the application type.

After the web crawler had crawled the Internet and downloaded information from nodes over a period of time, we tested the local storage. MongoDB containing the raw crawling data was put into use in this experiment. We selected vulnerability CVE-2015-2049 published on 2 March 2015 in the China National Vulnerability Database [12]. This vulnerability in a D-Link DCS-931L remote wireless cloud camera with firmware 1.04 or earlier is an intermediate risk. Unrestricted file upload vulnerability in the camera allows a remote unauthenticated user to execute an arbitrary code by uploading a file with an executable extension [13]. In the previous identification for fingerprint, the application type and node version related to this vulnerability were not determined. To monitor the posture of the vulnerability, the application type and version need to be determined. First, we need to find fingerprint used for the determination. After information has been collected from the Internet,

the fingerprint is defined by the keyword SCS-931L in the WWW-Authenticate field of the HTTP header. After constructing a query statement for MongoDB and searching for the results in the raw information database, several results were returned (**Fig. 3**). The time cost for this search was less than 2 s,



▲**Figure 3. The search result of D-Link DCS-931L.**

which is much faster than crawling the Internet and collecting information a second time.

References

[1] *Security in the Internet of Things* [online]. Available: http://www. Windriver.com/whitepapers/security-in-the-internet-of-things/wr_security-in-the-internet-of-things.pdf
[2] *The instruction to Shodan* [online]. Available: http//: drops.Wooyun.org/1tips/2469
[3] Y. Gu, S. Shen, and G. Zheng, "Application of NoSQL database in web crawling," *International Journal of Digital Content Technology and its Applications*, vol. 5, no. 6, pp. 261—266. 2011.
[4] H. Dong, A. Wu, Q. Wu, and X. Zhu, "A novel distributed web crawling approach based on Mongodb," *International Journal of Advancements in Computing Technology (IJACT)*, vol. 5, no. 6, pp. 794—801. 2013.
[5] H. David, P. Eelco, M. Peter, and H. Tim, *The Definitive Guide to MongoDB, Second Edition*. Beijing, China: tsinghua university press, 2015, pp. 4—8.
[6] H. Li, *Design and Implementation of Crawler System for Public Feelings on Internet*. Xia Men, China: Xia Men University, 2014.
[7] P. Zhao, *Design and Implementation of Distributed Books Web Crawler System*. Cheng Du, China: Southwest Jiaotong University, 2014.
[8] S. Shen, Y. Mao, Q. Fan, P. Zong, and W. Huang, "The concept model and architecture of the internet of things," *Journal of Nanjing University of Post and Telecommunication (Nature Science)*, vol. 30, no. 4, pp. 3—8, 2010.
[9] Z. Wang, Q. Pan, and T. Xing, "Survey on real-time search engine for entities of internet of things," *Application Research of Computers*, vol. 28, no. 6, pp. 2001—2010, 2011.
[10] X. Xu, W. Zhang, H. Zhang, and B. Fang, "WAN-based distributed web crawling," *Journal of Software*, vol. 21, no. 5, pp. 1067—1082, 2010. doi: 10.3724/SP.J.1001.2010.03725.
[11] X. Su, *The Research, Implement on Technology of Distributed Web Crawler*. Harbin, China: Harbin Institute of Technology, 2006.
[12] *Vulnerability summary for CVE-2015-2049* [online]. Available: https//:web.Nvd. Nist. gov/view/vuln/detail? vulnId=CVE-2015—2049
[13] *Cross-site request forging loophole of D-Link-DCS-931L* [online]. Available: http//:www.cnvd.org.cn/flaw/show/CNVD-2015-01375

Manuscript received: 2015-04-22

Biographies

Xuemeng Li (benqer@126.com) received her BS degree in computer science from Electronic Engineering Institute. She is currently a graduate student at the Electronic Engineering Institute, Hefei. Her research interest is computer security.

Yongyi Wang (rose_1203@yeah.net) is a professor of computer science at the Electronic Engineering Institute. He receives his graduate degree at the Electronic Engineering Institute. His research interest is computer security.

Fan Shi (shif00@gmail.com) received his master's degree in computer science from Electronic Engineering Institute. He now works in Electronic Engineering Institute. His research interests include networks and search engine.

Wenchao Jia (jiatoday2013@163.com) received his master's degree at the Electronic Engineering Institute. He is currently a PhD candidate there. His research interests include networks and big data.