Fine-Grained Dissection of WeChat in Cellular Networks

Qun Huang[1], Patrick P. C. Lee[1], Caifeng He[2], Jianfeng Qian[2] and Cheng He[2]
[1]Department of Computer Science and Engineering, The Chinese University of Hong Kong
[2]Noah's Ark Lab, Huawei Technologies
{*qhuang,pclee*}*@cse.cuhk.edu.hk,* {*hecaifeng,qianjianfeng,hecheng*}*@huawei.com*

***Abstract*—WeChat is one of the most popular mobile messaging applications worldwide. However, due to the proprietary nature of WeChat, its characteristics and performance impact on cellular networks remain largely unexplored. This paper presents the first measurement study that dissects real-world WeChat traffic in a cellular network. We build ChatDissect, a protocol inference tool that infers the unique protocol formats and semantics of WeChat in a fine-grained manner. ChatDissect enables us to distinguish WeChat and its specific tasks from general network traffic traces. As a case study, we collect a real-world dataset from a commercial 3G cellular network in China and use ChatDissect to identify around 150K WeChat users with 16GB of WeChat payloads. We unveil the signatures, server architecture, and workflows of WeChat, and further analyze the activities of the extracted WeChat traffic.**

I. Introduction

WeChat, developed by Tencent Holdings based in China, is one of the most popular mobile messaging applications worldwide. It includes various functionalities such as instant messaging, real-time chatting, and social networking, and supports various message types such as text, voices, pictures, and videos. By August 2014, WeChat reportedly has around 438.2 million active users worldwide, including 100 million users outside China, and accounts for almost 50% of Internet resources among social networks in China [4].

The popularity and diversity of WeChat raise a lot of open questions regarding user activities, service usage, and traffic patterns. However, only few studies (see Section VII) have addressed these questions, and they are limited in scope and scale. Characterizing WeChat is challenging due to its proprietary nature and the lack of its protocol specifications. Thus, it is complicated to distinguish WeChat from network traffic that comprises a mix of applications, not to mention to classify the specific functionalities inside WeChat.

In view of this, we leverage protocol inference to infer both formats and semantics of WeChat protocols, with a primary objective to perform *fine-grained* dissection of WeChat. By fine-grained, we mean that we can (i) distinguish WeChat traffic from network traffic traces, (ii) classify different WeChat functionalities, (iii) unveil WeChat workflows and its architecture, and (iv) characterize WeChat traffic dynamics.

To this end, we present the first measurement study of WeChat traffic in a cellular network. We make the following contributions:

- We design *ChatDissect*, a protocol inference tool that performs fine-grained dissection of WeChat. ChatDissect builds on existing protocol inference techniques, such as the longest common substring technique (e.g. [15], [18]) and the Voting Experts algorithm [6], to extract the signatures of WeChat that specify both the formats and semantics of WeChat protocols.
- As a case study, we collect real-world traces from a 3G operational cellular network in a city of China. We use ChatDissect to identify around 150K WeChat users with 16GB of WeChat traffic.
- We unveil the signatures, server architecture, and workflows of WeChat. Such information, to our knowledge, has not been explored before.
- We further analyze the user and service/task activities of WeChat traffic, and present the key observations.

The benefits of our measurement study are two-fold. First, for cellular network operators, due to the ever-increasing usage of mobile messaging applications, understanding the traffic behaviors of WeChat, or mobile messaging applications in general, is critical for better resource provisioning and network planning. Second, for WeChat and mobile messaging application developers, they can use the measurement results to enhance their applications for better user experience.

The rest of the paper proceeds as follows. Section II presents an overview of WeChat via controlled experiments. Section III presents the ChatDissect design. Section IV describes our real-world dataset that drives our evaluation. Section V analyzes the WeChat protocol derived from ChatDissect, and presents its signatures, architecture, and workflows. Section VI characterizes the WeChat traffic in our traces. Section VII reviews related work, and Section VIII concludes this paper.

II. A Glance at WeChat

We provide a high-level overview of WeChat from a measurement perspective. We use controlled experiments to extract clean WeChat traffic, from which we identify the key WeChat characteristics that guide our later measurement study.

A. WeChat Services

WeChat supports different types of functionalities. We call the operation of each functionality a *task*, and collectively call the tasks of the same type a *service*. Our measurement study focuses on the following five services:

- **Instant messaging**: It relays messages among users in store-and-forward mode. Examples of instant messages include texts, voices, pictures, and videos.
- **Real-time chatting**: It supports real-time communication of voices and videos among users. It can operate in either full-duplex VoIP mode or half-duplex walkie-talkie mode.

978-1-4673-7113-1/15/$31.00 ©2015 IEEE

- **Social networking**: It forms a social network among users. It implements a sharing platform called *Moments*, in which users can post texts, upload photos, and share Internet resources with their friends. Users can also comment or click "like" on the posts.
- **Media access**: It enables users to access various multimedia resources, which can be WeChat-specific (e.g., subscription articles published by WeChat users) or shared from Internet (e.g., a video from a third-party site).
- **System**: The above four services are triggered by users. The WeChat system itself also generates messages, such as heartbeats and querying news.

WeChat also supports other services, such as friend finding, payments, and games. However, such services only account for a small proportion of traffic in our measurement study (see Section VI), so we do not consider them in this work.

B. Controlled Experiments

We conduct controlled experiments to collect clean WeChat traffic, which serves as ground truths for our study.

1) Methodologies: We first overview the setup of our controlled experiments. We deploy two smartphones, one Android and one iPhone, and connect both smartphones to the Internet via wireless. We collect traffic traces generated by the smartphones and parse them using Wireshark [19]. We install a WeChat client on each smartphone and perform different tasks of the four user-triggered services described in Section II-A. We repeat each operation multiple times. We test two WeChat client versions: 4.5 and 5.0, which are released in February 2013 and August 2013, respectively (at the time of the writing, the latest WeChat version is 6.1).

To reduce noise, we disable all foreground applications on the smartphones. However, the smartphones still generate background traffic (e.g., iOS periodically probes Apple servers), which cannot be easily disabled. Thus, we perform post-processing on the captured traces to filter out any unwanted traffic. Specifically, we manually look up the hostnames of the server IP addresses in the traces. We eliminate any packet from the traces if its server IP address does not belong to WeChat.

After filtering, we obtain a total of 22K IP packets with 12MB of WeChat traffic. We see both TCP and UDP packets at the transport layer. We also identify by Wireshark that some TCP packets are HTTP and some UDP packets are DNS at the application layer, while the application protocols of the remaining packets are unknown. Thus, we categorize WeChat traffic into four *flow types*: *W-HTTP*, *W-DNS*, *W-TCP*, and *W-UDP*, which correspond to HTTP packets, DNS packets, non-HTTP TCP packets, and non-DNS UDP packets, respectively.

We group packets into flows by 5-tuples (i.e., source/destination IP addresses, source/destination ports, and protocol). We do not distinguish the direction of packets, but instead aggregate the uplink (i.e., phone-to-Internet) and downlink (i.e., Internet-to-phone) traffic with the same five tuples into one flow. We further determine the start and end points of each flow. For W-HTTP and W-TCP, we use SYN and FIN/RST to determine the start and end points of a flow, respectively. Also, if the interval between two TCP packets of the same five tuples exceeds a TCP timeout (30 minutes in this work), we treat them as two individual flows. For W-DNS and W-UDP, we employ a UDP timeout (1 minute in this work) to separate flows. We call a flow *active* if it has started but not yet terminated.

TABLE I
FLOW TYPES OF OPERATIONS IN CONTROLLED EXPERIMENTS.

Service	Task	W-HTTP	W-UDP	W-TCP
Instant Messaging	Send/receive texts			√
	Send/receive pictures	√		√
	Send/receive voices			√
	Send/receive videos			√
Real-time Chatting	VoIP video	√	√	√
	VoIP voice	√	√	√
	Walkie-talkie	√	√	√
Social Networking	Refresh	√		√
	View a post			
	View a user			
	Post texts			
	Post photos			
	Delete posts			
	Comment a post			
	Click "like"			
Media access	Media access	√		√
System	Not user-generated			√

[1] Some tasks are not included in the controlled experiments. However, our later analysis addresses a more extensive set of WeChat tasks through careful signature extraction.

2) Results: Table I details the tasks of different services, and describes the flow types observed in each task. We ignore W-DNS flows, whose occurrences depend on whether the DNS mappings have been cached. We provide some preliminary findings here.

Flow types vary: Tasks are implemented with different flow types. Also, the same task may be implemented with more than one flow type.

W-HTTP flows are short-term: W-HTTP flows are non-persistent and most of them have short durations less than 10s. We can easily determine the task of each W-HTTP flow by examining the HTTP request and response of the same 5-tuple.

W-TCP flows encapsulate multiple tasks: We find that multiple user-triggered tasks are encapsulated in a single persistent W-TCP flow. Thus, one challenge of our analysis is to decompose a W-TCP flow into different tasks. Payloads in W-TCP flows are encrypted, although we expect that protocol information is embedded in the first few bytes of a subset of packets [13].

System messages are found in W-TCP flows: In addition to user-generated traffic, we observe many traffic spikes within a W-TCP flow. These spikes are likely system tasks, such as heartbeats. A challenge is to separate the user-generated and system-generated traffic from a W-TCP flow.

Real-time chatting uses W-UDP flows: All W-UDP traffic that we capture is due to real-time chatting. We find that each real-time chat starts with a number of small-size and short-term W-UDP flows to multiple servers, and later uses one or two long-term W-UDP flows for chatting.

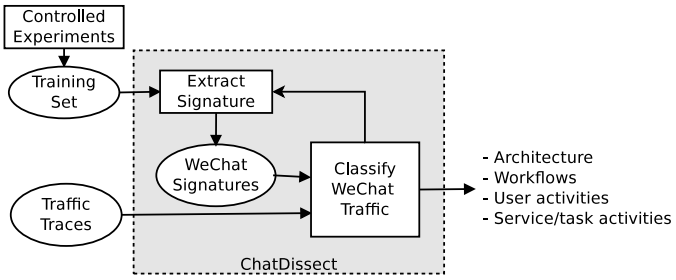


Fig. 1. ChatDissect architecture.

III. ChatDissect Design

We present ChatDissect, which specifically aims for *fine-grained* dissection of WeChat. Figure 1 outlines the ChatDissect architecture. Since WeChat is proprietary and its protocol specification is unknown to the public, the core component of ChatDissect is to analyze WeChat packet payloads and extract *signatures* that characterize the formats and semantics of the WeChat protocol messages. We assume that a packet payload is formatted as a sequence of *fields*, each of which is associated with a set of *values* that describe the semantics of the field. We define a signature as a field and its values.

ChatDissect uses the clean WeChat traffic captured from controlled experiments (see Section II-B) as the training set to extract signatures. Using the signatures as inputs, it parses traffic traces to distinguish WeChat traffic and classify WeChat tasks. To improve dissection accuracy, ChatDissect feedbacks the classification results to extract additional signatures that are not included in our controlled experiments but are found in traffic traces.

ChatDissect leverages and extends existing protocol inference techniques specifically for our WeChat analysis. We point out that our contribution is *not* to propose new methodologies, but instead we focus on how we properly combine existing methodologies in ChatDissect and make fine-grained dissection of WeChat possible.

The current ChatDissect design has the following limitations. Although ChatDissect can parse network traffic and perform WeChat dissection in real time, its signature extraction involves manual validation (see Section III-A4) as in many signature-based intrusion detection systems. Nevertheless, we expect that we only need to extract signatures infrequently (e.g., when WeChat upgrades its software). Also, ChatDissect operates at the byte-level granularity, and cannot distinguish bit-level fields. We validate that ChatDissect currently works effectively with WeChat versions 4.5 and 5.0, but it needs to be enhanced if the later version of WeChat uses bit-level fields.

A. Extracting Signatures

Recall that WeChat traffic comprises four flow types: W-DNS, W-HTTP, W-UDP, and W-TCP. Our goal is to extract signatures for each of the flow types. Since the flow types are inherently different, it is difficult to find a unified methodology to extract signatures for all flow types. Thus, ChatDissect is designed to extract signatures with two approaches. First, for W-DNS and W-HTTP flows, we directly inspect the DNS and HTTP fields, respectively. Second, for W-UDP and W-TCP flows, we conduct protocol inference through the following three steps: (i) segmenting payloads into candidate fields, (ii) categorizing candidate fields into different field types, and (iii) identifying *opcode* fields (i.e., the fields that identify WeChat tasks) and correlating their field values with specific WeChat tasks. We describe the first approach in Section III-A1, and the second approach in Sections III-A2 to III-A4.

TABLE II
Parameter settings in ChatDissect.

Parameter Description	Value
LCS for Field Keyword Extraction	
Minimum length of an extracted LCS	4
Minimum coverage of an extracted LCS	5%
VE Algorithm for Field Segmentation	
Length of sliding window	10
Minimum votes of a field boundary	6
Maximum length of a segmented field	4
Minimum coverage threshold of a segmentation result	10%
Heuristics for Field Type Inference	
Maximum consecutive occurrences for a request/response field	3
Minimum correlation coefficient for a length field	0.99
Maximum relative entropy of the distribution of opcode values	0.5
Heuristics for Opcode Correlation	
Maximum CV of inter-arrival time for a period task	0.2
Maximum time gap in which a task triggers an opcode (second)	1
Minimum ratio to determine a triggering	100

ChatDissect uses various threshold parameters, as summarized in Table II. We select the parameters based on either the default values in prior work or our experience.

1) Direct Field Inspection: ChatDissect extracts signatures for W-DNS and W-HTTP flows based on the DNS and HTTP documentations, respectively. One challenge is that both DNS and HTTP have an enormous number of fields, each of which can have diversified values. Thus, ChatDissect only examines the "representative" fields of DNS and HTTP that can uniquely identify W-DNS and W-HTTP flows, respectively, and extract keywords that "best" represent the field values.

Field selection: Selecting the representative fields for WeChat requires domain knowledge. Here, we consider a few fields that we believe can differentiate W-DNS and W-HTTP flows. Specifically, we make two observations for W-DNS and W-HTTP flows. First, they communicate with WeChat servers, and the hostnames of WeChat servers are included in payloads. Second, W-HTTP flows originate from the WeChat client, whose information is included in the User-Agent field.

Thus, we select the following representative fields: for DNS, we examine the queried hostnames in DNS requests; for HTTP, we examine five fields including Host, Referer, User-agent, Method, and URL in HTTP requests. The first three fields of HTTP are used for differentiating W-HTTP flows, while the fields Method and URL (which appear in the request line) are later used for classifying WeChat tasks.

Keyword extraction: Given the diversity of field values, we extract keywords that capture the important information of the values of the selected fields. For a field with only a few

possible values (e.g., the Method field in HTTP), each of the observed values is treated as a keyword; for other fields, we extract the keywords based on the *longest common substring (LCS)* approach, which has been used extensively in traffic classification (e.g., [15], [18]). The LCS approach identifies the best possible substring (a contiguous set of bytes) that characterizes most flows of a protocol. Specifically, we extract the LCS for every pair of observed strings. For example, we observe that the hostnames of WeChat servers differ slightly across geographic locations (e.g., "hkshort.weixin.qq.com" and "shshort.weixin.qq.com" for Hong Kong and Shanghai servers, respectively). Using the LCS approach, we obtain "short.weixin.qq.com". The LCS approach is often configured by two thresholds to select the most important LCSes [18]: (i) the selected LCS should have a minimum length, and (ii) the selected LCS should cover a minimum proportion of observed strings. We set the thresholds as 4 bytes and 5%, respectively.

2) Payload Segmentation: Given the lack of documentations, we cannot directly inspect the fields of W-UDP and W-TCP flows. Instead, we *infer* the fields by analyzing the payload characteristics. From our controlled experiments (see Section II-B), we observe that the payloads of W-UDP and W-TCP flows are encrypted, but we expect that protocol information is embedded in the first few bytes of payloads [13]. Thus, we choose to extract signatures from the first 16 bytes of every packet of each of the W-UDP and W-TCP flows.

We first partition each packet payload into different portions of contiguous bytes, such that each portion represents a field of WeChat with a high probability. ChatDissect extends one of the state-of-the-art payload segmentation techniques ProWord [21] for this purpose. ProWord adopts an unsupervised *Voting Experts (VE)* algorithm [6] to identify field boundaries based on statistical distributions. We further extend the VE algorithm of ProWord specifically for WeChat.

Basic VE algorithm of ProWord: We first describe the basic VE algorithm used by ProWord. The VE algorithm applies a sliding window of length L bytes to a packet payload. In each L-byte sliding window, it computes two entropies: (i) *word internal entropy*, which defines the entropy of words in the window, and (ii) *word boundary entropy*, which defines the entropy of the boundary at the end of words in the window. The VE algorithm then votes for two positions in each sliding window as candidate field boundaries, such that the string between the boundaries has a low word internal entropy value (i.e., it is likely a complete field) and the boundaries have high word boundary entropy values (i.e., they are likely boundaries). It finally selects the positions as the output field boundaries that (i) have more votes than adjacent positions and (ii) have at least a threshold number of votes. However, the basic VE algorithm is limited in two aspects for WeChat traffic. We elaborate the issues and explain how we extend the basic VE algorithm in the context of WeChat.

(i) Iterative VE algorithm: One of our observations is that the fields of WeChat often contain a small number of bytes, and some of them are even single bytes. We find that the basic VE algorithm is too conservative and produces too few field boundaries in WeChat payloads, leading to long fields. For example, if we choose the default values of ProWord [21] by setting $L = 10$ and the minimum vote threshold as 6, then more than 50% of W-TCP payloads have only one field boundary. We also test other parameters and see similar results. To address this problem, ChatDissect iteratively executes the VE algorithm to segment a long field until either all fields are shorter than a maximum field length threshold or any field cannot be further partitioned. In our case, we set the maximum field length threshold as 4 bytes.

(ii) Packet defragmentation: Another of our observations is that WeChat fragments a large W-TCP flow into small packets in the application layer. The protocol information is only included in the first few bytes of a subset of packets (we call them *protocol packets*), while the remaining packets (we call them *non-protocol packets*) have no protocol information. The basic VE algorithm only performs payload segmentation, but cannot separate protocol and non-protocol packets. Thus, we need to address the application-layer fragmentation issue. Note that the issue only appears in W-TCP flows, but not in W-UDP flows.

To distinguish between protocol and non-protocol packets, ChatDissect categorizes packets by how they are segmented by the VE algorithm. Packets with the identical segmentation result are put into the same group. If the proportion of packets of a group is larger than a minimum coverage threshold (which we set as 10%), then we treat all packets in the class as protocol packets; otherwise, the packets are treated as non-protocol packets.

ChatDissect then assembles non-protocol packets with the corresponding protocol packets. It first sorts the packets of each W-TCP flow by TCP sequence numbers and removes all retransmitted packets. Then it concatenates each non-protocol packet with the preceding protocol packet with the same direction (uplink or downlink). For each protocol packet and the following non-protocol packets that belong to the same W-TCP flow, we treat them collectively as a single flow (i.e., a long W-TCP flow can be further decomposed into multiple flows). Later, we identify the WeChat task for each decomposed flow.

3) Field Type Inference: Our previous payload segmentation step identifies protocol packets and divides each of their payloads into fields. We now examine the values of the fields and categorize them into different field types. In this work, we are interested in five field types of WeChat. (i) *Constant*: it takes a constant value and often represents a magic number in the protocol; (ii) *Sequence number*: it keeps packet ordering within a flow; (iii) *Request/response*: it indicates if the packet is a request or response. Note that a request may be originated from a server, such as news feeding; (iv) *Length*: it refers to the number of bytes of all packets in a decomposed flow; and (v) *Opcode*: it identifies a specific WeChat task.

We enumerate all fields in the protocol packets. For each field, we check it against different heuristics (in the order we present below) and attempt to assign each field with one of the above five types. (i) For the constant field, we check

if the observed values are identical among all packets. (ii) For the sequence number field, we check if the observed values are monotonic increasing. (iii) For the request/response field, we check if the field has exactly two values and if any consecutive occurrences of either value have length no larger than a maximum length threshold (which we set to 3). (iv) For the length field, we compute the correlation coefficient of the field values and the number of bytes, and check if the coefficient is larger than a threshold (which we set to 0.99). (v) Finally, we identify the opcode field, which often appears near the beginning of a payload, and has only few values. We examine the remaining unidentified fields, draw the distribution of the observed values of each such field, and compute the normalized entropy. The opcode field is chosen to be the first field from the beginning of a payload that has the normalized entropy less than an entropy threshold (which we set to 0.5). All remaining fields that do not belong to any of the above five types are marked as *unidentified*.

4) Opcode Correlation: Finally, ChatDissect correlates each opcode value (or opcode in short) with a WeChat task. Since WeChat tasks have varying characteristics, we exploit several types of extra information to help our correlation.

First, we correlate some opcodes with the traces collected in our controlled experiments (see Section II-B). For each task that we perform in controlled experiments, we extract the corresponding portion of the traces and identify the opcode. Note that this approach only identifies a small number of opcodes, and cannot identify the opcodes of the tasks that are not performed.

Second, we reverse-engineer the source code of WeChat client software and recover the opcodes. Our observation is that many tasks use different flow types for the same task in different platforms. For example, the task of sending a picture message uses the HTTP POST method in iPhone, and the URL field of the W-HTTP request is "/cgi-bin/micromsg-bin/uploadmsgimg". However, the same task uses W-TCP in Android, and its corresponding opcode is 9 by our trace correlation (see above). We find that the source code often includes both the URLs identified in W-HTTP flows and opcodes in W-TCP flows. Thus, we perform reverse engineering as follows. We start with the Android APK package of WeChat 4.5. We use the tools dex2jar [8] and JD-Gui [12] to disassemble the APK package into a collection of Java source code files. Then we search for each URL identified in our W-HTTP flows and find the corresponding opcode. The URLs can be obtained from the clean traces captured in controlled experiments or feedback results from the classified WeChat traffic (see Section III-B).

Finally, we examine the opcodes of the system tasks that are generated by the WeChat system rather than by users. We consider two types of system tasks: (i) periodic tasks (e.g., heartbeats) and (ii) background tasks triggered by another task (e.g., synchronization tasks triggered after a successful login). For periodic tasks, we calculate the coefficient of variation (CV) (i.e., the standard-deviation-to-mean ratio) for the inter-arrival times among all occurrences of an opcode in a flow. If the CV values of an opcode in all flows are less than a maximum CV threshold, we regard the opcode as a periodic task. For the triggered background tasks, we determine if an occurrence of task T triggers an occurrence of opcode O following the heuristic of Sherlock [1]. Specifically, we compute the probability that T is followed by O in a time interval less than a maximum task gap threshold (which we set as 1 second). If the ratio of the probability to the average arrival rate of T exceeds a minimum ratio threshold (which we set as 100), we treat opcode O triggered by task T.

B. Classifying WeChat Traffic

Given the input traffic traces, ChatDissect distinguishes WeChat traffic and classifies it into tasks. Specifically, ChatDissect first groups packets of the input traces by 5-tuple flows, and categorizes the flows as one of the four flow types: (i) DNS (i.e., UDP flows whose server-side ports are 53), (ii) HTTP (i.e., TCP flows that contain HTTP keywords such as "POST", "GET", and "HTTP"), (iii) non-DNS UDP flows, and (iv) non-HTTP TCP flows. We then check if they match the signatures for W-DNS, W-HTTP, W-UDP, and W-TCP, respectively. In a nutshell, we partition the packet payloads into fields, and check if the field values match the same signatures. In Section V, we elaborate the signatures of WeChat that we have identified.

In some cases, the signature matching procedure may have false positives. For example, some W-UDP flows only contain a single-byte constant field and a two-byte sequence number field as the signatures, which may be found in non-WeChat packets. To remedy this, we inspect every packet of a UDP flow, and ensure that the flow is a W-UDP flow only if all packets match WeChat signatures.

ChatDissect starts with the clean traces captured from controlled experiments to extract signatures. However, the clean traces may only cover a subset of WeChat tasks. Thus, we enable ChatDissect to exploit feedback information after parsing network traffic traces. Specifically, if a flow is classified as a WeChat flow and it accesses a WeChat server, we collect the packets of all the unclassified flows with the same server-side IP address and port number. We then apply the same signature extraction procedures to the collected packets. We also manually check the correctness of the new signatures.

We may need multiple rounds of feedbacks for signature extraction, until no more signatures are extracted. Nevertheless, based on our traces (see Section IV), our experience is that we only need to feedback our results *once* (i.e., after the first round of parsing the traces), and we can extract all possible WeChat signatures in the traces. In particular, the signatures obtained from the controlled experiments enable us to distinguish all WeChat traffic, and the signatures obtained from the feedback enable us to further identify tasks based on URLs (in W-HTTP flows) and opcodes (in W-UDP and W-TCP flows). We need further validation on the effectiveness of the feedback approach, and we pose it as future work.

IV. Dataset

Our measurement study is driven by real-world network traffic traces collected from a commercial 3G UMTS cellular

network in a city of China. Our dataset spans five three-hour periods in three different days of November 2013 (i.e., 15 hours of traffic in total). The collection periods represent the busy hours of network usage. It contains around 450 million raw IP packets, accounting for 185GB of data. We provide the statistics of the dataset when we perform evaluation on WeChat in Section VI.

We do not collect any control-plane subscriber information due to privacy concerns. Our analysis uses private IP addresses to distinguish users, as in previous work [17]. As private IP addresses may be reused, we assume that if a private IP has no data transmission for more than 60 minutes, its user session expires [17]. Based on this heuristic, we find that the dataset covers 310K users.

We discuss limitations of our dataset. First, our dataset was collected over a year ago (from the time of this writing), and it may not reflect today's usage. In particular, it only covers the WeChat traffic of versions up to 5.0. Second, most WeChat payloads are encrypted (although the protocol information remains), so we cannot identify some WeChat characteristics embedded inside the payloads, such as the friend lists of users. Finally, the scale of the dataset remains limited both spatially (e.g., a single collection point) and temporally (e.g., no full-day traffic). One important future work is to validate our findings with larger-scale and more up-to-date traces.

V. WeChat Protocol Dissection

In this section, we present the signatures, server architecture, and workflows of WeChat.

A. WeChat Signatures

We have extracted the signatures of WeChat from our traces in Section IV and use them to identify all WeChat traffic from the traces. We summarize the key characteristics of the signatures for different flow types. We plan to release the detailed signatures as a technical report accompanying the final version of the paper.

W-DNS and W-HTTP: We find that both flow types are associated with a small number of server hostnames. All server hostnames contain WeChat's Chinese aliases such as "weixin", "wx", and "mm" (a short form for "MicroMessenger"). We believe that these hostnames represent the servers belonging to the WeChat core architecture. Some of the servers (e.g., "*.weixin.qq.com") are responsible for performing the actual WeChat tasks. The others provide storage services for WeChat resources (e.g., photos) in the social networking service.

W-HTTP uses both POST and GET methods in requests. W-HTTP POST requests only access either of the two hosts: "short.weixin.qq.com" and "support.weixin.qq.com", and their URLs contain either of the two prefixes: "/cgi-bin/micromsg-bin/" and "/cgi-bin/mmsupport-bin/". The URLs with the former prefix are responsible for performing most WeChat tasks such as the instant messaging, social networking and system tasks. The latter prefix corresponds to the tasks for media access to the third-party sites. On the other hand, all W-HTTP GET requests are issued to retrieve resources. The W-HTTP GET requests always set the User-Agent field as "MicroMessenger" and configure the Referer field to contain WeChat-specific names such as "wechat" or "weixin". We can further identify the types of the resources by the hostnames and URLs. Specifically, the hosts "mmsns.qpic.cn", "wx.qlogo.cn", and "wx.qq.com" correspond to the servers storing WeChat-specific resources. The URLs to these hosts further reveal the detailed WeChat services that use the WeChat-specific resources. For example, the task of retrieving social network photos can be identified by the URL keyword "mmsns", which is a short form for "MicroMessengerSocialNetworkS". Apart from the tasks of retrieving WeChat-specific resources, we treat a W-HTTP GET request as accessing to third-party resources if its host is not included in our identified signatures.

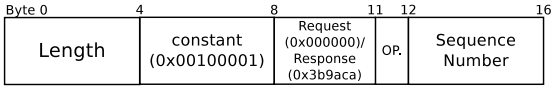


Fig. 2. First 16 bytes of W-TCP protocol packets.

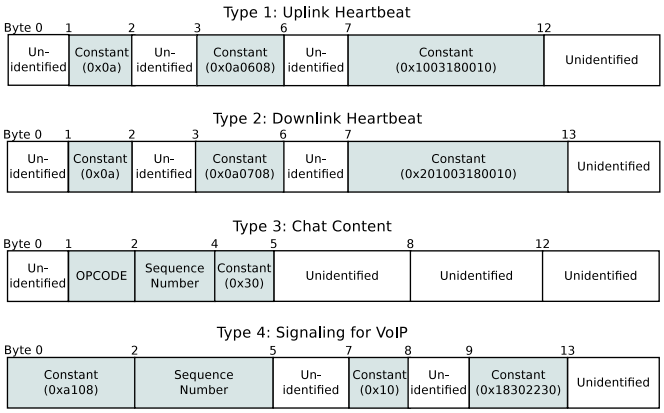


Fig. 3. First 16 bytes of W-UDP packets.

W-TCP: We find that W-TCP protocol packets can all be characterized by the first 16 bytes, whose field formats are depicted in Figure 2. Bytes 0 to 3 form a length field and specify the total number of bytes of a task (including all protocol and non-protocol packets); bytes 4 to 7 form a constant field and always have value `0x00100001`; bytes 8 to 10 are the request/response field, with `0x000000` indicating a request and `0x3b9aca` indicating a response; byte 11 is the opcode field; bytes 12 to 15 form a sequence number field that indexes each task in a W-TCP flow.

We have identified 126 opcodes for W-TCP flows and correlated each opcode with a specific task (see Section III-A4). Specifically, 18 opcodes are obtained through manual correlation with the traces collected from controlled experiments; 91 opcodes are associated through reverse-engineering the WeChat client package; the remaining 17 opcodes are associated with the system tasks including the periodic and background tasks.

W-UDP: Since UDP is connectionless, every W-UDP packet keeps the protocol information in its first 16 bytes. We identify four signature types for W-UDP, as shown in Figure 3. The first two signature types are the uplink and

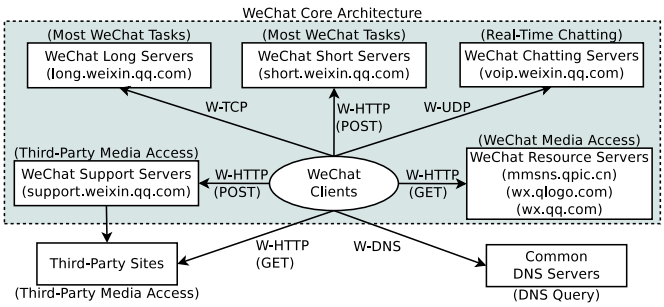


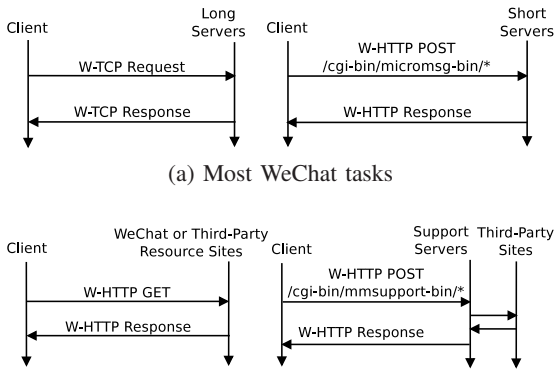
Fig. 4. WeChat server architecture.

downlink heartbeats for the real-time chatting, both of which contain three constant fields: the first two constant fields are in byte 1 and bytes 3 to 5; the third constant fields have different lengths according to our segmentation results, as the downlink heartbeats have one more byte. The third signature type represents real-time chat content, including VoIP and walkie-talkie. Its signature format is defined by four bytes: byte 1 is the opcode; bytes 2 to 3 form a sequence number field; byte 4 is a constant field. The fourth signature type represents the signaling of real-time chats. It has three constant fields: bytes 0 to 1, byte 7, and bytes 9 to 12. In addition, it contains a 3-byte sequence number field from bytes 2 to 4.

B. WeChat Server Architecture

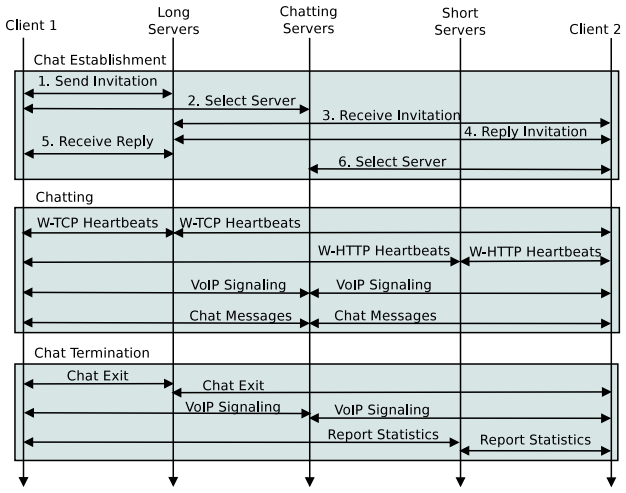
Our inferred WeChat signatures also give us a view of WeChat architecture, as shown in Figure 4. The core architecture is composed of a set of server clusters. The long servers "long.weixin.qq.com" and the short servers "short.weixin.qq.com" process most WeChat tasks using the persistent W-TCP flows and non-persistent W-HTTP flows, respectively. They are responsible for all the instant messaging and system tasks, as well as most tasks in the social networking service. They are also used for some real-time chatting functions, such as sending a real-time chat invitation. The chatting servers "voip.weixin.qq.com" are mainly responsible for sending/receiving real-time chatting messages. The resource servers are responsible for storing WeChat resources (e.g., photos in social networks), while WeChat clients may also access resources from third-parity sites. Furthermore, the support servers "support.weixin.qq.com" relay traffic between WeChat clients and third-party sites.

Figure 5 further shows how a client communicates with the servers to perform a task. Figure 5(a) presents two workflows for most WeChat tasks: a client either (i) sends a W-TCP request in which the opcode indicates the specific task to the long servers, or (ii) sends a W-HTTP POST request, with the prefix "/cgi-bin/micromsg-bin" in the URL field, to a short server to perform a task. Figure 5(b) shows how WeChat clients access resources. WeChat provides two methods for this purpose: a client either (i) directly sends a W-HTTP GET request for the resources, or (ii) sends a W-HTTP POST request to the WeChat support servers, which relay the request to third-parity sites. The most complicated workflow is real-time chatting, which is shown in Figure 5(c). Real-



(a) Most WeChat tasks

(b) Media access, including WeChat and third-party resources



(c) Real-time chatting

Fig. 5. WeChat workflows.

time chatting involves three flow types. To establish a real-time chatting session, WeChat clients use W-TCP to send and reply invitations. The clients also select a chatting server using W-UDP in the establishment. During chatting, clients send chatting contents of both VoIP and walkie-talkie, as well as VoIP signaling messages, through the selected chatting server. The clients also generate heartbeats using W-TCP (for long servers) or W-HTTP (for short servers). On chat termination, the clients send VoIP signaling messages and chatting statistics to the chatting and short servers, respectively.

C. Summary

We summarize our findings. First, WeChat employs a set of server clusters to handle different types of tasks. Their performance requirements are different. For example, the tasks performed by the long servers (e.g., instant messaging) and the support servers (e.g., media access) are sensitive to the user-perceived delay, while the tasks performed by the chatting servers require sufficient bandwidth.

Also, most WeChat tasks are performed via persistent W-TCP flows or non-persistent W-HTTP flows, while real-time chatting tasks are performed via W-UDP flows. We further study the traffic characteristics of each flow type in Section VI.

TABLE III
SUMMARY OF THE NUMBER OF USERS AND TRAFFIC VOLUME.

	Date	Time	# of Users (K)		Traffic Volume (GB)	
			Total	WeChat	Total	WeChat
1	2013-11-01	8-11am	51.94	30.23	39.14	3.34
2	2013-11-01	6-9pm	72.22	38.70	49.08	3.94
3	2013-11-04	8-11am	65.15	25.74	28.68	2.66
4	2013-11-04	6-9pm	65.67	32.01	37.35	3.77
5	2013-11-05	8-11am	50.83	24.59	30.58	2.30
Total			305.81	151.27	184.84	16.00

VI. WeChat Traffic Characteristics

In this section, we study the characteristics of WeChat traffic from two perspectives: (i) user activities and (ii) service and task activities. Table III summarizes the statistics of the number of users and traffic volume in our dataset. Among all 310K users (i.e., private IP addresses) in our dataset, 150K (49.46%) of them carry WeChat traffic. These users generate 1.35M (8.31%) WeChat flows and 35M (7.98%) WeChat packets, accounting for 16GB (8.66%) of traffic volume.

A. User Activities

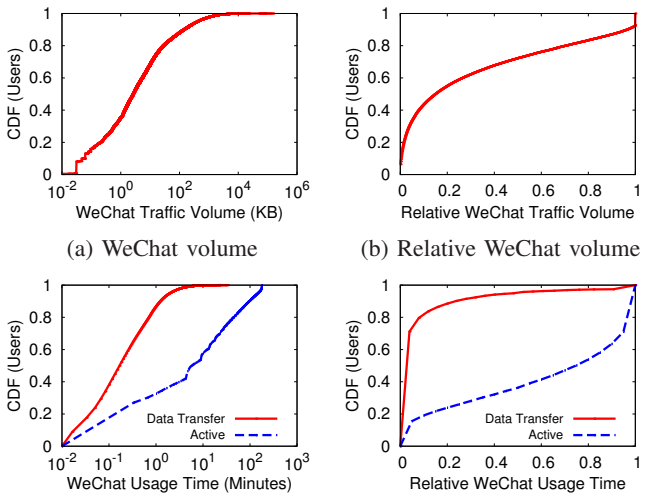
We analyze the WeChat usage among the 150K WeChat users. We plot the cumulative distribution functions (CDFs) of the number of WeChat users versus different metrics.

We first consider the WeChat usage in terms of traffic volume. Figure 6(a) plots the CDF for the WeChat traffic volume generated by a user. We observe some heavy WeChat users who generate a large amount of WeChat traffic. For example, the top 150 (0.1%) users generate 10MB of traffic each and the top 20K (13%) users generate nearly 100KB of traffic each. Figure 6(b) also plots the CDF for the relative WeChat traffic volume of a user to the total volume of the same user. WeChat accounts for more than 50% of traffic volume in 45K (30%) users. 11K (7%) users include only WeChat traffic.

We further analyze the WeChat usage time. We divide time into one-second bins and count the number of bins that involve WeChat. We consider two types of usage time: (i) a user sends or receives WeChat traffic (labeled as "Data Transfer") and (ii) a user has an active WeChat flow (labeled as "Active") (see Section II-B for the definition of an active flow). Figure 6(c) plots the CDFs for the usage time. We observe some long-term users. For example, the top 10K (7%) and 38K (26%) users keep WeChat active for over 2 hours and 30 minutes, respectively. Figure 6(d) plots the CDFs for the relative usage time of a user to the total usage time of the same user. It shows that around 71K (46%) users keep an active WeChat flow for most of the time (above 80%). However, only a small fraction of time is actually used to transfer WeChat traffic. Figure 6(c) shows that the time with WeChat traffic transfers is less than one minute in 131K (87%) users; Figure 6(d) also shows that only 27K (18%) users spend more than 10% of time in transferring WeChat traffic. This implies that WeChat users are silent most of the time.

B. Service and Task Activities

We analyze WeChat usage across different services and tasks. We are interested in the five services and their corresponding tasks as described in Section II-A. From our dataset, we identify the tasks in such services, as detailed in Table IV. The tasks cover over 99.8% of all WeChat traffic in our dataset.



(a) WeChat volume (b) Relative WeChat volume

(c) WeChat usage time (d) Relative WeChat usage time

Fig. 6. User activities in terms of volume and usage time.

1) Volume Analysis: Table IV details the number of WeChat users and the traffic volume of various tasks categorized by different services. We see that the usage of the services diverse a lot. For example, only 60 (0.04%) users employ the real-time chatting service, but the system service is commonly found in the WeChat traffic of nearly all users (over 99%).

We also find that the downlink traffic dominates over the uplink traffic. Specifically, the total volume of the downlink traffic is 14.5GB (90.61%), while that of the uplink traffic is only 1.5GB (9.39%). We correlate this observation with user behaviors. We find that the downlink-consuming tasks have more users than the uplink-consuming tasks. For example, the social networking service covers nearly 51K (34%) users. Only 1,842 (1.19%) users post a text and 1,324 (0.85%) users upload photos, yet all social networking users refresh posts and nearly 25K (16.45%) users download photos. The task of downloading social networking photos generates 7.94GB (49.60%) of traffic, among which the uplink traffic only has volume 182MB (1.11%).

The difference between uplink and downlink traffic is also huge in the instant messaging service, although the same type of instant messages (e.g., text) has similar numbers of sending users and receiving users. For example, 5,029 (3.25%) users send voice messages and 7,259 (4.69%) users receive voice messages. However, receiving voices generates 345MB (2.10%) traffic, which is significantly larger than that generated by sending voices, whose volume is 92MB (0.56%). One main reason for the difference is due to the group communication pattern in the instant messaging service, as a user sends one message to multiple users.

2) Flow Characteristics: We now measure the flow characteristics of WeChat tasks. We focus on the tasks implemented with W-HTTP and W-TCP because the two flow types cover most WeChat tasks.

TABLE IV
WECHAT SERVICE AND TASK USAGES.

Service	# of Users (ratio to total users)	Task	# of Users (ratio to total users)	Volume (MB) (ratio to total WeChat traffic volume)					
				Total		Uplink		Downlink	
Instant Messaging	20,269 (13.08%)	Send texts	15,679 (10.12%)	45.009	(0.27%)	26.608	(0.16%)	18.401	(0.11%)
		Receive texts	18,964 (12.24%)	86.796	(0.53%)	35.138	(0.21%)	51.658	(0.32%)
		Send pictures	1,962 (1.27%)	25.988	(0.16%)	24.813	(0.15%)	1.175	(0.01%)
		Receive pictures	2,593 (1.67%)	131.516	(0.80%)	1.964	(0.01%)	129.552	(0.79%)
		Send voices	5,029 (3.25%)	92.403	(0.56%)	87.196	(0.53%)	5.207	(0.03%)
		Receive voices	7,259 (4.69%)	345.152	(2.10%)	3.917	(0.02%)	341.235	(2.08%)
		Send videos	55 (0.04%)	3.182	(0.02%)	3.138	(0.02%)	0.044	(<0.01%)
		Receive videos	69 (0.04%)	30.238	(0.18%)	0.124	(<0.01%)	30.114	(0.18%)
		Send emoji	1,216 (0.78%)	0.921	(0.01%)	0.718	(<0.01%)	0.203	(<0.01%)
		Receive emoji	1,814 (1.17%)	2.247	(0.01%)	0.921	(<0.01%)	1.326	(<0.01%)
Real-time Chatting	60 (0.04%)	Control messages	60 (0.04%)	0.289	(<0.01%)	0.186	(<0.01%)	0.103	(<0.01%)
		Chatting heartbeats	55 (0.04%)	31.097	(0.19%)	0.516	(<0.01%)	30.581	(0.19%)
		Walkie-talkie content	2 (<0.01%)	2.392	(0.02%)	1.419	(0.01%)	0.973	(0.01%)
		VoIP content	32 (0.02%)	776.066	(4.73%)	423.309	(2.58%)	352.757	(2.15%)
Social Networking	52,490 (33.88%)	Get header images	14,430 (9.31%)	452.322	(2.76%)	35.258	(0.22%)	417.064	(2.54%)
		Get photos	25,480 (16.45%)	8134.363	(49.60%)	181.931	(1.11%)	7952.432	(48.49%)
		Post a text	1,842 (1.19%)	4.119	(0.03%)	2.101	(0.02%)	2.018	(0.01%)
		Post a photo	1,324 (0.85%)	130.774	(0.80%)	127.898	(0.78%)	2.876	(0.02%)
		Refresh moment	52,490 (33.88%)	288.892	(1.76%)	43.764	(0.27%)	245.128	(1.49%)
		View a user	7,133 (4.60%)	57.763	(0.35%)	7.476	(0.04%)	50.287	(0.31%)
		View a post	3,358 (2.16%)	9.349	(0.06%)	1.517	(0.01%)	7.832	(0.05%)
		Comment a post	5,375 (3.47%)	15.458	(0.09%)	4.249	(0.02%)	11.209	(0.07%)
		Tag a post	323 (0.21%)	0.181	(<0.01%)	0.115	(<0.01%)	0.066	(<0.01%)
		Other operations	122 (0.08%)	1.512	(0.01%)	0.451	(<0.01%)	1.061	(<0.01%)
Media Access	16,390 (10.58%)	Media access	16,390 (10.58%)	4511.506	(27.51%)	95.263	(0.58%)	4416.243	(26.93%)
System	153,507 (99.09%)	W-TCP heartbeats	121,824 (78.64%)	30.422	(0.19%)	17.862	(0.11%)	12.560	(0.08%)
		Report to servers	67,366 (43.48%)	185.390	(1.13%)	157.544	(0.96%)	27.846	(0.17%)
		Query servers	72,778 (46.97%)	457.361	(2.79%)	94.142	(0.57%)	363.219	(2.22%)
		Connect to servers	115,665 (74.66%)	171.193	(1.04%)	70.788	(0.43%)	100.405	(0.61%)
		DNS	146,268 (94.41%)	177.996	(1.09%)	65.025	(0.40%)	112.971	(0.69%)
		Server push news	55,091 (35.56%)	168.734	(1.03%)	21.975	(0.13%)	146.759	(0.90%)
Other	N/A	N/A	< 150 (0.1%) each	28.230	(0.17%)	15.043	(0.09%)	13.187	(0.08%)
Total	N/A	N/A	154,916 (100.00%)	16398.861	(100.00%)	1539.974	(9.39%)	14858.887	(90.61%)

TABLE V
AVERAGE SIZES PER REQUEST AND PER RESPONSE FOR INSTANT MESSAGING TASKS.

Task	Average Size per Request (bytes)		Average Size per Response (bytes)	
	W-TCP	W-HTTP	W-TCP	W-HTTP
Send texts	340	N/A	221	N/A
Receive texts	196	408	358	580
Send pictures	4,935	5,216	209	401
Receive pictures	256	454	38,645	41,693
Send voices	2,191	2,823	183	372
Receive voices	276	511	27,143	37,248
Send videos	16,405	17,465	196	384
Receive videos	231	430	53,189	58,733
Send emoji	406	627	183	370
Receive emoji	N/A	331	N/A	523

The tasks implemented with W-HTTP and W-TCP exhibit the request-response fashion. We observe 3.11M requests and 2.93M responses in our traces. As an example, Table V lists the average sizes per request and per response for instant messaging tasks, which use W-HTTP or W-TCP for communication. We see that most of the requests and responses have size less than 1KB. On the other hand, we also observe a few large requests and responses. These tasks are used to send or receive multiple media resources. For example, receiving video messages has an average response size of 58KB.

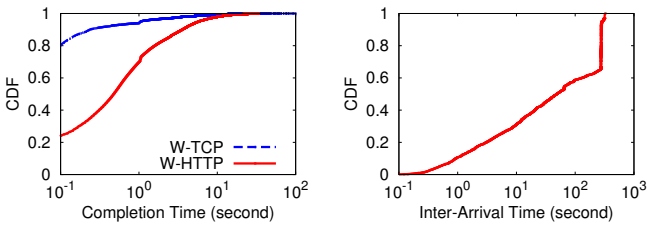
We see that the tasks implemented with W-HTTP have larger request and response sizes than those implemented with W-TCP, since the binary format of W-TCP is more compact in volume than the text format of W-HTTP. For example, W-TCP uses a single-byte opcode to indicate tasks, while W-HTTP specifies its tasks in the URL field, which has tens of bytes.

We further analyze the time patterns of both W-HTTP and W-TCP implementations. Figure 7(a) shows the CDFs of number of tasks versus the completion times of both implementations. We see that W-TCP has shorter completion time, with more than 80% of W-TCP tasks are completed within 0.1s. On the other hand, W-HTTP has longer completion time, mainly because the tasks require to establish a TCP connection via 3-way handshake for the communication.

Recall that a single W-TCP flow often encapsulates multiple tasks (see Section II-B). Figure 7(b) shows the inter-task time distribution in a W-TCP flow. We see that around 40% of cases have inter-task time around 300s, mainly because W-TCP flows generate periodic heartbeats every 300s. In general, the inter-task time varies a lot. The short inter-task time (e.g., less than 0.1s) corresponds to the system tasks, such as querying the servers for news after users connect to the servers; the long inter-task time (e.g., over 10s) corresponds to the user-generated tasks. For example, the time between sending two instant messages often takes tens of seconds.

C. Summary

We summarize our findings. First, WeChat is used by a significant portion of users (nearly 50% in our dataset). Many users keep persistent flows with WeChat servers for a long



(a) Task completion time (b) Inter-task time in W-TCP

Fig. 7. Time patterns of W-TCP and W-HTTP.

time, but only transfer a small amount of traffic. This may pose maintenance overhead on WeChat servers.

Second, our WeChat traces show a much higher volume of downlink traffic than that of uplink traffic (90.61% versus 9.39%). This suggests some potential optimizations for better user experiences. For example, WeChat developers can optimize their storage servers for read-intensive workloads, or leverage the content delivery network to reduce the response time and avoid redundant transfers.

Finally, WeChat employs two flow types to implement most of the tasks: persistent W-TCP flows and non-persistent W-HTTP flows. The W-TCP implementation has better performance as it has less traffic and faster completion time, but generates heartbeat messages to keep a flow persistent. How to choose between W-HTTP and W-TCP implementations needs further investigation.

VII. Related Work

There has been a plethora of studies on analyzing messaging applications. For example, Xiao *et al.* [20] characterize the traffic patterns of instant messaging applications such as AIM and MSN. Bonfiglio *et al.* [2] measure the performance, user activities and signaling overheads of Skype. On the other hand, WeChat is more complicated as it supports various types of tasks and messages (see Section II-A). Some studies analyze WhatsApp, another popular mobile messaging application. Church *et al.* [5] and O'Hara *et al.* [16] study the user activities of WhatsApp, mainly by user interviews and surveys. Fiadino *et al.* [9] study the flow-level characteristics of WhatsApp based on traces collected from a European nationwide network core, and the traces cover millions of chat flows, audio/content flows, and video flows. Liu *et al.* [14] study WeChat video messaging based on traces captured from mobile devices.

ChatDissect exploits payload features to infer the protocol format of WeChat. Protocol inference has been well studied in network traffic measurements. Many studies (e.g., [3], [7], [10], [11], [15], [18], [21]) also exploit machine learning techniques to examine the statistical features of packet payloads, and infer the format of protocol fields and extract payload signatures. However, existing machine learning approaches have not been applied for characterizing mobile messaging applications. ChatDissect makes the first attempt to apply machine learning (e.g., the Voting Experts algorithm) to infer the protocol format. It further analyzes the field semantics (e.g., opcodes) to classify WeChat tasks.

VIII. Conclusions

We present the first measurement study of WeChat in a cellular network. WeChat supports different types of tasks, messages, and flows. Thus, we build ChatDissect, which characterizes the formats and semantics of different WeChat tasks in a fine-grained manner. We use ChatDissect to study WeChat traffic collected in a real-world cellular data network. We present the WeChat server architecture and workflows, and characterize the user and service/task activities of WeChat. Our measurement study provides insights into better designing and managing mobile messaging applications in cellular networks.

Acknowledgments

This work was supported in part by GRF CUHK413711 from the Research Grant Council of Hong Kong and YB2014110015 from Huawei Technologies.

References

[1] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang. Towards Highly Reliable Enterprise Network Services via Inference of Multi-level Dependencies. In *Proc. of SIGCOMM*, 2007.

[2] D. Bonfiglio, M. Mellia, M. Meo, N. Ritacca, and D. Rossi. Tracking Down Skype Traffic. In *Proc. of INFOCOM*, 2008.

[3] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli. Revealing Skype Traffic: When Randomness Plays with You. In *Proc. of SIGCOMM*, 2007.

[4] By the Numbers: 20 Amazing WeChat Statistics. http://expandedramblings.com/index.php/wechat-statistics/.

[5] K. Church and R. de Oliveira. What's up with WhatsApp?: Comparing Mobile Instant Messaging Behaviors with Traditional SMS. In *Proc. of MobileHCI*, 2013.

[6] P. Cohen and N. Adams. An Algorithm for Segmenting Categorical Time Series into Meaningful Episodes. In *Advances in Intelligent Data Analysis*, 2001.

[7] W. Cui, J. Kannan, and H. J. Wang. Discoverer: Automatic Protocol Reverse Engineering from Network Traces. In *Usenix Security Symposium*, 2007.

[8] dex2jar. https://code.google.com/p/dex2jar.

[9] P. Fiadino, M. Schiavone, and P. Casas. Vivisecting Whatsapp Through Large-scale Measurements in Mobile Networks. In *ACM SIGCOMM Demo*, 2014.

[10] A. Finamore, M. Mellia, M. Meo, and D. Rossi. KISS: Stochastic Packet Inspection Classifier for UDP Traffic. *IEEE/ACM Transactions on Networking*, 18(5):1505–1515, 2010.

[11] P. Haffner, S. Sen, O. Spatscheck, and D. Wang. ACAS : Automated Construction of Application Signatures. In *ACM MineNet*, 2005.

[12] JD-gui. http://jd.benow.ca.

[13] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee. Internet Traffic Classification Demystified : Myths , Caveats , and the Best Practices. In *Proc. of CoNEXT*, 2008.

[14] Y. Liu and L. Guo. An Empirical Study of Video Messaging Services on Smartphones. In *Proc. of NOSSDAV*, 2014.

[15] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. M. Voelker. Unexpected Means of Protocol Inference. In *Proc. of IMC*, 2006.

[16] K. P. O'Hara, M. Massimi, R. Harper, S. Rubens, and J. Morris. Everyday Dwelling with WhatsApp. In *Proc. of ACM Computer Supported Cooperative Work*, 2014.

[17] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, and O. Spatscheck. Characterizing Radio Resource Allocation for 3G Networks. In *Proc. of IMC*, 2010.

[18] A. Tongaonkar, R. Keralapura, and A. Nucci. SANTaClass : A Self Adaptive Network Traffic Classification System. In *IFIP Networking Conference*, 2013.

[19] Wireshark. http://http://www.wireshark.org.

[20] Z. Xiao, L. Guo, and J. Tracey. Understanding Instant Messaging Traffic Characteristics. In *Proc. of ICDCS*, 2007.

[21] Z. Zhang, Z. Zhang, P. P. C. Lee, Y. Liu, and G. Xie. ProWord : An Unsupervised Approach to Protocol Feature Word Extraction. In *Proc. of INFOCOM*, 2014.