I, Zongjie Tu, hereby submit this original work as part of the requirements for the degree of Doctor of Philosophy in Computer Science & Engineering.

It is entitled:  
DHT-based Collaborative Web Translation

Student's name:  
Zongjie Tu

This work and its defense approved by:

Committee chair: Chia Han, Ph.D.

Committee member: Fred Annexstein, Ph.D.

Committee member: Anca Ralescu, Ph.D.

Committee member: William Wee, Ph.D.

Committee member: Xuefu Zhou, Ph.D.
DHT-based Collaborative Web Translation

A dissertation submitted to the
Graduate School
of the University of Cincinnati
in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

in the Department of Electrical Engineering and Computing Systems
of the College of Engineering and Applied Science

by

Zongjie Tu

B.E. Fudan University
M.S. Fudan University
M.S. University of Cincinnati

October 2016

Committee Chair: Chia-yung Han, Ph.D.
Abstract

Linguistic diversity on the web has stimulated demand for web translation. Online translation services backed by machine translation are able to perform in-place translation on a web page within seconds, giving a web user a general idea of what the web page is about.

Nowadays, peer-to-peer applications are adopting Distributed Hash Table (DHT), a serverless approach that distributes loads among all interconnected devices. Kademlia and its variant Mainline DHT have been widely adopted. DHT runs over an overlay network, where interconnected devices as well as keys are assigned node IDs output by a hash function. The value of a key is stored onto one node (or more nodes if needed) whose node ID is the closest to that of the key. Web applications are now in widespread use due to merits of cloud computing, such as cross-platform compatibility and server-centric software maintenance. Node.js unifies server-side and client-side coding and has thrived in recent years. Browserify enables web browsers to exploit existing Node.js modules in the colossal npm repository. WebRTC makes it possible for disparate web browsers to communicate on a real-time basis, paving the way for a browser-based peer-to-peer network where crowdsourced translation can be achieved. Technologies such as bookmarklet, userscript and browser extension empower Internet users to personalize their favorite websites on the fly.

The success of Google Translate can be attributed to responsiveness, acceptable accuracy, and integration with web browser. However, Google Translate suffers drawbacks in terms of data privacy, service availability, idiomatic translation, context sensitivity, personalization, minority
languages, transliterated text, etc. A DHT-based system for collaborative web translation is proposed to address those drawbacks.

Modules of the proposed system include embedded graphical user interface for translation, peer exchanging translation works with its counterparts in an overlay network through WebRTC, translations in volatile memory, non-volatile storage areas for data persistence, and optional servers that assist peer discovery and connection establishment. The proposed system is intended to function in both single-tab and multi-tab scenarios and is capable of synchronization between web browser tabs and peers. It adopts a three-level matching scheme for translation download, taking hyperlinks into account. A caching mechanism is devised to boost performance and lessen network traffic during synchronization. Human users are able to download existing translation from the overlay network, apply translation in their web browsers and upload each piece of translation (whether or not it is their original works) to the overlay network on a voluntary basis. The voluntary sharing mechanism along with inherent hashing of content in DHT helps protect user privacy. A signaling server has been devised to cluster users with similar language background and to get around Network Address Translation (NAT) and proxies, particularly in a multi-tab scenario. Security measures such as Transport Layer Security (TLS) have been taken to guard against common network attacks.

A prototype of the proposed system has been implemented. Experimental results showed that the prototype remained responsive in heavy-duty tasks and scaled to hundreds of peers.
This page is intentionally left blank.
Acknowledgements

I would like extend my heartfelt gratitude to my advisor, Dr. Chia-yung Han, for his mentorship throughout my graduate school journey. Dr. Han is not only an amiable instructor but also an enlightened scholar. His insightful perception of numerous academic topics in computer science has immensely deepened my understanding of the discipline and inspired me to explore new territory in research. Moreover, Dr. Han is a warmhearted friend I feel indebted to. He has repeatedly rendered help and advice to me ever since I started graduate student life. Without his guidance in academic study and campus life, I would not have been able to complete this dissertation.

I would also like to express my appreciation to Dr. Fred Annexstein, Dr. Anca Ralescu, Dr. William G. Wee, and Dr. Xuefu Zhou for taking the time to review my dissertation and for their patience in serving on my committee.

Finally, special thanks go to my family, who have always supported me both emotionally and financially while I pursue my doctorate.
# Table of Contents

1 Introduction ......................................................... 1

2 Background .......................................................... 4

2.1 Dictionary and Hash Table........................................... 4

2.2 Peer-to-peer Data Sharing Network.............................. 6

2.2.1 Serverless Peer-to-peer Technology: Distributed Hash Table (DHT)................................. 7

2.2.2 Overlay Network and DHT..................................... 7

2.2.3 DHT Case Study: Kademlia.................................... 10

2.2.4 Possibility of Incorporating Kademlia into the Web............................ 13

2.3 Web Application.................................................... 14

2.3.1 Cloud Computing................................................ 14

2.3.2 Conventional Web Applications.............................. 14

2.3.3 Thriving Node.js Server-side Web Applications..................... 15

2.3.4 WebRTC......................................................... 16

2.4 Augmented Browsing................................................ 17

2.4.1 Personalized Web Experience................................. 18

2.4.2 Bookmarklet.................................................... 18

2.4.3 Userscript...................................................... 19

2.4.4 Browser Extension............................................. 20

2.5 Crowdsourcing..................................................... 20
2.5.1 Definition of Crowdsourcing..............................................................................21
2.5.2 Evolution of Crowdsourcing Systems................................................................21
2.5.3 Crowdsourcing in Language Translation............................................................22

3 Problem Statement ...................................................................................................... 24

3.1 Web Translation........................................................................................................24
3.2 Case Study: Google Translate......................................................................................25
  3.2.1 Problems with Client-server Model and Centralized Data.................................27
  3.2.2 Translation Inaccuracy.........................................................................................29
  3.2.3 Limited User Collaboration..................................................................................30
  3.2.4 Restrictions on Submitted Translation...............................................................31
  3.2.5 Lack of Personalization.......................................................................................31
  3.2.6 Lack of Support for Multilingual Web Pages.....................................................31
  3.2.7 Lack of Support for Minority Languages and Dialects........................................32
  3.2.8 Overdependence on English as a Pivot Language.............................................32
  3.2.9 Poor Support for Ever-evolving Internet Slang..................................................32
  3.2.10 Lack of Support for Transliterated or Transcribed Text....................................33
  3.2.11 Wasted Web Traffic Owing to Identical Queries..............................................37
3.3 Significance of Research and Expected Contributions..............................................37

4 Proposed Solution: DHT-based Collaborative Web Translation ................................ 38

4.1 Prerequisites..............................................................................................................38
  4.1.1 Origin of a Web Page.........................................................................................38
  4.1.2 The localStorage Property..............................................................................39

vi
4.1.3 Web Browser Fundamentals

4.1.4 Locale

4.2 Methodology

4.2.1 Assumptions

4.2.2 Overview of Adopted Technologies

4.2.3 Architecture of the Proposed System (Single-tab Scenario)

4.2.4 Architecture of the Proposed System (Multi-tab Scenario)

4.2.5 Signaling Server in Multi-tab Scenario

4.2.6 Translation Download and Upload

4.2.7 User Roles and Peer Interaction with DHT

4.2.8 Data Format

4.2.9 Robustness

4.2.10 Local Screening

4.2.11 Context-sensitive Translation

4.2.12 Triggering Translation Synchronization

4.2.13 Data Import and Export

4.2.14 Preferred Locale

4.2.15 JSON Data Structure

4.3 Tackling Challenges

4.3.1 Server-assisted Connectivity

4.3.2 Incentives to Contribute

4.3.3 Browser Integration
4.3.4 System Responsiveness.................................................................76
4.3.5 Peer Clustering...........................................................................76
4.3.6 Extensibility..............................................................................77
4.3.7 Security.....................................................................................77
4.3.8 Privacy......................................................................................80
4.3.9 Expected Results.....................................................................80

5 Results..........................................................................................82

5.1 Translation GUI...........................................................................82
5.2 Secure Signaling Server..............................................................92
5.3 Evaluation....................................................................................94
  5.3.1 Testing Environment..............................................................94
  5.3.2 GUI Test..................................................................................95
  5.3.3 Scalability Test.......................................................................96
  5.3.4 Summary..............................................................................99

6 Conclusions and Future Work......................................................100

6.1 Conclusions..............................................................................100
6.2 Future Work...............................................................................101
  6.2.1 Supplement to Crowdsourced Moderation..........................101
  6.2.2 Paradigm for Distributed Knowledge Sharing...................101

Bibliography......................................................................................103
List of Tables

Table 1: Examples of Internet slang Google Translate fails to interpret correctly .................. 32
Table 2: URLs of tabs of Browser in a multi-tab scenario .................................................. 47
Table 3: Comparison of three implementation candidates .................................................. 76
Table 4: Specifications of the laptop used in evaluation .................................................... 94
Table 5: Software environment ......................................................................................... 95
List of Figures

Fig. 1: Google Translate applying in-place English-Chinese translation to the main menu of an English website.................................................................2
Fig. 2: Three entries of an English-Chinese dictionary as key-value pairs.....................4
Fig. 3: A hash table storing entries in Fig. 2....................................................................5
Fig. 4: An example of overlay network............................................................................8
Fig. 5: A Distributed Hash Table storing entries in Fig. 2 using CRC-32 as hash function.....9
Fig. 6: An example of closeness in Kademlia.................................................................11
Fig. 7: A simple example of Kademlia RPC....................................................................12
Fig. 8: Comparison of module repository sizes (data source: modulecounts.com)...........16
Fig. 9: WebRTC peers interacting with servers in presence of NAT................................18
Fig. 10: A userscript for replacing "customize" with "customise"........................................20
Fig. 11: “Google Translate” extension performing on-demand translation.......................25
Fig. 12: An example of English-Chinese translation in Google Translate as a web application.26
Fig. 13: Google failing to locate a web resource..............................................................28
Fig. 14: A CAPTCHA Google displays upon unusual traffic.........................................29
Fig. 15: “Crane” misinterpreted as a machine on a zoo’s website.....................................30
Fig. 16: Google Translate with “wushu” as input.............................................................34
Fig. 17: The Chinese movie “Dragon Fist” on IMDb.....................................................34
Fig. 18: Google Translate with “Longquan” as input.......................................................35
Fig. 19: Google Translate with “Long quan” as input and “Chinese” as specified source language........................................................................................................................................35

Fig. 20: Google Translate translating “Long quan” on IMDb with source language specified as “English”..................................................................................................................................................36

Fig. 21: Google Translate translating “Long quan” on IMDb with source language specified as “Chinese”..................................................................................................................................................36

Fig. 22: The big picture of adopted technologies..........................................................................................................................................................................................41

Fig. 23: Architecture for the proposed system (single-tab scenario)...............................................................................................................................................42

Fig. 24: Architecture for the proposed system (multi-tab scenario)......................................................................................................................................47

Fig. 25: A possible detrimental feedback loop.................................................................................................................................................................48

Fig. 26: Work flow of Peer as a client of Signaling Server...........................................................................................................................................51

Fig. 27: Work flow of Signaling Server.................................................................................................................................................................53

Fig. 28: Screenshot of CNN.com on Aug. 5, 2016..................................................................................................................................................56

Fig. 29: Footer of CNN.com on Aug. 5, 2016..................................................................................................................................................56

Fig. 30: Substituting downloaded translation for original display text “Game on”...................59

Fig. 31: Screenshot of CNN.com on Aug. 6, 2016..................................................................................................................................................59

Fig. 32: Screenshot of CNN.com “Game on” article on Aug. 5, 2016.................................60

Fig. 33: Substituting downloaded translation for original title of “Game on” article..............60

Fig. 34: Gist of delayedSync.........................................................................................................................63

Fig. 35: Pseudocode of delayedSync........................................................................................................63

Fig. 36: Pseudocode of sync used in delayedSync..................................................................................64

Fig. 37: An example of value in Local Storage......................................................................................66
Fig. 38: A sample response for lookup (download) of key “en-US CNN”.................................67
Fig. 39: A sample request for insertion (upload) of value “Cable News Network”...............67
Fig. 40: An example illustrating internal working process of the proposed system.............67
Fig. 41: Screenshot of integrated Translation GUI on a web page....................................82
Fig. 42: Key source of a hyperlink for“Link+Text” matching level..................................84
Fig. 43: Key source of a hyperlink for“Link Only” matching level..................................84
Fig. 44: Key source of text for “Text Only” matching level.............................................84
Fig. 45: Screenshot after “Save” button is pressed in Fig. 41............................................86
Fig. 46: Screenshot after “Filter On” checkbox is ticked in Fig. 45.....................................88
Fig. 47: Screenshot after “Reset” button is clicked in Fig. 46...........................................88
Fig. 48: Screenshot when clicking “Set Locale” button in Fig. 45......................................90
Fig. 49: Screenshot after “Set Locale” is confirmed in Fig. 48...........................................90
Fig. 50: Screenshot when “Reset All” button is clicked in Fig. 49......................................91
Fig. 51: A Sample exported JSON file.............................................................................92
Fig. 52: Scalability test setup............................................................................................97
Fig. 53: Experimental results of scalability test...............................................................98
Chapter 1

Introduction

Although the World Wide Web, or simply the web, was invented by a native English speaker, it has now become a commonplace throughout the globe regardless of language or cultural background. There is no denying that English has hitherto remained a web lingua franca, yet recent years have witnessed a decline in its predominance over the Internet [1]. As the web has started to flourish in countries where English is not the first language, the growth of non-English websites is gaining momentum. Furthermore, the percentage of native non-English speakers on the Internet has soared to 74.1% [2]. The resultant linguistic diversity has stimulated demand for web translation, bringing into being online translation services backed by natural language processing [3] and machine translation [4]. Web users can, via merely a few mouse clicks, take advantage of an online translation service whenever they encounter a web page in a language other than their own. Upon request, the service in question is able to translate a portion of or the entire web page in place within seconds. Although the translation quality may not be as high as that of a professional human translator, web users can at least get a general idea of what the web page in question is about.

There is often need for timely and accurate translation of current events on the web. For example, a hurricane striking multiple countries in the same region might concern victims speaking different languages. Those who wish to know the latest information about relief and aid
might resort to news websites for live updates. It is vital that those websites provide information in their respective native languages. However, it is quite challenging and costly for news websites to publish multiple versions (i.e., languages) of latest news reports in a timely fashion.

At present, Google Translate [5] proves by far to be the most successful translation service on the web. First, it is capable of quick and in-place translation. Second, it supports translation between more than 100 languages. Third, it is free of charge. Fourth, it is a versatile tool that can be integrated into a website by a webmaster, incorporated into a web browser by a software developer, or accessed interactively on Google Translate website by a web user. Fig. 1 shows Google Translate website applying in-place English-Chinese translation to the main menu of an English website.

![Google Translate](http://www.cnn.com/)

*Fig. 1: Google Translate applying in-place English-Chinese translation to the main menu of an English website*

Although Google Translate is virtually dominating the web translation market, alternative services do exist. Among them are Microsoft Bing Translator [6] and Dictionary.com [7]. Like
Google Translate, Microsoft Bing Translator also provides, in addition to interactive access to its website, a convenient translation widget [8] that can be plugged into any web page by a webmaster. Unlike Google Translate and Microsoft Bing Translator, Dictionary.com focuses on translation of short paragraphs as it limits length of source text to 300 characters on its interactive interface.

Today’s web browser is so versatile that it forms an integral part of daily life. Thanks to a plethora of web applications, web users can now read news, check emails, manage bank accounts, book air tickets, get driving directions, watch video clips, chat with friends and deal with other routines, all within the same browser window. The success of web applications can be attributed to cross-platform compatibility, server-centric software maintenance and ubiquity of wireless Internet, among others. State-of-the-art web technologies have even made it possible to achieve inter-browser communication, which lays the foundation for any prospective browser-based peer-to-peer network.
Chapter 2

Background

In this chapter, we review all technologies and concepts pertinent to this work, including Distributed Hash Table, peer-to-peer data sharing network, web application, augmented browsing and crowdsourcing. Emphasis is placed on three enabling technologies, namely, Kademlia, Node.js and WebRTC.

2.1 Dictionary and Hash Table

In old days, bilingual dictionaries were widely used as a tool in language translation. In essence, a bilingual dictionary maps words in one language to those in another, and hence can be regarded as a data structure containing multitudes of key-value pairs. Fig. 2 shows three entries of an English-Chinese dictionary as key-value pairs.

<table>
<thead>
<tr>
<th>Key (English)</th>
<th>Value (Part of Speech and Chinese)</th>
</tr>
</thead>
<tbody>
<tr>
<td>abandon</td>
<td>verb: 抛弃</td>
</tr>
<tr>
<td>abide</td>
<td>verb: 遵守</td>
</tr>
<tr>
<td>ability</td>
<td>noun: 能力</td>
</tr>
</tbody>
</table>

Fig. 2: Three entries of an English-Chinese dictionary as key-value pairs

Since key-value pairs in Fig. 2 are listed in lexical order, it is not difficult to retrieve the Chinese explanation of an English word utilizing binary search algorithm, which has $O(\log n)$
time complexity in terms of search, insert and delete operations for both average and worst-case performance. As responsiveness is a pivotal factor in web translation, however, O(log n) time complexity is still apt to degraded user experience. Fortunately, performance of a dictionary can be further boosted by hash table [9], a data structure having O(1) time complexity in terms of search, insert and delete operations for the average case. The worst-case time complexity of hash table is O(n), which suggests collisions always occur. A collision is an undesirable incident where at least two different keys hash to the same value. Fig. 3 illustrates a hash table storing entries in Fig. 2. The hash function maps variable-length keys to fixed-length (i.e., 4-bit) addresses, which hold values corresponding to those keys, respectively.

![Hash Table Diagram](image)

*Fig. 3: A hash table storing entries in Fig. 2*

Although collisions can be avoided most of the time given a carefully designed hash function, they must be resolved once present. Common techniques of collision resolution include chaining and open addressing [9]. Nevertheless, those techniques would fail once the hash table becomes full. For example, once the number of keys in Fig. 3 exceeds 16, i.e., the maximum number of values a 4-bit address can represent, a collision is bound to occur. In that case, we
have to resize the the hash table by using longer (e.g., 8-bit) addresses. As collisions are rare in practice, we can still attain amortized O(1) time complexity.

Hash tables can work very well in a standalone computer. Yet pervasive computer networks have posed a problem for once centralized hash tables, that is, how to achieve high performance given that all hash tables reside on one centralized computer confronting zillions of prospective clients. One plausible tactic is to invest in a powerful yet expensive server. But that is no more than delaying the long-term scalability predicament, as there is always a limit to what a single server can handle concurrently, no matter how expensive that server could be. Neither does it solve potential problems such as single point of failure. One may point out that current commercial solutions often feature shared or even dedicated computer clusters that come with load-balancing, failover and other sophisticated performance-enhancing capabilities. None the less, a cluster of commercial servers involving costly bandwidth and maintenance service is seldom a preferable option for low-budget and non-profit projects where funding is limited, if any.

2.2 Peer-to-peer Data Sharing Network

Popular peer-to-peer applications like eMule [10] and BitTorrent [11] used to rely exclusively on centralized servers for peer discovery [12][13]. Notwithstanding the low traffic volume between servers and peers in peer-finding process, the overdependence over centralized servers rendered traditional peer-to-peer data sharing susceptible to single point of failure. Moreover, those servers were sometimes haunted by legal issues related to copyright
infringement. Complaints from copyrights holders have even caused some servers to be blocked in countries such as the Netherlands [14].

2.2.1 Serverless Peer-to-peer Technology: Distributed Hash Table (DHT)

With a view to reinforcing system robustness and avoiding copyright lawsuits, a growing number of peer-to-peer applications, including eMule and BitTorrent, have adopted serverless approaches to peer discovery, Distributed Hash Table in particular. A Distributed Hash Table (DHT) [15] is a distributed system that provides key-value storage and lookup service [16] in absence of central coordination. In DHT, load is distributed among all interconnected computers (or other devices such as smartphones).

Since 2001, numerous DHT implementations have been proposed [17][16][18][19][20][21] [22]. Yet only a few, most notably Kademlia and its variant Mainline DHT, have gained popularity in practical peer-to-peer data sharing applications.

2.2.2 Overlay Network and DHT

An overlay network is a virtual network built atop physical network. A node, which represents a device (e.g., desktop, laptop, smartphone, etc.) in the overlay network, reaches another via a logical link. The logical link in question may correspond to a complex path (e.g., traversing multiple routers and switches) in the physical network. In the context of routing, a node is also referred to as a contact. Fig. 4 shows an example of overlay network. It can be seen
that the three nodes in the overlay network can reach one another via more than one physical path.

![Overlay Network Diagram](image)

*Fig. 4: An example of overlay network*

Fundamentally, DHT runs over an overlay network. Each DHT node is assigned a unique node ID, which is the output of a hash function. Node IDs are also assigned to keys, as is the case with conventional centralized hash table. A node joins DHT by bootstrapping, the initialization process of admitting a new node to DHT by presenting it with one contact that is already in DHT. For brevity’s sake, we call the contact presented seed. A seed can be either static (e.g., a predefined node that is never offline) or dynamic (e.g., those that have joined DHT recently). Seed is used as an entry point to DHT. Fig. 5 shows a DHT storing entries in Fig. 2, using CRC-32 [23] as hash function. It is worth noting that the value of a key is not stored onto a node having exactly the same node ID as the key. Instead, it is stored onto one node (or more
than one node when needed) whose node ID is the closest to that of the key. As a result, multiple keys may correspond to a same node. The closeness, or distance between two node IDs, can be defined as per design requirements. The simplest definition of closeness is probably the difference between two node IDs. For example, the key “abandon” hashes to “8D8BB078”. Consequently, the Chinese value corresponding to “abandon” should be stored on the node with node ID “8D8BB077”, which is the closest of the three in DHT (with difference being 1). When we need to increase redundancy (and hence robustness) of DHT by requiring a key to be stored on two nodes, we need to additionally store the Chinese value corresponding to “abandon” onto the node with node ID “88105DB5”, as “88105DB5” is closer (i.e., with difference being 57B52C3) than “35CFEE3A” (i.e., with difference being 57BBC23E) to “8D8BB078”. The concept of closeness lays the foundation for routing in the overlay network.

![Diagram](image)

**Fig. 5: A Distributed Hash Table storing entries in Fig. 2 using CRC-32 as hash function**
In our discussion, “DHT” is the only application running over “overlay network” and hence those two terms can be used interchangeably. We emphasize storage aspect of the system when we use “DHT” and networking aspect of the system when we use “overlay network”.

2.2.3 DHT Case Study: Kademlia

Kademlia [20] is a type of DHT prevalent in peer-to-peer applications. It came into being in 2002 and has since attracted more than 3 million users [24] around the world. Nowadays, it is adopted by popular peer-to-peer software like eMule and Vuze [25]. In essence, Kademlia is a structured overlay network [26] built atop the User Datagram Protocol (UDP) [27].

Kademlia has a unified ID space for both nodes and files. Each node and file is represented by an unsigned 160-bit binary ID, respectively. In a naive implementation, a node ID can be obtained by generating a 160-bit hash, e.g., from the node’s IP address. A file ID is its SHA-1 hash [28], which is also a 160-bit number. Thus, both node and file can be located by looking up, in an iterative style, distance between two IDs, which can be obtained with an exclusive-or (XOR) operation

\[ \text{Distance}(A, B) = A \oplus B. \]  

(1)

The less the unsigned value that the exclusive-or operation yields, the closer a pair of nodes or files are to each other. Fig. 6 shows an example illustrating closeness in Kademlia.
Kademlia uses four types of Remote Procedure Call (RPC), namely, PING, STORE, FIND_NODE and FIND_VALUE. A simple example of how nodes in Kademlia use these RPCs to locate each other and store/retrieve resources is shown in the Fig. 7.

Suppose we generate Node IDs from the following IP addresses: A = 10.1.1.11; B = 10.1.1.12; C = 10.1.1.13

*NodeA* = SHA1(A) = 379473F1862068A70FCC643AD7606E758F7865AB

*NodeB* = SHA1(B) = 341BBCE129F21F7DF84A2A1783D232661D4AD05E

*NodeC* = SHA1(C) = 0FF99E6C3CAED5A71C8A8A38D1A11B6FCA73B358

*NodeA XOR NodeB* = 038FCF10AFD277DAF7864E2D54B25C139232B5F5

*NodeC XOR NodeB* = 3BE2228D155CCADAE4C0A02F52732909D7396306

*NodeA XOR NodeB* < *NodeC XOR NodeB*

Hence, A is closer to B than C.

*Fig. 6: An example of closeness in Kademlia*

Most merits of Kademlia stem from inherent properties of its XOR metric. Other contributing factors include agile routing and trust in long-standing nodes. The following list briefly summarizes advantages that Kademlia enjoys.

- XOR metric is simple and fast to compute.

- XOR metric is symmetric and hence load-balanced.

- XOR operation along with all 160-bit binary numbers form an *abelian group*, thus allowing closed analysis.

- Parallel and asynchronous binary-search routing yields high performance.

- Preference for long-standing node in the routing table maximizes the probability that those nodes will remain online in the next hour [20].
Kademlia has a few variants, most notably Mainline DHT, which is the de facto standard for BitTorrent DHT and is being used by 15 to 27 million people on a day-to-day basis [29]. BitTorrent clients used to rely on a central server called "tracker" to discover peers and exchange files. After the introduction of DHT, however, this is no longer the case. Varvello and Steiner [30] monitored the activity of European ISP subscribers, and found that approximately 40% of the BitTorrent users already abandoned trackers in favor of the Mainline DHT and Azureus (a less popular BitTorrent DHT).
Mainline DHT differs from Kademlia in the following aspects.

- Simplified routing table
- GET_PEERS in place of FIND_VALUE (as in Kademlia)
- ANNOUNCE_PEER in place of STORE (as in Kademlia)

Since there is no radical change in Mainline DHT as compared to Kademlia, the popularity of Mainline DHT can be deemed a success of Kademlia as well.

2.2.4 Possibility of Incorporating Kademlia into the Web

KadOH (Kademlia over HTTP) [31] is an early attempt to incorporate Kademlia into the web. It was designed to work on both server-side Node.js and client-side web browser. Its browser support was built upon XMPP over BOSH [32]. XMPP (Extensible Messaging and Presence Protocol) [33] is a communications protocol designed to support instant messaging by relaying messages between different servers that only service their own clients, respectively. BOSH (Bidirectional-streams Over Synchronous HTTP) [34] simulates a persistent session by sending successive HTTP requests from a web browser to a server.

The browser support proposed in KadOH has several drawbacks. First, XMPP is inconvenient as a user is often required to create an account on a website and to log in when connecting to a server. Second, XMPP involves establishing numerous XMPP-compliant servers, which counteracts the merits of a Distributed Hash Table such as Kademlia. Third, XMPP employs XML [35][36] to wrap up data, which is not as compact as JSON [37] and hence wastes
network traffic. Fourth, BOSH yields, as a result of lengthy HTTP headers, higher latency than more recent technologies such as WebSocket [38].

2.3 Web Application

Web application has prospered as both users and developers are turning to cloud computing for easier software maintenance. Node.js and WebRTC are among the contributing factors of this process.

2.3.1 Cloud Computing

Cloud computing [39] relieves application users of the burden of maintaining application locally. They do not need to install or update the software any more. Nor do they need to store data locally, as everything they generate can be managed via cloud. Most cloud-based applications are also cross-platform, accessible from various operating systems and devices. Server-wise, application developers can focus on implementing application logic instead of maintaining multiple versions of the same function for disparate operating systems. It is the merits of cloud computing that have motivated developers to come up with various web applications.

2.3.2 Conventional Web Applications

JavaScript has been employed by many popular web applications [40][41] as the client-side programming language as it utilizes local computing capability and hence eliminates the
need to wait for a server response. None the less, very few web applications adopted JavaScript for server-side coding until Node.js burgeoned in 2009 [42].

2.3.3 Thriving Node.js Server-side Web Applications

Node.js is a runtime environment for server-side web development. The most striking characteristic of Node.js is its event-driven architecture, which significantly boosts throughput and scalability. Moreover, Node.js practically unifies server-side and client-side coding, as Node.js is based on JavaScript, a programming language that client-side web developers have been using on a day-to-day basis. The resultant ease of learning dramatically lowers the barrier to entry for an overwhelming percentage of web developers worldwide. Furthermore, Node.js is also distinguished for its tremendous repository of modules, namely, npm [43]. In July 2014, npm became the largest module repository of its kind [44]. As is shown in Fig. 8, the number of modules in npm reaches 280,000 (as of May 30, 2016 [45]), leaving other major module repositories far behind. The rapid growth of Node.js has spurred an upsurge of interest in web industry, intriguing a good many noted corporations including Groupon, IBM, LinkedIn, Microsoft, Netflix, PayPal, SAP, Walmart and Yahoo! [46]. Server-side as Node.js was originally designed for, it has also found application in distributed networks such as WebTorrent [47], a peer-to-peer data sharing project.

In 2011, a convenient tool called Browserify [48] came into being, making a Node.js module convertible to client-side JavaScript, as long as the module does not use any feature that is forbidden in a web browser (e.g., raw TCP [49] and UDP [50] connections, which are not
permissible in a web browser due to security concerns). In other words, multitudes of Node.js modules could be easily converted to code executable by a web browser without the need for any single server. Since then, developers have been able to develop client-side applications utilizing tens of thousands of existing Node.js modules in npm.

![Comparison of module repository sizes](data-source: modulecounts.com)

**Fig. 8: Comparison of module repository sizes**

### 2.3.4 WebRTC

WebRTC [51] is designed for direct or server-assisted communication between browsers. It is specially tailored for real-time communications, such as voice calling or video chat, yet is also capable of handling exchange of conventional data, such as peer-to-peer file sharing. WebRTC is
currently natively supported by major PC browsers like Google Chrome, Firefox and Opera, as well as Android, the most popular [52][53] mobile platform [54].

In the simplest scenario, where all communicating peers are on the same Local Area Network (LAN), no server is required. A signaling server may be needed to facilitate exchange of metadata arising from session and media management but is not mandatory.

In a realistic Internet scenario, however, at least one server is required to cope with firewalls that implement Network Address Translation (NAT) [55]. This is where Interactive Connectivity Establishment (ICE) [56] comes into play, using servers such as Session Traversal Utilities for NAT (STUN) [57] and Traversal Using Relays around NAT (TURN) [58] to cater to the need of NAT Traversal [59]. A STUN server tries to get an external IP address and a port for an internal peer that resides behind NAT, thus setting up a connection through which the peer communicates with an external peer. In case the STUN server fails to accomplish the aforementioned task, ICE allows the internal peer to connect to a TURN server, which is able to relay all traffic between the internal peer and any external peer. As all traffic is routed via the TURN server in this case, the communication between those two peers is no longer peer-to-peer. Fig. 9 shows two WebRTC peers interacting with a signaling server, two STUN servers and two TURN servers to get around NAT.

2.4 Augmented Browsing

Nowadays, numerous augmented browsing technologies such as bookmarklet, userscript and browser extension are available to help web users customize the content in web browsers.
2.4.1 Personalized Web Experience

As web surfing has become an everyday occurrence, some power users are no longer satisfied with being fed by whatever websites offer to them. They need more control over what content is displayed and how it is displayed. They want the web to cater to their specific need. For example, web users may wish to block advertisement that does not interest them at all. Fortunately, technologies [60][61] have emerged that empower users to, from the client side, alter their favorite websites on the fly. Most of those technologies are driven by JavaScript [62], the programming language of the web [63].

![Diagram of WebRTC peers interacting with servers in presence of NAT]

Fig. 9: WebRTC peers interacting with servers in presence of NAT

2.4.2 Bookmarklet

A bookmarklet [64] is a lightweight JavaScript program that can be placed on bookmark bar of a web browser and triggered manually by a mouse click, the same way of opening a regular bookmark. As a bookmarklet has to be triggered manually, it cannot invoke itself in the background each time a web page is opened, as a daemon program normally would.
2.4.3 Userscript

A userscript [65], or user script, is a client-side computer program written in a scripting language such as JavaScript. It is able to modify a web page on the fly, when executed in a web browser. Any power user having a good command of JavaScript can write a userscript to modify or add a function, or customize the appearance of a web page. For example, if a British user prefers “customise” to “customize” and wishes to replace every occurrence of “customize” on a web page with “customise”, he or she can write a userscript that automates that task, as shown in Fig. 10. Better yet, that British user can share the userscript on a userscript hosting site such as GitHub Gist [66] so that other British users can install that userscript to personalize their web experience in the same way. Most browsers do not have built-in support for userscript. (Google Chrome provides partial built-in support [67] for userscript, though.) Consequently, web users usually have to install and manage userscripts via a userscript manager such as Greasemonkey [60] and Tampermonkey [68]. None the less, the use of a userscript manager brings about an array of benefits. First, a userscript can be executed on any browser without modification, as the responsibility of resolving any potential compatibility issue is left to the underlying userscript manager. Put another way, userscripts become totally browser-independent. Second, the scope in which a userscript takes effect can be customized. For instance, a userscript can be disabled on certain websites but permitted to run on others. Third, as the original author of a userscript publishes updated versions, the userscript can be updated automatically via the userscript manager without user intervention.
// ==UserScript==
// @name "customize" replaced with "customise"
// @include *
// ==/UserScript==

```
document.body.innerHTML = document.body.innerHTML.replace(/customize/g, 'customise');
```

Fig. 10: A userscript for replacing "customize" with "customise"

### 2.4.4 Browser Extension

A browser extension [69] is a client-side computer program that augments browser functionality. Unlike a userscript, which can also be executed in a browser, a browser extension is more technically involved as it may contain a bundle of files written in JavaScript, HTML (i.e., HyperText Markup Language) [70][71] and CSS (i.e., Cascading Style Sheets) [72]. Besides, it has privileged access to browser-specific API (i.e., Application Programming Interface). Not surprisingly, a browser extension is more powerful than a userscript in that it is more tightly coupled with internals of a browser. The tight coupling can, however, lead to an awkward situation where a browser extension depends so much on specifics of one browser that it is rather taxing, if not impossible, to port it to another browser. For the aforementioned text replacement scenario, both Google Chrome [73] and Firefox [74] have browser extensions suited for the task, but an extension developed for one browser cannot be directly installed on the other.

### 2.5 Crowdsourcing

Crowdsourcing may find valuable application in language translation, particularly in the aspect of improving translation accuracy.
2.5.1 Definition of Crowdsourcing

The term “crowdsourcing” was coined in 2005 by Jeff Howe [75][76] when he was discussing with his editor about how the Internet was helping businesses use amateurs to replace professionals. Howe brought his portmanteau word to public attention in 2006 with an article [77], in which he described crowdsourcing as a new pool of cheap labor where everyday people using their spare cycles to create content, solve problems, and even do corporate R&D. Merriam-Webster defines “crowdsourcing” as the practice of obtaining needed services, ideas, or content by soliciting contributions from a large group of people and especially from the online community rather than from traditional employees or suppliers [78], emphasizing the large number of participants and the inclination of online task dispatching.

2.5.2 Evolution of Crowdsourcing Systems

The first notable instance of successful crowdsourcing might be the legendary Goldcorp Challenge [79] launched in March 2000, five years before the term “crowdsourcing” was coined. Participants around the world were encouraged to examine the geologic data from Goldcorp’s Red Lake Mine and submit proposals identifying potential targets where the next 6 million ounces of gold will be found. By offering more than US $500,000 in prize money to 25 top finalists who identified the most gold deposits, Goldcorp attracted more than 475,000 hits to the Challenge’s website and more than 1,400 online prospectors from 51 countries. The numerous
solutions from the crowd confirmed many of Goldcorp’s suspected deposits and identified several new ones, 110 deposits in all.

Since 2000, websites specializing in crowdsourcing have sprung up in profusion [79]. Some of them are for-profit marketplaces (e.g., Amazon Mechanical Turk [80]) whereas others are nonprofit information-sharing platforms (e.g., Wikipedia [81]). The websites mentioned above can be categorized as explicit [82] crowdsourcing, i.e., participants are well aware that they get involved in a multi-user task. Yet there exist implicit [82] crowdsourcing systems, in which an individual is unknowingly enlisted to solve a problem behind the scenes. A case in point is reCAPTCHA [83], a free service that protects website from spam and abuse by using advanced risk analysis techniques to tell humans and bots [84] apart. As a human is solving CAPTCHA [85], the user’s interaction with reCAPTCHA is recorded in the background with a view to digitizing text, annotating images, and building machine learning datasets. This in turn helps preserve books, improve maps, and solve hard artificial intelligence problems.

2.5.3 Crowdsourcing in Language Translation

As language translation is an activity that entails intensive human labor, it is not surprising that crowdsourcing has found valuable application in translation between natural languages. Ambati et al. [86] proposed a paradigm that combines active learning with crowdsourcing to enable automatic translation for low-resource language pairs. Their experiments with crowdsourcing on Amazon Mechanical Turk have shown that it is possible to create parallel corpora using non-experts, and with sufficient quality assurance, a translation system trained
using these corpora could achieve translation quality comparable to that of experts. Crowdsourced translation has also been integrated into several language-related websites such as Duolingo [87]. Facebook has also tried crowdsourcing website translation to its end users [88].
Chapter 3

Problem Statement

This chapter addresses the realistic problem we plan to resolve. Shortcomings of the most popular web translation system (i.e., Google Translate) are analyzed. Additionally, the significance of tackling those shortcomings and expected contributions of this research are given at the end of this chapter.

3.1 Web Translation

Internet has brought together people from all over the world. It is not unusual for a web user to encounter a web page written in a language other than his or her native language. This everyday occurrence has given rise to web translation technologies. Conventional web translation is, for the most part, based on client-server model and machine translation. A web user can install a browser extension from a translation service provider and trigger translation on any web page by clicking a “Translate” button or menu item that is embedded into a browser\(^1\). Fig. 11 shows a snapshot of the Google Chrome extension “Google Translate” in action [89].

\(^1\) Google Chrome has built-in support for Google Translate and hence does not require any extension to be installed.
3.2 Case Study: Google Translate

Google Translate is currently the most popular web translation service [90][91]. It is ubiquitous, integrated into various Google products such as Google Chrome. It also provides a widget called “Website Translator” that can be easily plugged into any web page [92]. But it can also work on any web browser as a web application, which does not require installation of any plugin. For advanced users familiar with URL [93] syntax, translation of any web page can be readily obtained by directly opening a well-formed hyperlink. The core URL syntax of Google Translate web application is as follows:\(^{2}\).

---

\(^{2}\) This is not intended to be a complete guide to Google Translate parameters. Only the core part of syntax is mentioned.
Fig. 12 shows an example of English-Chinese translation in Google Translate as a web application, which can be obtained by visiting the hyperlink below.


Fig. 12: An example of English-Chinese translation in Google Translate as a web application

Google Translate boasts quite a few merits. First, it is capable of quick response to on-demand translation, outputting generally readable and intelligible translated text within seconds. For occasional use, the response time is totally acceptable and the translation quality is usually sufficient for a rudimentary grasp of general ideas. Second, the on-demand service that Google Translate provides is quite cost-effective as it involves no human labor. Google Translate is based on Statistical Machine Translation [94] and hence requires no human intervention during translation. Third, Google Translate can be integrated into any extensible web browser thanks to augmented browsing technologies. Basic functions of Google Translate such as translating the
whole web page have been built into Google Chrome by default since 2010 [95], and advanced functions such as translating selected text are available after a user installs its browser extension [89]. For all its merits, Google Translate has quite a few shortcomings inherent in client-server model and machine translation.

3.2.1 Problems with Client-server Model and Centralized Data

First, data privacy and security may loom large in a client-server model. In the case of Google Translate, user data need to be sent to Google’s centralized servers. How those data are handled is at the discretion of Google, as pointed out in Google’s Terms of Service [96]:

“...When you upload, submit, store, send or receive content to or through our Services, you give Google (and those we work with) a worldwide license to use, host, store, reproduce, modify, create derivative works (such as those resulting from translations, adaptations or other changes we make so that your content works better with our Services), communicate, publish, publicly perform, publicly display and distribute such content.”

That said, data privacy is not guaranteed in all Google’s services, including Google Translate. Free lunch as Google Translate appears at first glance, users pay a hidden cost to Google when using its service. In addition, centralized data in a client-server model poses a risk to data security. Although Google has so far kept an almost impeccable record with respect to data security, millions of users could be affected if a data breach would happen to any one of Google’s datacenters.
Second, Google Translate may be discontinued at any time in Google’s sole discretion. Despite the numerous free services Google offers online, Google is a for-profit corporation after all. It would not hesitate to shut down any service that is unlikely to generate revenue, no matter how popular it has become. For example, Google Reader [97] was discontinued in 2013 amid objection from over 150,000 users [98], for the mere sake of declined usage [99]. There is no guarantee that Google will not do the same to Google Translate.

Third, Google Translate may be inaccessible at times. In spite of the multitude of servers Google possesses, Google, as a service provider, is not infallible. For example, Google cached hyperlinks [100] can occasionally return invalid results, as illustrated by the snapshot in Fig. 13. Likewise, it is entirely possible that Google Translate becomes temporarily inaccessible.

![Google 404 Error](image)

**Fig. 13: Google failing to locate a web resource**

Fourth, even if Google Translate is up and running, it may deny service as a precaution against Distributed Denial-of-Service (DDoS) attack [101]. When Google detects a burst of high traffic from a frequent, it may display a CAPTCHA, as illustrated in Fig. 14, to verify that the requests it has been receiving are from a human user rather than a robot or a malicious program.
The interruption incurred by CAPTCHA, which is a side-effect of DDoS protection, may upset and annoy heavy Google users who are prone to send multiples requests within a short period of time.

To continue, please type the characters below:

![Image](image)

Submit

About this page

Our systems have detected unusual traffic from your computer network. This page checks to see if it's really you sending the requests, and not a robot. Why did this happen?

*Fig. 14: A CAPTCHA Google displays upon unusual traffic*

### 3.2.2 Translation Inaccuracy

First, the output of Google Translate often deviates from natural language (e.g., chaotic grammar, awkward sentences, etc.) because it does not rely on grammatical rules, which are extensively used in Rule-based Machine Translation (RBMT) [102].

Second, everyday phrases may not be translated in an idiomatic way [103]. For example, English-Chinese translation of “that movie is not my cup of tea” yields “那部电影是不是我的那杯茶” (literally, “that movie is not is my that cup of tea”) in Google Translate (as of this
writing), which is a typical (and grammatically incorrect) word-for-word translation and does not convey idiomatic meaning of the phrase “not one’s cup of tea” (i.e., not attractive to one).

Third, Google Translate does not handle polysemy well and lacks context-sensitivity. Misinterpretation of polysemous words such as “crane” is possible, especially when context is not clear (e.g., an isolated word as a web hyperlink rather than in a textual sentence). Even in the context of a zoo, “crane” can also be misinterpreted as a machine instead of a bird. Fig. 15 shows that Google Translate falsely interprets “crane” as a machine (“起重机” in Chinese) on a zoo’s website.

![Fig. 15: “Crane” misinterpreted as a machine on a zoo’s website](image)

3.2.3 Limited User Collaboration

Although users are allowed to contribute and review translation on Google Translate Community [104], Google is the only entity they can interact with. Put differently, no direct collaboration between users is allowed.
3.2.4 Restrictions on Submitted Translation

Translation submitted to Google Translate is subject to approval in Google’s sole discretion and unavailable for immediate use. Other collaboration websites like Wikipedia, Quora [105] and Stackoverflow [106] are less restrictive. On those websites, most user-generated content is instantly available to the public. Some content has to undergo a simple peer-review but it usually will not take long (e.g., several minutes for a hot topic being watched by many users).

3.2.5 Lack of Personalization

In Google Translate, users are not allowed to personalize translated text. For instance, Google Translate will not use “customise” in place of “customize” during translation even if it is visited from a United Kingdom IP address. As a matter of fact, Google Translate does not provide a way to specify that user preference at all.

3.2.6 Lack of Support for Multilingual Web Pages

Google Translate only allows selection of one source language, which does not accommodate the case where a web page contains text in more than one language (e.g., mixture of Chinese and Japanese).
3.2.7 Lack of Support for Minority Languages and Dialects

At present, Google Translate supports some 100 languages [107]. Given that there are over 5,000 languages spoken worldwide [108], however, Google Translate still lacks support for minority languages, let alone dialects.

3.2.8 Overdependence on English as a Pivot Language

Even if a language is supported, Google Translate may still use English as an intermediate step [109] for translation between two non-English languages, introducing distortion that aggravates inaccuracy in translation.

3.2.9 Poor Support for Ever-evolving Internet Slang

Internet slang [110] diverts from natural language and is ever evolving, sometimes accompanied by emergence of an Internet meme [111]. The dynamic nature of Internet slang makes it difficult for any statistical machine translation technology to catch up. Table 1 lists a few examples of Internet slang that Google Translate fails to interpret correctly (as of this writing).

<table>
<thead>
<tr>
<th>Internet Slang</th>
<th>Plain English</th>
</tr>
</thead>
<tbody>
<tr>
<td>FWIW</td>
<td>For what it's worth</td>
</tr>
<tr>
<td>This.</td>
<td>I agree.</td>
</tr>
<tr>
<td>inorite</td>
<td>I know, right</td>
</tr>
<tr>
<td>b4</td>
<td>before</td>
</tr>
</tbody>
</table>
3.2.10 Lack of Support for Transliterated or Transcribed Text

Many languages, such as Chinese and Arabic, can be written in different scripts. The process of converting text from one script to another is called transliteration, or less commonly, transcription [112][113]. The principal difference between transliteration and transcription is that the former emphasizes spelling correspondence whereas the latter focuses on pronunciation. Another difference is that transcription can also denote representing speech or sign language with a written script. A common form of transliteration and transcription in English-speaking countries is romanization, the process of converting non-Latin script to Latin script. In the translation industry, romanization can be used to record text or speech of non-English languages before idiomatic English translation becomes available. Useful as romanization is, it does not guarantee verbatim precision during conversion. Put another way, romanization can make it impossible to distinguish between homophones, i.e., words having different meanings yet with same pronunciation. The ambiguity accompanying romanization can severely undermine quality of machine translation. Google Translate, in this regard, is no exception. For example, romanized pinyin [114] “wushu” correspond to a few Chinese words, such as “无数” (English: countless), “武术” (English: martial arts) and “巫术” (English: witchcraft). It goes without saying that no machine translation technique can interpret “wushu” accurately unless its context is clearly given. When we input “wushu” into Google Translate, it correctly detects input as Chinese but returns only one interpretation, that is, “无数”. (See Fig. 16.) Things get even more intricate
when it comes to proper nouns. Take the Chinese movie “Dragon Fist” on IMDb [115] as an example. As can be seen in Fig. 17, the main title of that movie is “Long quan”, which is the pinyin romanization of two Chinese characters, i.e., “龙拳”.

![Google Translate with “wushu” as input](image)

*Fig. 16: Google Translate with “wushu” as input*

![IMDb](image)

*Fig. 17: The Chinese movie “Dragon Fist” on IMDb*

Now we input “Long quan” into Google Translate, as shown in Fig. 18. Upon input, Google Translate automatically detects “Long quan” as Vietnamese, which is not we are looking for. In the meantime, the output “Long view” has nothing to do with “Dragon Fist”. After we
explicitly specify “Chinese” as source language, “long full” shows up as output, as is shown in Fig. 19. But that output makes even less sense, as “long full” is not even a valid English phrase. Nor is it relevant to “dragon” or “fist”. Apparently, Google Translate mistakes “Long” in the input for an English word, while it is actually intended to represent the Chinese character “龙”.

![Google Translate with “Long quan” as input](image1)

*Fig. 18: Google Translate with “Long quan” as input*

![Google Translate with “Long quan” as input and “Chinese” as specified source language](image2)

*Fig. 19: Google Translate with “Long quan” as input and “Chinese” as specified source language*

One may argue that it is the lack of context that has led to Google Translate’s failure to interpret “Long quan” correctly. However, Google Translate also fails even when the entire web page is given as context. Fig. 20 and Fig. 21 show translation results of Google Translate when the entire page is translated into English, with the source language and target language specified
as English and Chinese, respectively. The former outputs “Long concerned” whereas in the latter “Quan Long”, neither of which goes anywhere near the correct interpretation. It is worth noting that the correct interpretation “Dragon Fist” is already given in movie description under the main title but unfortunately Google Translate does not make use of that piece of information.

Fig. 20: Google Translate translating “Long quan” on IMDb with source language specified as “English”

Fig. 21: Google Translate translating “Long quan” on IMDb with source language specified as “Chinese”

36
3.2.11 Wasted Web Traffic Owing to Identical Queries

In Google Translate, fetched translation is not stored locally, which will incur extra traffic in the event of duplicate requests for the same piece of text.

3.3 Significance of Research and Expected Contributions

The proposed system is aimed at overcoming various drawbacks plaguing conventional web translation by exploiting creativity of online communities and pooling wisdom of the crowd. It also places emphasis on collaborative translation of a monolingual website in its entirety. Furthermore, the proposed system has a long-term goal of bridging cultural gaps in the realm of Internet, bringing together netizens with similar interests around the world. The proposed system is not meant to displace the existing machine translation framework, but to serve as a potent supplement.

Once completed, the proposed system should yield a robust, responsive and scalable network of cross-cultural knowledge, independent of any business organization. The resultant network is also supposed to enrich inter-browser communication technologies and advance the frontiers of web-oriented distributed data store.
Chapter 4

Proposed Solution: DHT-based Collaborative Web Translation

This chapter begins with fundamentals of web development, which serve as prerequisites for designing a collaborative web translation system. Then we continue to explicate the methodology adopted in the proposed solution. Finally, we list the challenges we aim to tackle, along with expected results we can obtain after tackling them.

4.1 Prerequisites

Prerequisites for designing a collaborative web translation system include web page origin, localStorage property, browser tab and locale.

4.1.1 Origin of a Web Page

The origin [116] of a web page is the portion of its URL that precedes the first single slash “/” (i.e., excluding consecutive slashes such as “//”), or the entire URL if no single slash exists. For instance, origin of the URL “http://www.example.com/path/to/file.html” is “http://www.example.com”. Origin is an crucial concept in web security [117] and in principle, resources belonging to one origin are not permitted to be shared with other origins.
4.1.2 The `localStorage` Property

A web browser allocates, for each origin, an isolated storage object to the `localStorage` [118] property. Web pages of the same origin are allowed to store and share data in `localStorage`. Unlike storage objects like HTTP Cookie [119] and `sessionStorage` [120], data stored in `localStorage` do not expire and will persist even if device hardware powers off. When a tab modifies `localStorage` of an origin, all other tabs of the same origin can be notified of the change.

4.1.3 Web Browser Fundamentals

Modern web browsers are capable of keeping multiple tabs open at the same time. Each tab has an independent URL, which may and may not correspond to web pages from the same origin. For example, a human user may switch between similar products on the same e-commerce website. But that human user may also compare prices of the same product on two e-commerce websites. It should be noted that two or more tabs may have exactly the same URL. Although uncommon, this may be beneficial in some corner cases, e.g., a human user may try to compare different paragraphs of the same article opened in two tabs.

4.1.4 Locale

Generally, a locale [121] is an array of parameters defining a human user’s language, region, and other related preferences that help customize user interface to a state that the human user in question feels comfortable with. The simplest form of locale is a language code (e.g., ISO
639-1:2002 [122]) and a region code (e.g., ISO 3166-1 alpha-2 [123]) joined with a hyphen. For example, the locale “en-US” denotes English (i.e., “en”) spoken in the United States (i.e., “US”).

4.2 Methodology

Key aspects of the adopted methodology are design assumptions, system architecture, interaction between peer and signaling server, user roles in relation to DHT, translation download and upload, data management, conditions for triggering network synchronization, system robustness, local screening, preferred locale, and context-sensitive translation.

4.2.1 Assumptions

- Target audience: web users accessing websites displayed in foreign languages (i.e., other than their respective native languages).
- Translation contributors: amateur and professional translators.

4.2.2 Overview of Adopted Technologies

Fig. 22 provides the big picture of adopted technologies as a 5-layer model. The top three layers, namely, application, Kademlia, WebRTC are involved in the development a browser plugin, whereas the bottom two layers, namely, WebSocket and TCP/IP, are left to browsers and operation systems.
4.2.3 Architecture of the Proposed System (Single-tab Scenario)

For ease of understanding, first consider a simplified case where a browser has only one tab. Fig. 23 shows the architecture of the proposed system when there is only one tab in a browser.

**Human User**

Human User (User for short) interacts with the proposed system through the interface of Web Browser (Browser for short). User may contribute/retrieve translation through Browser.

**Plugin**

Plugin is a container for customized functionality such as a userscript converted from Node.js code via Browserify [48], a browser extension, etc. At present, all popular web browsers support plugins of some kind [69][124][125][126][127][128]. Therefore, a plugin becomes a natural choice for fusing Kademlia and WebRTC into any existing Browser on the web.

Graphical User Interface for Translation (Translation GUI for short) is integrated into any web page opened in Browser. User can modify any web page in a WYSIWYG (i.e., What You
See Is What You Get) manner. Translation GUI can be triggered simply by hovering mouse cursor over an eligible web element, such as a hyperlink. Once that happens, the element in question will be highlighted (e.g., surrounded by dashed line) and a dialog will pop up, allowing User to perform translation-related operations.

WebRTC in Browser is accessed via simple-peer [129], a Node.js module.
**Volatile Memory**

Volatile Memory is the type of computer memory which will lose all stored information when Browser is closed. (Please note that Volatile Memory defined in the proposed system is slightly different from conventional volatile memory in that the latter loses all stored information only when the hardware is powered off. Since we are only concerned with the portion of volatile memory used by Browser, however, the difference can be neglected.) Nowadays, random-access memory (RAM) has become the most prevalent volatile memory in mainstream computer systems. Unless otherwise stated, Volatile Memory in the proposed system is composed of RAM. Volatile Memory is used to accommodate two modules, namely, Translations and Peer.

Translations is a data structure that stores all recently accessed data relevant to the current web page. It serves as a cache for all data loaded (during initialization) from or to be flushed to the module called Local Storage. The Translations module always holds the most up-to-date and hence authoritative information concerning User’s interaction with the proposed system. No module in the proposed system other than Translations is allowed to interact with Local Storage directly. Data cached in Translations need to be flushed to Local Storage from time to time.

In a BitTorrent network, a peer is defined as an entity that implements BitTorrent protocol and listens on a TCP port whereas a node implements DHT protocol and listens on a UDP port. In that context, a peer controls a node. As there is no entity that listens on a TCP port in the proposed system, however, Peer in the proposed system can also be called a node in the overlay network as it behaves exactly like a node.
When Translations needs to request new translation from or send existing translation to Overlay Network, it forwards the request or data to Peer, which is responsible for network communication. Note that communication with Overlay Network is controlled by User. On the one hand, if User is overly concerned with privacy, he or she may decide not to share any translation among Overlay Network and thus connection between Translations and Peer is cut off completely. On the other hand, if User only wishes to keep a small portion of works for private use, he or she may enable sharing by default and disable sharing for specific translation works.

**Non-volatile Memory**

Non-volatile Memory is the type of computer memory which will not lose any stored information when Browser is closed. Hard disk drives (HDD) and solid-state drives (SSD) are two popular types of non-volatile memory on the market. Unless otherwise stated, Non-volatile Memory in the proposed system is composed of HDD and SSD. Non-volatile Memory is used to hold two modules, namely, Local Storage and Global Storage.

Local Storage stores User data, e.g., translations User has chosen. Each origin has its own allocated portion of Local Storage, since the same phrase might be interpreted quite differently on different web sites. This helps resolve issues of polysemy and context-insensitivity in conventional web translation.

Global Storage is an integral part of DHT. It is the portion of storage each Peer shares with other Peers in the Overlay Network module. Global Storage can also used to store routing tables.
As a side note, the concept “Local Storage” put forward here is slightly different from the officially defined “localStorage”, which is equivalent to “Non-volatile Memory” in our design and hence includes “Global Storage” as well. Since “Global Storage” is managed by a third-party library [130] in our implementation and hence not discussed, we will not differentiate between “Local Storage” and “localStorage”.

Servers

In WebRTC, signaling is the process of exchanging media session description, which specifies the transport (and ICE) information as well as media configuration parameters (i.e., media type and format) necessary to establish the media path [131]. Theoretically, WebRTC does not require the Signaling Server module. Two Users can simply exchange all forms of metadata by copying/pasting plain text from/to emails. But that is rather inconvenient as the signaling process is not transparent to User. Signaling Server that automates that process is therefore a rational choice. In the proposed system, Signaling Server can be used to keep track of recently seen Peers so as to facilitate bootstrapping for any new Peer.

STUN Server and TURN Server can also be plugged into the proposed system for the purpose of NAT Traversal.
**Overlay Network**

Overlay Network, where Kademlia functions, comprises many of the aforementioned modules, including Peer in Volatile Memory, WebRTC in Plugin, Global Storage in Volatile Memory, and optional Servers.

### 4.2.4 Architecture of the Proposed System (Multi-tab Scenario)

Now consider a more realistic and complicated setting where Browser has multiple tabs running on multiple websites and hence multiple origins. Suppose we have three tabs open and their URLs are listed in Table 2. The architecture in this scenario is illustrated in Fig. 24. Each tab contains exactly one Peer, which will try to connect to Overlay Network on behalf of the tab containing it.

Now we are faced with how to synchronize two tabs from the same origin. The problem gets a bit tricky because we have to synchronize not only data but also behaviors of those two tabs. For example, if User translates some text in one tab, the translation should be reflected on the other tab if the other tab happens to contain identical text. Fortunately, this is achievable thanks to the notification mechanism provided by *localStorage*. When User performs translation on Tab 1, the change is saved to Local Storage, after which the change is propagated to Tab 2. Tab 2 then looks for identical text on the web page it is on and if there is a match, apply the same translation accordingly. It is worth pointing out that the term “synchronize” here pertains to the effort to coordinate tabs of the same Browser (i.e., tab synchronization) and differs from “translation synchronization”, which is a concept to be brought up in Section 4.2.6.
Table 2: URLs of tabs of Browser in a multi-tab scenario

<table>
<thead>
<tr>
<th>Tab Number</th>
<th>URL</th>
<th>Origin</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td><a href="http://a.com/2/">http://a.com/2/</a></td>
<td><a href="http://a.com">http://a.com</a></td>
</tr>
<tr>
<td>3</td>
<td><a href="http://b.com/1/">http://b.com/1/</a></td>
<td><a href="http://b.com">http://b.com</a></td>
</tr>
</tbody>
</table>

Fig. 24: Architecture for the proposed system (multi-tab scenario)
When multiple Peers from the same origin are connected to Overlay Network at the same time, they may forward, in an asynchronous and hence unpredictable manner, conflicting data to the same Local Storage through their respective Translations modules. Tab synchronization in this case is also error-prone as coordination between those Translations modules gets complicated, if not impossible. To make matters worse, any potential feedback loop may subvert system stability due to effect of positive feedback. A possible detrimental feedback loop arising from Fig. 24 is illustrated in Fig. 25. The adverse effect of feedback loop is likely to become ever more severe when multiple Peers are connected to a same optional server (e.g., Signaling Server).

![Diagram](image)

*Fig. 25: A possible detrimental feedback loop*

The most effective way of preventing such a feedback loop is to remove one of its segments. In our proposed design, we guarantee there is at most one segment between Peers and
Network by enforcing an active/dormant Peer model. For each origin, at most one Peer is allowed to connect to Network at any time. That Peer is called the active Peer of that origin. Other Peers from the same origin, called dormant Peers, requests regularly to become the active Peer of the same origin, by setting a flag in Local Storage. The active Peer is responsible for turning down any such request by clearing that flag, so that dormant Peers get to know there is already an active Peer. After the active Peer is gone (e.g., after the tab containing the active Peer is closed), any newly set flag in Local Storage is no longer cleared, and consequently one of the dormant Peers will become the new active Peer of the origin in question.

4.2.5 Signaling Server in Multi-tab Scenario

Although Signaling Server is not required for WebRTC, it is often needed as it would otherwise be too laborious for Peers to discover each other in Overlay Network. When Signaling Server is present, it manages Peers in tabs of all Browsers, even if those Browsers operate on different devices. Each Peer wishing to join Overlay Network should first connect to it using a unique identifier, which we call nickname. Signaling Server will register the Peer with its nickname, as long as there is no registered Peer that already uses the same nickname. The simplest design of nickname is probably generation of a random string that does not collide with existing ones. In our proposed system, we adopt a similar design that has a 5-byte locale (e.g., “en-US” denoting American English) prefixed to a 40-byte random string.

The locale prefix, which we refer to as preferred locale, is necessary as we wish to cluster users with similar language preference by sending seeds to Peers with the same locale prefix
during bootstrapping. Those seeds have a better chance of providing useful translations to connecting Peers. It should be noted, however, that the contacts a Peer can have should not be restricted to those with similar language preference. Otherwise Users speaking minority languages would not stand a chance of distributing their translation works across Overlay Network, thus exacerbating resource scarcity. The preferred locale is saved in Local Storage so that any Peer of the origin in question can retrieve it when necessary.

The 40-byte random string is the hexadecimal representation of a 160-bit number generated the first time User visits an origin and creates a Peer thereof. The nickname is stored in Local Storage of that origin, and will remain unchanged for a specific origin, until either of the following happens.

- A nickname collision occurs at Signaling Server and the connecting Peer is asked to resolve the collision by changing the nickname corresponding to the origin in question.
- User changes preferred locale of the origin in question.

Now we go through the procedures of Peer and Signaling Server, which are depicted in Fig. 26 and Fig. 27, respectively.

- Work flow of Peer as a client of Signaling Server
  1. When User opens a URL in a tab, a dormant Peer is bound to the origin of that tab and hence is able to access the Local Storage of that origin.
  2. If a nickname exists in Local Storage as a result of a previous visit to the origin in question, the Peer just continues using it; otherwise, a random 160-bit number is
generated, encoded as 40-byte hexadecimal string, and appended to the 5-byte preferred locale retrieved from Local Storage, forming a 45-byte nickname.

(3) The dormant Peer requests to become the active Peer by setting a flag in Local Storage.

(4) If an active Peer exists and clears the flag, the dormant Peer waits for a predefined period of time and requests to become the active Peer again, i.e., Step (3).

(5) If no active Peer exists, the dormant Peer becomes the active Peer.

![Flowchart](image)

*Fig. 26: Work flow of Peer as a client of Signaling Server*

(6) The active Peer (if not connected to Signaling Server yet) connects to Signaling Server with its nickname. The next step the active Peer will take depends on the response it receives from Signaling Server.
(7) If Signaling Server responds with an empty nickname, which suggests no seed is available (e.g., the active Peer is the first to join Overlay Network), the active Peer starts sending seed requests regularly (e.g., at predefined intervals) to Signaling Server (i.e., repeating Step (6)) with a view to establishing connection with future Peers in Overlay Network (i.e., bootstrapping).

(8) If Signaling Server responds with nickname of a seed, either of following will happen.

(a) If nickname of the seed collides with the active Peer, the active Peer must generate a new nickname, notify dormant Peers of the new nickname by saving it into Local Storage, and try connecting again to Signaling Server with the new nickname, i.e., Step (6).

(b) If nickname of the seed does not collide with that of the active Peer, the active Peer will attempt to connect to that seed. If connection attempt fails, the active Peer will, again, regularly send seed requests to Signaling Server, i.e., repeating Step (6). Otherwise, the active Peer completes bootstrapping and joins Overlay Network successfully.

(9) When the number of contacts of the active Peer falls to zero, indicating disconnection from Overlay Network, the active Peer may request seed again, i.e., Step (7).

• Work flow of Signaling Server

(1) Signaling Server listens for incoming connection.
(2) When an active Peer connects with its nickname, Signaling Server checks if the nickname is already registered.

(3) If the nickname is already registered, Signaling Server responds with the nickname in question, signifying to the connecting active Peer that its nickname collides with that of a registered Peer. Signaling Server then returns to Step (2).

(4) If the nickname is not registered, Signaling Server will try to find nickname of a recent Peer (i.e., seed) from a pool of registered Peers having the same preferred locale (i.e., first 5 bytes of nickname) as the connecting active Peer.
(5) If a seed is found, Signaling Server prepares a response containing nickname of the seed. If no seed is found, Signaling Server prepares a response containing an empty nickname.

(6) Signaling Server then registers nickname of the connecting active Peer, adds it to the pool of registered Peers having that same preferred locale, and sends the prepared response to the connecting active Peer. Signaling Server then returns to Step (2).

(7) When a registered active Peer requests seed again, Signaling Server returns to Step (4), except that Signaling Server need not check whether the active Peer is registered.

(8) When an active Peer disconnects from Signaling Server (e.g., when the tab containing that Peer is closed), Signaling Server unregisters it and removes it from the pool containing it. Signaling Server then returns to Step (2).

4.2.6 Translation Download and Upload

When we wish to translate a sentence in a certain foreign language on a web page into our native language, the simplest way might be copying and pasting it into an online dictionary or translator. When we do that, however, we lose the sentence's context, without which the online dictionary or translator is unlikely to produce a relevant and accurate outcome. The same can be said of human produced translation. Indeed, translation context plays a vital part in the translation process. Today's web abounds with hyperlinks, an extraordinary type of web element that can be seen on almost every website. It is so prevalent that analysis of hyperlinks [132] has led to the rise of a predominant Internet giant called Google. A noteworthy property of hyperlinks is that they are unambiguous, carrying the same context information most of the time.
(There are exceptions like permanently moved web pages but they are uncommon in reality as compared to web pages the majority of hyperlinks point to. ) This advantage may be helpful in improving translation accuracy, which is best exemplified by news websites like CNN.com. Take the homepage of CNN.com on Aug 5, 2016 [133] as an example. As can be seen in Fig. 28, the title of breaking news is a hyperlink with display text being "Game on", which is rather vague in meaning. What game is on? A soccer game? A basketball game? A baseball game? There are a myriad of possibilities. None the less, everything becomes clear when we look at the hyperlink revealed at the bottom, i.e., “http://www.cnn.com/2016/08/05/sport/olympics-live-coverage-opening-ceremony/index.html”. Obviously, it is related to the opening ceremony of 2016 Olympics and the title can now be translated into a sentence literally equivalent to “2016 Olympics Opening Ceremony” in English.

Please note it is not the textual information contained in the hyperlink that reveals the specific meaning of “Game on” but the hyperlink per se. For instance, the hyperlink would still remain a qualified identifier if “olympics-live-coverage-opening-ceremony” would be replaced by “article12345”, as long as the hyperlink stays globally unique. An extra benefit of hyperlink-based translation is that translation may work across numerous web pages on the same website owing to the ubiquitous navigation menu. For instance, almost every page on CNN.com has a navigation footer that depicts, with section hyperlinks, skeleton of the entire website. A screenshot of that footer is shown in Fig. 29 [133]. Therefore, it is entirely justified to take hyperlinks into consideration when designing a well-rounded translation download mechanism.
The proposed system adopts a three-level matching scheme for translation download. The first level is matching by both hyperlink and its display text (“Link+Text” for short). In the previous "Game on" example, we join the hyperlink “http://www.cnn.com/2016/08/05/sport/olympics-live-coverage-opening-ceremony/index.html” and its display text “Game on” with a space “ ” to form “http://www.cnn.com/2016/08/05/sport/olympics-live-coverage-opening-ceremony/index.html Game on”, a structure we refer to as Key Source (keySrc). When we wish to translate it, for example, into Chinese used in Hong Kong, we prefix it with “zh-HK”. We refer to the prefix locale used here as Key Locale (keyLcl). The complete key for translation download is thus “zh-HKhttp://www.cnn.com/2016/08/05/sport/olympics-live-coverage-opening-ceremony/index.html Game on”. We send the complete key into Overlay Network and download its value, which is possibly “2016 奧運開幕”. After we receive that value, we substitute it for the original display text so as to finish translation. The substitution effect is shown in Fig. 30. If no match is found in the first level, the second level is triggered, in an effort to match by hyperlink alone (“Link Only” for short). We now illustrate how to use this level, again, with the “Game on” example. On Aug 6, 2016, CNN changed the breaking news title from “Game on” to “Games on”, while leaving the hyperlink unchanged. (See Fig. 31 [134].) Tiny change as it may seem, it renders ineffective our previous translation download in the first level because the display text did not match any more. Given the second level of matching, however, this change no longer posed a problem, because the hyperlink still matched and translation download will always succeed no matter how
CNN changes the display text. When the second level is in effect, the Key Source is changed to the hyperlink itself and the complete key turns into “zh-HKhttp://www.cnn.com/2016/08/05/sport/olympics-live-coverage-opening-ceremony/index.html”. However, if neither of the two levels above works (e.g., the original text is not presented as a hyperlink at all), our proposed system will fall back to the third level, that is, matching by text alone (“Text Only” for short). For example, after we open the “Game on” hyperlink, we lands on an article titled "Rio 2016 Olympics opening ceremony: Follow the action live", as is shown in Fig. 32. The downloaded translation might be “里約2016年奧運開幕：現場直播”, again, in Chinese used in Hong Kong. (See Fig. 33.) The complete key in this case is “zh-HK Rio 2016 Olympics opening ceremony: Follow the action live”. Note that the space between “zh-HK” and “Rio 2016 Olympics opening ceremony: Follow the action live” must not be omitted because it differentiates text from a hyperlink. (Think of a corner case where an article title without a hyperlink starts with “http:/”.)

To sum up, the general format of complete key is “<Key Locale><Key Source>”, where “<Key Source>” is “[Optional Hyperlink] <Display Text>”.
2016 奧運 開幕

Fig. 30: Substituting downloaded translation for original display text “Game on”

Games on

Fig. 31: Screenshot of CNN.com on Aug. 6, 2016
Rio 2016 Olympics opening ceremony: Follow the action live

By Sophie Eastaugh, for CNN

(CNN) — After months of planning, hundreds of headlines and years of training for the world’s

Fig. 32: Screenshot of CNN.com “Game on” article on Aug. 5, 2016

里約2016年奧運開幕：現場直播

By Sophie Eastaugh, for CNN

(CNN) — After months of planning, hundreds of headlines and years of training for the world’s

Fig. 33: Substituting downloaded translation for original title of “Game on” article
Since translation download is triggered whenever User hovers over an eligible web element containing text (e.g., hyperlink), a caching mechanism may be desirable such that redundant downloads concerning the same piece of translation within a short period of time can be eliminated. The same applies to translation upload. For instance, User may modify translation multiple times within seconds, probably due to typos. If an upload were triggered immediately after each modification, a large proportion of network traffic would be wasted, since only the last upload would be valid. The design of a caching algorithm is technically involved as network latency could lead to data inconsistency. (Think of an upload immediately followed by a download. ) Another factor we need to consider when designing upload mechanism is how to preserve earlier contributions made by others before we upload our own. A sensible solution is to precede each upload with a download so as not to erase existing contributions from other Users. Specifically, what we do in the proposed system is to download what others have already translated, merge our own contribution into it, and upload merged translation to Overlay Network. To that end, we have devised a unified synchronization method “delayedSync(key, value, forcesUpload)” for both download from and upload to Overlay Network. When a Peer has nothing new to contribute yet only wishes to see what others have translated, it can just leave out the “value” and “forcesUpload” parameters when calling “delayedSync”. Otherwise, it will merge its own “value” into the translation downloaded by “key”. When the optional “forcesUpload” is set to true, an upload is forced so that all closest nodes in DHT will be set to
“value”. A gist of “delayedSync” is depicted in Fig. 34 and the pseudocode for “delayedSync” is shown in Fig. 35 and Fig. 36.

### 4.2.7 User Roles and Peer Interaction with DHT

When interacting with the proposed system, User can play either of two roles, namely, non-contributor and contributor. A non-contributor accesses websites in foreign languages without contributing any translation whereas a contributor is an amateur or professional translator who contributes translation when accessing websites in foreign languages they are familiar with.

- Peer representing a non-contributor, who could be a monolingual user, grabs published translations for DHT and uses one published translation (possibly at random).
- Peer representing a contributor, who could be a bilingual/multilingual user, grabs published translations from DHT; screens published translations (e.g., apply blacklisting) and either uses one published translation (if at least one of the published is satisfactory) or creates a new translation for later use (if none of the published is satisfactory); then publishes screened translations (possibly along with a new translation) to DHT.

### 4.2.8 Data Format

User data are stored in the form of key-value pairs in both Local Storage and Global Storage.
Fig. 34: Gist of delayedSync

```plaintext
procedure delayedSync(key, value, forcesUpload)
    if key does not exist then
        return
    endif
    if Peer is not connected to Overly Network then
        return
    endif
    if key has been allocated a slot in cache, namely, cache[key] then
        if value exists then
            cache[key] <- value
        endif
    else
        allocate a slot in cache to key, that is, create cache[key]
        if value exists then
            cache[key] <- value
            sync(key, value, forcesUpload)
        else
            sync(key)
        endif
    endif
end procedure
```

Fig. 35: Pseudocode of delayedSync
procedure sync(key, value, forcesUpload)
  needsToCancel <- false
  download translation
  (after a predefined delay, which restricts download/upload frequency)
  if cache[key] is different from value then
    sync(key, cache[key], forcesUpload)
  needsToCancel <- true
  else
    free the slot allocated to key in cache, namely, cache[key]
  endif
return
(after Overlay Network returns downloadedTranslation)
needsToUpload <- forcesUpload
if needsToCancel is true then
  return
endif
if downloadedTranslation contains new content then
  update “Translations”, “Translations GUI”, and “Local Storage”
endif
if downloadedTranslation contains blacklisted content then
  remove blacklisted content from downloadedTranslation
  needsToUpload <- true
endif
if value exists and downloadedTranslation does not contain value then
  merge value into downloadedTranslation
  needsToUpload <- true
endif
if needsToUpload is true then
  upload downloadedTranslation to Overlay Network
endif
end procedure

Fig. 36: Pseudocode of sync used in delayedSync

Key format and value format for Local Storage

The key format for Local Storage is a fixed key prefix followed by Key Source. A key prefix is necessary since Local Storage items can reside in the same namespace as that of Global Storage. (Recall that both Local Storage and Global Storage belong to Non-volatile Memory.) The key prefix chosen should be short, meaningful and distinctive. For example, the key for “CNN” with the key prefix being “keyLS_” is “keyLS_CNN”.

64
The value format for Local Storage is a collection of associative arrays that mingles a variety of data. It follows JavaScript Object Notation (JSON) format. Fig. 37 is an example of value in Local Storage. In that example, the chosen locale (lclC) is English spoken in the United States, or American English (“en-US”) while the chosen target HTML (tgtHtmC) is “Cable News Network”. The all-lowercase candidate “cable news network” is not chosen. Another available locale is Spanish spoken in the United States (“es-US”), which has only one candidate, namely, “Cadena de Noticias por Cable”. There is also a blacklisted item (tgtBhtm) in the locale data for “es-US”, that is, “Cable News Network”. It has been blacklisted as it is not in Spanish and hence not valid. Note that “es-US” also has a tgtHtmC, namely, “Cadena de Noticias por Cable”. As “es-US” is not the current lclC, however, the tgtHtmC of “es-US” does not take effect. It is also worth mentioning that locale data can contain empty fields, as is the case with tgtBhtm of “en-US” in Fig. 37.

**Key format and value format for Global Storage**

The key format for Global Storage is a fixed key prefix followed by the concatenation of Key Locale and Key Source. For example, if the key prefix is “keyGS_” and we need to store American English translations for “CNN”, the key will be “keyGS_en-US CNN”. When we need to query existing American English translations for “CNN” in DHT, however, we must strip the key prefix and then send a lookup (i.e., download) request (equivalent to Step 2 in Fig. 7) into Overlay Network with the key being “en-US CNN”. It is worth pointing out that only the SHA-1
hash of the key is sent, thus concealing the content for which User is seeking translation and hence protecting privacy.

```json
{
  "lclC": "en-US",
  "en-US": {
    "tgtHtm": {
      "Cable News Network": "",
      "cable news network": ""
    },
    "tgtBHtm": {},
    "tgtHtmC": "Cable News Network"
  },
  "es-ES": {
    "tgtHtm": {
      "Cadena de Noticias por Cable": ""
    },
    "tgtBHtm": {
      "Cable News Network": ""
    },
    "tgtHtmC": "Cadena de Noticias por Cable"
  }
}
```

*Fig. 37: An example of value in Local Storage*

The value format for Global Storage is what formulates the response (equivalent to Step 3 in Fig. 7) from Overlay Network. Like the value format for Local Storage, it also follows JSON format. The value in Global Storage practically embraces the latest `tgtHtm` in DHT. The value contained in response may look like Fig. 38. The value in Fig. 38 has only one member, namely, “cable news network”, which is a proper noun that is not capitalized. Suppose we are dissatisfied with this all-lowercase proper noun and decides to replace it with “Cable News Network”. What we do is adding “Cable News Network” to `tgtHtm` and removing “cable news network”. We
subsequently send an insertion (i.e., upload) request (equivalent to Step 1 in Fig. 7) to Overlay Network, with the key being “en-US CNN” and the value being that in Fig 39.

```
{
    "tgtHtm": {
        "cable news network": ""
    }
}
```

*Fig. 38: A sample response for lookup (download) of key “en-US CNN”*

```
{
    "tgtHtm": {
        "Cable News Network": ""
    }
}
```

*Fig. 39: A sample request for insertion (upload) of value “Cable News Network”*

Fig. 40 shows a flow chart illustrating internal working process of the proposed system.

```
Load "Translations" from Local Storage.  

Task: interpret "CNN" in American-English. 

Request "en-US" of "CNN" in "Translations". 

Use cached translation. 

Has it been cached in "Translations"? 

Send SHA1("en-US CNN") = 00681F3B0CF974E809D8340E81E1552A8375327E instead of original content in a request to DHT. 

Flush "Translations" to Local Storage. 

Update "Translations" and modify the web page in place instantly. 

Send SHA1("en-US CNN") along with "Cable News Network" to DHT, storing the value in Global Storage of some node with Node ID closest to that of the key. 

All lowercase acceptable? 

Receive "cable news network" in a response from DHT. 
```

*Fig. 40: An example illustrating internal working process of the proposed system*
4.2.9 Robustness

(1) Ability to correct the erroneous state of more than one active Peer in presence of Signaling Server

The flag-setting/flag-clearing mechanism is supposed to work most of the time, yet there exist two corner cases in which two or more Peers simultaneously become active. The two cases that give rise to that erroneous state are as follows.

(a) At exactly the same moment, two or more Peers detect that a flag in Local Storage is not cleared, signifying that there exists no active Peer. As a result, they simultaneously attempt to clear that flag in order to become the active one. After the flag is cleared, each of them considers itself to be the one that has cleared the flag, which is not necessarily the case. The root cause for this problem is that there is no way in specifications [118] to lock localStorage for exclusive access and hence no way for a Peer to prevent other Peers from entering a critical section (e.g., flag-detecting followed by flag-clearing) in program code.

(b) The tab containing the active Peer hangs for a noticeable period of time (e.g., 10 seconds due to drastic layout [135] change), and consequently fails to clear any flag. After the tab recovers from that halt, there may be a Peer in another tab that turned active during the halt.

The countermeasure we adopt as a remedy for emergence of more than one active Peer is asking an active Peer to surrender, whenever it detects a change of nickname in Local
Storage, its active status by disconnecting from Signaling Server. This works because any additional active Peer is bound to receive a duplicate nickname from Signaling Server, as the first active Peer has already registered by the time any additional active Peer connects to Signaling Server. By design, the additional active Peer has to generate a new nickname and notify other Peers from the same origin of that new nickname. Therefore, the first active Peer can always detect the nickname change, and hence existence of an additional active Peer. The downside of this countermeasure is that during a short period of time, there exists more than one active Peer. As a result, a small amount of network traffic might be wasted since additional active Peers might send out exactly the same data (e.g., from the same Local Storage) as the first active Peer, albeit using different nicknames. Moreover, the countermeasure slightly adds to load of Signaling Server as there are redundant nicknames to manage, although all of them but one will vanish soon as a result of surrender of active status.

(2) Ability to tackle uncleared flag in Local Storage after a Browser crash

The proposed system is immune to such junk data as uncleared flag since a dormant Peer treats it the same way as it would when encountering a flag set by another dormant Peer. Specifically, after Browser is launched again after a crash, the first Peer of an origin with an uncleared flag tries to set a flag as usual, and detects that a (junk) flag already exists. The Peer therefore does nothing but wait for a predefined period of time to see if the flag
is still there. Since no actual active Peer exists to clear the flag, the Peer in question simply clears it by itself and becomes active shortly thereafter.

4.2.10 Local Screening

User is entitled to local screening of downloaded translation, which is achieved by blacklisting. The blacklisting performed on one node may affect other nodes, because the blacklisted translation is temporarily removed from Overlay Network. Yet it is not decisive or final, as every node has absolute control over its own Local Storage. The blacklisted translation might be stored in Local Storage of another node, which is likely to upload it again to Overlay Network.

4.2.11 Context-sensitive Translation

Since translation data are stored in origin-specific Local Storage, User is able to choose different interpretations for the same text on different websites. This helps address polysemy as it takes into account context difference. This is particularly true in case of initialism. For instance, “ATM” could mean “Automated Teller Machine” on a bank’s website. On a tutorial website featuring network technology articles, however, “ATM” could mean “Asynchronous Transfer Mode” instead.

4.2.12 Triggering Translation Synchronization

Translation synchronization is triggered if User allows sharing, Peer is connected to Overlay Network, and one of the following is true.
• A chosen translation is being applied to a web page. This can occur either when a web page affected by previously chosen translation is being loaded or when User saves a translation different from the current one (if any) through Translation GUI.

• Peer has just recovered from a disconnection from Overlay Network. This happens whenever the number of contacts in Peer’s routing table increases from 0 to a positive number. Local changes made through Translation GUI when Peer was not connected need to be synchronized with Overlay Network.

• User hovers over an eligible web element and one locale is selected in Translation GUI. Synchronization is needed at this moment as we hope to present most recent translations in DHT with respect to that selected locale to User.

4.2.13 Data Import and Export

The proposed system is able to provide convenient import and export features for User data restoration and backup. User may export translation in Local Storage to a JSON file or import previously exported JSON files to Local Storage through Translation GUI.

The export and import feature can also come in handy when User wishes to perform numerous edits on translations pertinent to a website all at once in his or her favorable text editor in case the website has undergone a major interface change (e.g., changing all hyperlinks in its navigation menu). User may also batch-process exported translation files with external software and imported them back to Browser after batch-processing is complete. User is also free to email
an exported file to another User to realize translation of a whole website in seconds through import.

4.2.14 Preferred Locale

Although User is free to assign a \textit{lclC} to each translation, the preferred locale (\textit{lclP}) can also be specified as a global default. In the event that User does not explicitly specify a \textit{lclC}, \textit{lclP} can be used as a fallback. Typically, \textit{lclP} is User’s native language in the form of locale. Moreover, \textit{lclP} is used as a prefix in User nickname generation, which helps Signaling Server bootstrap a new node and hence achieve peer clustering, grouping Users with similar language background.

4.2.15 JSON Data Structure

As stated earlier in Section 4.2.8, data stored in Local Storage and Global Storage are in JSON format. The reasons why we choose JSON over other serialization formats such as XML and YAML [136] are as follows.

- JSON derives from JavaScript, which is the primary programming language employed in the proposed system. On that account, JSON is a natural and rational choice.
- JSON is faster and less resource-intensive than XML [137].
- YAML, as a superset of JSON, is more complex and slower than JSON [138].
In the proposed system, JSON’s default character encoding, i.e., UTF-8 [139], is used to encode translation data. UTF-8 is capable of encoding all Unicode [140] characters and has been adopted by more than 87.5% of all websites [141].

4.3 Tackling Challenges

There are several challenges in system design that must be tackled. Those challenges include server-assisted connectivity, incentives to contribute, browser integration, system responsiveness, peer clustering, extensibility, security, and privacy.

4.3.1 Server-assisted Connectivity

Although data are stored in a distributed fashion, the proposed system cannot function well without assistance provided by servers.

**WebRTC signaling server**

Although WebRTC is well suited for real-time collaboration, it needs a signaling server to attain optimal performance, e.g., fast peer discovery.

**Nodes behind proxies and firewalls**

Nodes behind proxies and firewalls may rely on some NAT traversal technique (e.g., STUN server) to stay fully functional. Fortunately, WebRTC can be combined with ICE to get around NAT.
4.3.2 Incentives to Contribute

Contributing translation is not economically profitable for any participating User. Yet there exist other incentives to contribute.

Prospective incentives

- Ability to personalize web experience if a website does not provide an interface in User’s native language, or User is not satisfied with the provided interface.
- Ability to share translation works in real-time through reciprocal exchange of data. This is extremely important when a website is not able to provide translation for time-sensitive content such as breaking news.
- Interest and altruism, such as that in Wikipedia contributors. This is mostly seen on entertainment websites such as IMDb.

Free riding

It is entirely possible that Users only use translations locally rather than share them, which is closely correlated with a typical problem in peer-to-peer network design, namely, free riding.

Since translation sharing is voluntary, there is no way to force User to translate. However, Users who never translate can also contribute by using, or rather, endorsing existing translation. Recall that translation synchronization takes place whenever a chosen translation is being applied to a web page. Put another way, when User chooses to apply existing translation contributed by others, he or she is republishing that translation to Overlay Network, casting an implicit vote for
the translation in question. That implicit endorsement can be regarded as a form of contribution
that substantially counteracts free riding.

4.3.3 Browser Integration

Since the proposed system is supposed to work on any website, it has to be integrated into
Browser. One of the major merits of web-based applications is minimum installation and
maintenance on the client side. Therefore, the integration should be as effortless as possible on
User’s part. Specifically, factors worth considering are as follows.

• Transparent Local Storage: Browser should be able to store and retrieve translation
  locally (e.g., on User’s HDD or SSD) without human intervention.

• Fault tolerance: User should be able to store and retrieve translation even when
disconnected from Overlay Network.

Implementation-wise, there are three candidates worth considering, namely, bookmarklet,
userscript and browser extension. All of them have advantages and disadvantages, as listed in
Table 3. Bookmarklet has the weakest functionality among the three because it cannot invoke
itself as a daemon program and it cannot update itself automatically, both of which adds to
difficulty in usability and maintenance. Browser extension is more powerful than userscript in
terms of functionality as it can make use of privileged libraries. However, its inability to work
across browsers complicates both development and maintenance, as a separate extension has to
be maintained for each kind of Browser the proposed system is supposed to support. Besides,
User has to install disparate browser extensions when he or she is using more than one kind of
Since our primary goal is to minimize installation and maintenance labor, userscript stands out as the best choice.

<table>
<thead>
<tr>
<th></th>
<th>Bookmarklet</th>
<th>Userscript</th>
<th>Browser Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Libraries with privileges?</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>As a daemon program?</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Cross-browser?</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Lightweight?</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Automatic update?</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Easy maintenance?</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

### 4.3.4 System Responsiveness

Given sufficient Internet bandwidth, translated text (if available) should be fetched seconds after User explicitly requests specific translation (e.g., hitting a “Translate” button in Google Translate), or after a web page is fully loaded in case User opts to translate every web page. Longer network latency is inclined to degrade user experience. Loading locally stored (yet possibly outdated) translations can be an effective fallback to eliminate noticeable delay. In the proposed system, this issue is addressed by Translations and Local Storage.

### 4.3.5 Peer Clustering

It would be desirable to cluster Users with the same preferred locale (e.g., native Spanish speakers in the United States) so as to facilitate spread of translation in the same language,
allaying relative scarcity of resources. However, there are two caveats we need pay attention to. First, the clustering should not encumber efficiency of the entire system. Second, the clustering should not be limited to neighbors in terms of geolocation, although it is plausible that people living in the same area tend to speak the same language.

In the proposed system, the caveats above have been taken into account as prefixing User’s nickname with the locale of User’s choice is both efficient and geolocation-unrestrictive.

### 4.3.6 Extensibility

The number of locales/languages supported by the proposed system is unlimited since the proposed system is driven by human translators. When we wish to add a new locale, the only thing we need to do is to add an entry for the new locale in the Translation GUI. If necessary, non-locale entries such as romanization of Chinese can also be added to Translation GUI.

### 4.3.7 Security

With a distributed translation memory [142], the proposed system is prone to network attacks unprecedented in centralized translation services such as TransSearch [143].

**Node ID collision**

Although the probability of SHA-1 collision is extremely low (e.g., $1 - (2^{160} - 1)/2^{160} < 10^{-48}$ for two random hash function calls), it is still possible. However, node ID collision is not a problem in the proposed system because a node will switch to another random ID if it detects a node with identical ID.
**Data validation**

Data should be validated when being loaded into Volatile Memory, either from Local Storage or from Overlay Network. Data validation is necessary for the following reasons.

First, Local Storage is shared by all programs that have access to its origin. A malicious program visiting that origin may try to tamper with data stored in Local Storage. There is no guarantee data being retrieved are the same as what has been previously saved. Second, Local Storage is part of Non-volatile Memory, which may contain bad sectors or blocks, especially after a long time of hard usage. Third, data received from Overlay Network may be corrupted, particularly when the node that data originate from has been compromised. It is worth pointing out that we do not validate semantics (e.g., accuracy of translation) but syntax (e.g., format) of data.

**Levels of translation data credibility**

Translation data stemming from different modules should be treated with different levels of credibility. In the proposed system, a three-level hierarchy of credibility is employed.

- High level: Translations module in Volatile Memory
  
  Translation data in Translations module have already been validated and are very unlikely to be corrupted. Therefore, they deserve a high level of credibility.

- Medium level: Local Storage in Non-volatile Memory
Local Storage can be accessed by all programs visiting the same origin and hence is subject to the risk of being modified by a malicious program. Hence, translation data originating from Local Storage should always be validated before being loaded into Volatile Memory. Since detection of modification is possible thanks to the notification mechanism provided by `localStorage`, however, translation data in Local Storage are granted a medium level of credibility.

- Low level: Translation data received from Overlay Network

There is no practical way of determining whether data from Overlay Network are corrupted. (Checksum is not a viable option because a compromised node may forge a valid checksum before sending corrupted data.) Consequently, all translation data received from Overlay Network are associated with a low level of credibility and must be validated before being loaded into Volatile Memory.

Credibility level serves as a guideline for data recovery in the event of data inconsistency between two or more modules. For example, when a mismatch of data is detected between Translations and Local Storage, data in Local Storage will be overwritten with data in Translations because the latter module has a higher level of credibility.
4.3.8 Privacy

Pseudonymous sharing

Since nicknames are generated at random, User’s true identity can never be revealed. By “pseudonymous”, we mean that sharing of translation is not completely anonymous because only preferred locale of User would be exposed.

As mentioned earlier in Section 4.2.8, when a node sends a download request into Overlay Network, SHA-1 hash of the key in question is sent in place of the key itself, thus concealing the content User is viewing and hence protecting privacy.

Optional sharing of translation works

In the proposed system, User has full control over which piece of translation can be shared with others in Overlay Network. The sharing option is initially enabled for all translations so as to allay resource scarcity. Nevertheless, Users concerned with privacy are free to turn off that option for any piece of translation at any time.

4.3.9 Expected Results

In brief, we aim at enabling human users to collaboratively translate a web page utilizing modern browser technologies without storing data on any centralized server. The deliverables are as follows.
A distributed online dictionary transforming any website on the fly

We expect to deliver a powerful web-browsing utility that does in-place translation using a distributed and comprehensive multilingual corpus [144], which resembles Wiktionary [145] in terms of user collaboration.

A pseudonymous community of voluntary translators

Professional and amateur translators with diverse cultural backgrounds are able to cooperate and socialize in a pseudonymous manner, while making a concerted effort to break down cultural barriers for all humanity.
Chapter 5

Results

We implemented a prototype of the proposed system that can function across web browsers on different computers. The prototype consists of a userscript generated by Browserify with client code as input and a secure (TLS) signaling server. All code was written in JavaScript, including open-source third-party libraries such as KadTools [146].

5.1 Translation GUI

Fig. 41 shows a screenshot of our Translation GUI in action. The Translation GUI implemented consists of several components, which are listed below.

Fig. 41: Screenshot of integrated Translation GUI on a web page
• Eligible web element

The Translation GUI is shown whenever the mouse cursor is hovering over an eligible web element. An eligible web element must meet the following requirements.

• It must contain non-whitespace text.

• It must not contain anything that is not text.

• It must take some space on web page; otherwise it would be impossible to hover over it.

The current eligible web element will be surrounded with highlighting dots so that it becomes conspicuous, as is illustrated in Fig. 41.

• Key Source and matching level

The Key Source is displayed at the left-top corner of Translation GUI. In Fig. 41, the mouse cursor is placed over a hyperlink, i.e., “http://www.cnn.com/us”. Yet the level chosen is “Text Only”, as is shown on the top-right corner. Therefore, the hyperlink “http://www.cnn.com/us” is not considered and the Key Source is simply “U.S.”, where the space preceding “U.S.” signifies that its matching level is “Text Only”. Whenever User chooses a different matching level, Key Source changes accordingly. Fig. 42 and Fig. 43 show Key Source values for matching levels of “Link+Text” and “Link Only”, respectively.
If the web element User is hovering over is text, none the less, “Text Only” will become the only selectable option and “Link+Text” and “Link Only” will be grayed out, as is shown in Fig. 44.

- Chosen locale and search box for chose locale

The chosen locale (lc|LC) represents the language that User wants some foreign text to be translated to. As stated in Section 4.1.4, it can be composed of a language code and a region code, joined with a hyphen. On the Internet, however, it should be converted to some human-readable format such as “English (US)”. The lc|LC in Fig. 41 is “Chinese”
used in mainland China. When User resets translation of a web element (e.g., a hyperlink) via the “Reset” button, lclC will be set to “-----”, signifying that User intentionally opts not to translate that web element. The implemented prototype has been preloaded with 166 locales pertaining to 97 languages. Users can quickly find his or her preferred locale with the help of a convenient filter-as-you-type search box. In Fig. 41, User typed “ch” in search box and were instantly presented all the locale options that contain “ch”.

• Chosen target HTML and target blacklisted HTML

The chosen target HTML (tgtHtmC) is the translation User has chosen for the combination of a Key Locale (i.e., lclC in Translation GUI) and a Key Source. By “chosen”, we mean that the translation is either picked from a list of existing translations retrieved from Overlay Network, which we refer to as target HTML (tgtHtm), or directly input by User via keyboard and mouse. The tgtHtmC in Fig. 41 is “美国”, the most common translation for “United States”. The text box containing tgtHtmC is placed to the right of search box for the chosen locale. The tgtHtm in Fig. 41 have two members, namely, “美国” and “美利坚合众国”. The former is an accurate translation of “U.S.” and has been chosen by User, whereas the latter is a literal translation of “United States of America” and hence is not chosen. In Fig. 41, there is another member placed below all members of tgtHtm, namely, “我们”. That member belongs to target blacklisted HTML (tgtBHtm), a collection of translations that User considers inappropriate. “我们” is a literal translation for “us”, which is obvious not the correct interpretation in the given
context. As it has been blacklisted by User, it is shown in strikethrough format. Members from both $tgtHtm$ and $tgtBhtm$ are displayed as a list below the text box containing $tgtHtmC$. None the less, User can choose to hide members from $tgtBhtm$ by checking the “Filter On” checkbox.

- “Save” button and “Share” checkbox

“Save” button saves $tgtHtmC$ to both Translations (in Volatile Memory) and Local Storage (in Non-volatile Memory). When “Share” is ticked, which signifies User is willing to share specific translation with others, it also synchronizes changes with Overlay Network. After User presses the “Save” button in Fig. 41, the translation will be applied to the current page immediately, as is shown in Fig. 45. Note that “Reset” button

![Fig. 45: Screenshot after “Save” button is pressed in Fig. 41](image)

now becomes available (as opposed to its grayed-out state in Fig. 41), giving User a chance to revert his or her chosen target HTML. Also note that “Share” checkbox to the right of “Save ” button is not ticked, implying that User refuses to share the translation in
question and hence nothing will be synchronized with Overlay Network the moment “Save” button is pressed.

- “Blacklist” button

“Blacklist” button hides a member of tgtHtm that User disapproves of. It also moves that member from tgtHtm to tgtBHtm in both Translations and Local Storage. Although the member is not present in Translations momentarily, it might be loaded again into Translations during next synchronization with Overlay Network, particularly when another User, who approves that member, shares it again. Even in that case, however, that member will not be shown again because it is blacklisted according to tgtBHtm.

- “Filter On” checkbox

“Filter On” checkbox can be unticked to show otherwise hidden tgtBHtm members so that User has the option of reversing blacklisting by pressing “Save” button on a blacklisted member. In other words, it makes it possible to move members of tgtBHtm back to tgtHtm. When “Filter On” checkbox is unticked, tgtBHtm members are shown in strikethrough format. When it is ticked again, all blacklisted members will revert to hidden state, as shown in Fig. 46.

- “Reset” button
“Reset” button forces the proposed system to “forget” the translation pertaining to current Key Source. It restores any modified text of current Key Source on the web page back to its original state as if no translation had ever been performed. It also clears chosen locale and sharing status. Fig. 47 shows a screenshot after “Reset” button is pressed in Fig. 46.

*Fig. 46: Screenshot after “Filter On” checkbox is ticked in Fig. 45*

*Fig. 47: Screenshot after “Reset” button is clicked in Fig. 46*

Note that the text box intended to hold chosen target HTML is disabled and prompts User
to “select a language” because \textit{lclC} has been cleared and no locale is chosen in the list of
locales. Also note that the filter “ch” in search box is still working, i.e., the list of locales
only shows locales containing “ch”. The “Set Locale” button is disabled as no locale is
selected.

- “Set Locale” button

“Set Locale” button is used to change the preferred locale (\textit{lclP}). Suppose User is a native
Chinese speaker and wishes to set “Chinese (Mainland China)” as default so that he or
she need not key in “ch” in the search box each time he or she translates. After User
selects “Chinese (Mainland China)” and presses “Set Locale”, it becomes the preferred
locale and will be selected automatically next time Translation GUI is open, unless one or
more of the following is true.

- User explicitly saves translation in a locale other than “Chinese (Mainland China)”.
- User presses “Reset” button on a certain Key Source.
- User presses “Reset All” button.

It should also be noted that “Set Locale” button also changes the nickname in Local
Storage, which suggests all Peers from the same origin would be notified and the active
Peer would have to reconnect to Signaling Server using a different nickname. (Recall that
\textit{lclP} constitutes the first 5 bytes of nickname.) Obviously, this is no minor change and
should be confirmed by User so as to guard against accidental click. It is worth
emphasizing that the confirmation dialog illustrated in Fig. 48 is unlikely to pose a
nuisance because User would not wish to change the preferred locale frequently unless he or she knew many languages and needed to switch between them on a day-to-day basis.

**Fig. 48: Screenshot when clicking “Set Locale” button in Fig. 45**

If User confirms the “Set Locale” operation, the preferred locale will be changed to the currently selected one and the new preferred locale will be highlighted in bold, as in shown in Fig. 49.

**Fig. 49: Screenshot after “Set Locale” is confirmed in Fig. 48**

- “Reset All” button
“Reset All” button revokes all changes the proposed system has made to the current origin, deleting all related data in Local Storage. After deletion is complete, the current web page is reloaded. This is obviously a major action and confirmation illustrated in Fig. 50 is needed to guard against accidental click.

![Screenshot](image)

*Fig. 50: Screenshot when “Reset All” button is clicked in Fig. 49*

- **“Export” button**

User may use “Export” button to backup all translation works pertaining to an origin as a JSON file. The exported file is named after the current origin, with non-alphanumeric characters replaced by underscores. For example, the file exported on “http://www.cnn.com” is named “http___www_cnn_com.json”. User can save the exported file to Non-volatile Memory like HDD and SDD, so that it can be later imported into the same (e.g., a newly installed web browser) or another device (e.g., sharing with another User via email). A sample exported file is shown in Fig. 51. The “synInt” parameter has a value of “-1”, indicating the translation is not shared.
“Import” button

The “import” button is used to import any JSON file exported by the “Export” button to both Local Storage and Translations. When it is clicked, a file picker dialog pops up, prompting User to pick files to be imported. After the import operation is complete, the web page will be modified as per imported content.

5.2 Secure Signaling Server

We implemented a Signaling Server secured by TLS (Transport Layer Security) [147]. The TLS support is rendered by a Node.js WebSocket library [148]. A self-signed certificate [149] for the secure Signaling Server can be generated for free with open-source tools like OpenSSL [150] [151]. For example, we may use the following command to generate a key pair and certificate signed by the private key in the key pair.

```
openssl req -x509 -newkey rsa:2048 -keyout key.pem -out cert.pem -days 365 -nodes
```

Now we briefly explain core parameters in the command above.

- “x509”: a standard for managing certificates [152].
- “rsa”: generating key pair using RSA algorithm [153].
- “2048”: length of the composite number used in RSA algorithm, in bits.
• “key.pem”: output file containing private key, in PEM format [154].

• “cert.pem”: output file containing certificate, in PEM format.

• “days”: number of days the generated certificate is valid for.

• “365”: the generated certificate is valid for 365 days, or one year.

• “nodes”: not encrypting private key. (The default is encrypting with Triple DES [155].)

The generated private key and certificate are then placed in a directory accessible by the secure Signaling Server. Browser wishing to connect to the secure Signaling Server needs to import certificate out of band and accept the certificate before connecting. Otherwise Browser would be subject to man-in-the-middle attack [156]. In a realistic production environment, however, this is never an issue because a certificate authority, whose public key has been shipped with operating systems, can always guarantee a certificate signed by it is authentic. For experimental purposes, none the less, the out-of-band transfer of self-signed certificate is acceptable.

If needed, the secure Signaling Server may be deployed as a server publicly accessible over the Internet so that it can relay messages between Peers behind proxies or firewalls. In that case, the secure Signaling Server plays the role of TURN server as well.
5.3 Evaluation

We set up a testing environment to evaluate the effectiveness and efficiency of the proposed system. Both GUI and scalability tests have been carried out. Experimental results are analyzed and a summary of results is given.

5.3.1 Testing Environment

Hardware

A laptop connected to the Internet is used to evaluate the implemented prototype. Specifications of the laptop are as listed in Table 4.

*Table 4: Specifications of the laptop used in evaluation*

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel Core i7-4700MQ CPU @ 2.40GHz</td>
</tr>
<tr>
<td>RAM (Volatile Memory)</td>
<td>16GB</td>
</tr>
<tr>
<td>Hard Disk Drive (Non-volatile Memory)</td>
<td>1TB</td>
</tr>
<tr>
<td>Network Interface</td>
<td>Intel Wireless-N 7260</td>
</tr>
</tbody>
</table>

Software

Major software used in evaluation are listed in Table 5.
Table 5: Software environment

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Debian 4.6.4-1 (2016-07-18) x86_64 GNU/Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browser</td>
<td>Chromium 53.0.2785.143</td>
</tr>
<tr>
<td></td>
<td>Firefox 49.0</td>
</tr>
<tr>
<td>Node.js</td>
<td>v4.4.7</td>
</tr>
<tr>
<td>Userscript Manager</td>
<td>Tampermonkey v3.13.4890</td>
</tr>
<tr>
<td></td>
<td>Greasemonkey v3.9</td>
</tr>
</tbody>
</table>

5.3.2 GUI Test

Preprocessing time

To facilitate and automate translation, we turn each piece of non-whitespace text [157] into an eligible web element (e.g., SPAN [158]) that can be translated separately and instantly. This procedure, which we call preprocessing, takes place automatically and immediately after a web page is loaded. If a match of translation is found in Local Storage during preprocessing, the piece of text in question is replaced by the matched translation. The preprocessing can take a few seconds if the web page in question is very long.

We tested the implemented prototype in terms of preprocessing time on the longest [159] article [160] on English Wikipedia, which spans 333 pages (when printed) and contains 22,318 pieces of non-whitespace text. The preprocessing time for that web page is approximately 5 seconds. It is worth pointing out that the web page remains responsive (i.e., User is able to interact with all web elements) in the preprocessing phase and consequently user experience is not degraded. Given that an average web page on the Internet spans only dozens of pages, the
typical preprocessing time of the implemented prototype would be less than 1 second and hence hardly noticeable.

**Interactive replacing time**

When User enters some new translation for a piece of text interactively on Translation GUI, all occurrences of the same text on the web page are replaced by the new translation. The replacement may take a while if there are multitudes of occurrences to be replaced. We tested our implemented prototype by replacing 338 occurrences of "MC" with "Military Cross" on the aforementioned Wikipedia article [160] and the replacing time is approximately 1 second. This is an acceptable delay, given the large number of occurrences replaced. Again, the web page remains responsive in the replacing phase and consequently user experience is not degraded.

### 5.3.3 Scalability Test

**Setup**

Two browsers containing three tabs each are used to conduct experiments. (See Fig. 52.) Each tab contains $n$ peers, that is, one interactive peer and $n-1$ hidden peers in $n-1$ embedded iframes [161], respectively. Each of the three tabs in a browser has a web address of example.com, example.net and example.org, respectively. A peer joins Overlay Network by connecting to a random seed. Experiments are carried for different numbers of peers in Overlay Network, at intervals of around 100. (The interval varies slightly as a multiple of 6 has to be chosen as a result of a total of 6 tabs.) In each experiment, a total of of 30 (i.e., $2P_0$) attempts are
made between all possible pairs of the 6 tabs, which remain open simultaneously throughout the experiment. For each attempt, the system response time (i.e., the time elapsed between input of a piece of translation in one tab and output of the same piece of translation in another) is measured. If the system response time is longer than a minute in one input/output attempt, the attempt in question is considered a failure due to timeout. Memory consumption per peer is also recorded for each experiment.

![Fig. 52: Scalability test setup](image)

**Experimental results**

As is shown in Fig 53, experiments were carried out when the number of peers was set to 102, 204, 300, 402, 504, 600, 702, and 804, respectively. The average time elapsed between store (i.e., input) and retrieval (i.e., output) never exceeds 5 seconds, which signifies that the proposed system remains responsive even when confronted with 804 peers and hence qualify for real-time collaboration between hundreds of users. The memory used per peer peaked at 13MB and had
been declining ever since. The bounded memory consumption per peer suggests that given a known number of peers, memory requirements of the proposed system are always predictable. No retrieval failures occurred when the number of peers stayed below 500. When the size of Overlay Network grew beyond 500, however, there is a growing tendency for retrieval timeouts and one-third retrievals failed at the size of 804.

![Graph](image)

**Fig. 53: Experimental results of scalability test**

**Discussion of scalability test results**

The increase in retrieval failures with growth of Overlay Network results from escalating sparsity of DHT in terms of data distribution, that is, DHT is getting so sparse that scattered stale
data have little chance of being overwritten by the most recent store operation in a timely manner.

None the less, retrieval failures can be mitigated by users’ endorsement of the good translation, which propagates that specific translation to a broader audience and hence leaves little room for stale (and hence less valuable) data to survive. In other words, the rationale of the proposed system is “Good translation travels fast.”

5.3.4 Summary

The experimental results show that the implemented prototype has achieved decent overall responsiveness and the interactive delay introduced is acceptable even in heavy-duty tasks. The proposed system is expected to scale well as long as most web users make rational choices (i.e., endorsing good translation) when using the proposed system.
Chapter 6

Conclusions and Future Work

6.1 Conclusions

After analyzing drawbacks of existing online translation services, we implemented a browser-based system that bundles enabling technologies such as WebRTC, Node.js, Kademlia, userscript into a decentralized solution for interactive and collaborative web translation. Design of the proposed system follow guidelines in regard to usability, robustness, responsiveness, stability, and security. Specific considerations include coordinated multi-tab web browsing, web browser compatibility, multi-level matching for web elements, fault tolerance upon disconnection from overlay network, context-sensitive translation, caching of synchronization requests, fine-grained control of privacy, and convenient data backup and restoration. As regards deliverables, a graphical user interface for translation is seamlessly integrated into web pages, and a secure signaling server is incorporated to facilitate peer discovery and to group peers with similar language preferences. The implemented system is able to function across multiple tabs on disparate web browsers. Experimental results demonstrate that the implemented system exhibits acceptable responsiveness even when confronted with extremely long web pages, stressful tasks and large numbers of web users. The proposed system is expected to enable multilingual netizens to share idiomatic translation through real-time collaboration and, more importantly, deliver a
personalized experience to anyone that is fond of devouring foreign cultures in a web browser window.

6.2 Future Work

6.2.1 Supplement to Crowdsourced Moderation

Current design of the proposed system relies on human users for moderation of translation works (e.g., blacklisting). Nevertheless, human moderation may not suffice once the system gains popularity and hence becomes a valuable target for spammers. Nowadays, anti-spam solutions such as SpamAssassin [162] are readily available but it remains a question if they can be properly integrated into the proposed system without sacrificing responsiveness or degrading user experience.

6.2.2 Paradigm for Distributed Knowledge Sharing

Unlike centralized knowledge bases such as Wikidata [163] and Google Knowledge Graph [164], the proposed system is aimed at distributed knowledge sharing. Translation data as the current focus is, the proposed system may serve as a paradigm for distributed sharing of all kinds of knowledge. For example, one may replace translation with statistics of a country. When a mouse cursor is hovered over a country name on a web page, a dialog could be popped up, displaying statistics of that country, such as area and population (instead of languages such as
English and Spanish). However, the possibility of transforming the proposed system into a generic platform for distributed knowledge sharing is yet to be explored.
Bibliography


[28] Information Technology Laboratory, “Secure Hash Standard (SHS)

bittorrent mainline dht”, 2013 IEEE Thirteenth International Conference on Peer-to-Peer
Computing (P2P), pp. 1-10.


[31] Lachèze, A. and Guillemiot, P., “Kademlia over HTTP, a Javascript framework bringing
DHT to mobile applications”, http://jinroh.github.io/kadoh/.


[34] Paterson, I., Smith, D., Saint-Andre, P., Moffitt, J., Stout, L. and Tilanus, W., “Xep-0124:

language (XML) 1.0”, https://www.w3.org/TR/2008/PER-xml-20080205/PER-xml-
20080205.pdf.


[58] Mahy, R., Matthews, P. and Rosenberg, J., “Traversal using relays around NAT (TURN): relay extensions to session traversal utilities for NAT (STUN)

[59] Dutton, S., “WebRTC in the real world: STUN, TURN and signaling”,


[64] mozilla.org, “Use bookmarklets to quickly perform common web page tasks”,


[88] Hosaka, T.A., “Facebook asks users to translate for free”,


[95] Google Translate Blog, “A faster, simpler, and safer browser goes polyglot”,


https://support.google.com/websearch/answer/1687222.


2010 International Conference on Natural Language Processing and Knowledge Engineering
(NLP-KE), pp. 1-10.


conversion of scripts”, Institute of Modern Languages.

Wiley & Sons.


https://www.w3.org/TR/webstorage/#the-localstorage-attribute.


https://www.w3.org/TR/webstorage/#the-sessionstorage-attribute.


http://www.iso.org/iso/country_codes.


[155] Barker, W.C. and Barker, E.B., “Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher”,


http://spamassassin.apache.org/.


[164] Google Inc., “Introducing the Knowledge Graph: things, not strings”,
https://googleblog.blogspot.co.uk/2012/05/introducing-knowledge-graph-things-not.html.