Litter: A Lightweight Peer-to-Peer Microblogging Service

Pierre St Juste, David Wolinsky, P. Oscar Boykin, Renato J. Figueiredo
Advanced Computing and Information Systems Lab, University of Florida, Gainesville, FL 32611
Email: {pstjuste,davidiw,boykin,renato}@acis.ufl.edu

Abstract

Microblogging has become an important part of the social web evolution and is being utilized in many aspects such as advertising, political campaigns, and popular uprisings. Due to its heavy centralization, many have proposed decentralized alternatives based on a variety of models. This paper suggests a fully distributed approach built on top of existing peer-to-peer technologies. We demonstrate that, by exploiting the services of current peer-to-peer middleware along with the properties of the social graph, it is possible to create a simple, yet practical microblogging service that is impervious to many of the shortcomings of their centralized counterparts. The approach has been implemented as a software prototype that is readily available for download in order to test our design in real life environments.

1. Introduction

Microblogging, which began as a simple means of publishing your statuses, has now become a primary mechanism for publishing information in a concise, and practical manner; for example, microblogs are used in advertising, stock analysis, news reporting, political campaigns, and revolutionary uprisings. Hence, microblogging is being accepted as an efficient method of disseminating information to the masses by leveraging the social web. However, most microblogging activities occur in a few well-known services such as Twitter or Identica, which has prompted the research and free software community to start thinking about decentralized alternatives [1]–[4]. Unfortunately, no practical implementation has ever materialized.

This paper aims to be the first to describe a practical distributed microblogging service which is built on top of peer-to-peer technologies, making the system self-sustaining without dependence on centralized servers. More specifically, we focus on leveraging readily available peer-to-peer, virtual private networking technology such as Hamachi [5], Wippien [6], or SocialVPN [7] to minimize the complexity of our microblogging service. This paper demonstrates that the system requirements of a microblogging service are relaxed enough so the service can be deployed on a dynamic peer-to-peer system, such as one dominated by mobile nodes while delivering fairly adequate performance in terms of message latency and bandwidth consumption.

Our paper focuses on the following novel contributions:

- A decentralized microblogging approach which uses best-effort packet delivery without the use of acknowledgments or timeouts for message retransmissions.
- A functional prototype implementation which demonstrates the feasibility and practicality of our design and also works on both typical LAN environments and virtual private networks.

The rest of the paper is outlined as follows. Section 2 discusses the benefits and features adopted from available peer-to-peer networking system. We elaborate on our design in Section 3, followed by the details of our implementation in Section 4. Finally, Sections 5 and 6 present related works and conclusion.

2. Requirements and Assumptions

Peer-to-peer networking and public key cryptography are the two fundamental pillars of our design. This paper does not focus on solving the intricacies of peer-to-peer connectivity because that topic has been researched extensively over the past decade. Therefore, we assume the availability of peer-to-peer systems that enable direct communication amongst peers [8], [9]. One such solution is SocialVPN, a peer-to-peer virtual private network which leverages social networks to bootstrap encrypted IP tunnels through network ad-

978-0-7695-4578-3 2011
U.S. Government Work Not Protected by U.S. Copyright
DOI

dress translators between friends [10]. Our microblogging service relies on this peer-to-peer communication layer to propagate updates to followers. For peer discovery and cryptographic key distribution, we can adopt the Freenet darknet model, which allows users to exchange their endpoint information and public keys through some trusted, out-of-band communication path (e.g. thumbdrive, email, or instant messaging) [11]. Another solution is the SocialVPN approach which utilizes the XMPP federation as the trusted medium for peer discovery and key exchange [10], [12]. Once again, our design assumes the adoption of one or more of these peer discovery and key distribution solutions.

3. The Litter Design

Our design consists of three main components: the message format, a push/pull model for propagating messages, and a message recovery mechanism. In this section, we demonstrate that how our algorithms achieve this and we analyze it in the following section.

3.1. The Message Format

The message form is the first basic component of our design and it contains the following fields: uid, timestamp, sequence number, data, and signature. The uid is the unique identifier for the creator of the post. The timestamp is simply the time of the message creation. Each user maintains a local counter which increments by one for every new post the user creates and also serves as the message's sequence number. The sequence number is key in detecting missing messages in a user's timeline. The data is the actual message created by the user with a 140 character limit. Finally, each message has a unique signature which serves to identify the message, verify its integrity, and confirm its creator. The signature is the typical HMAC scheme of encrypting the hash of the whole message (i.e uid, timestamp, sequence number, and data) with the creator's private key.

3.2. Pushing Messages

Messages propagate through the peer-to-peer social graph through a combination of both a push and pull model. When a user creates a post, that post is sent to followers by broadcasting it to all one-hop peers in the peer-to-peer graph. Every peer who receives the messages stores it locally. Since one-hop peers are also followers, they serve as both consumers of the message as well as caches for the message (Algorithm 1). In the ideal scenario, this basic push model would be sufficient since posts are sent directly to all followers once they are generated. However, there are three main factors which preclude this. The most obvious obstacle is churn in peer-to-peer systems meaning that a node may be offline during the time of the post. Therefore, with the simple push model, followers would never receive posts sent while they were offline. The second obstacle is packet loss. Although, actual packet drop on the Internet is fairly low at about 1%, it has been shown that it can be much higher in particular continents such as Asia or Africa where the networking infrastructure is less developed [13], [14]. In our analysis, we evaluate the performance of our design at both 1% and 10% packet drop rates. The third barrier is resource limitation, or more specifically, bandwidth. It would not be realistic to expect a user with thousands of followers to send each post to thousands of peer-to-peer nodes without exhausting his/her bandwidth. To cope with these issues, our design also employs periodic pulls and a simple message recovery system.

Algorithm 1 Push Mechanism

```
1: procedure PUBLISHPOSTS($post$, $C$, $seq$)
2:     $C$ is a list connections to online followers
3:     $seq := seq + 1$
4:     $msg := (uid, time, seqno, post, sig)$
5:     for all $con$ in $C$ do
6:         sendTo($con$, $msg$)
7:     end for
8: end procedure
9: procedure RECEIVEPOST($msg$, $DS$)
10:    $DS$ is the local datastore
11:    if isValidSig($msg$) and $msg$ not in $DS$ then
12:        storeMsg($DS$, $msg$)
13:        updateHighestSeq($msg$)
14:    end if
15: end procedure
```

3.3. Pulling Messages

Churn is a common side-effect of peer-to-peer design because many of the nodes that make up the system can be mobile. This can include laptops, tablets, smartphones, or any other mobile device where uptime is low and connectivity is dynamic. As a result, mechanisms have to be in place to deal with the unavailability of peers during the time of a post. Our design uses a periodic pull model to handle this issue. Online nodes periodically send a pull request to all the nodes that they have direct peer-to-peer connections with. The pull request contains a time range, a list of unique identifiers of the users being followed, and the highest

Algorithm 2 Pull Mechanism

```
procedure REQUESTPOSTS($C$, $F$, $range$)
    $C$ is a list connections to online followings
    $F$ is a list of $uids$ with highest $seq$
    $range$ is the time window for request
    $msg := (F, range)$
    for all $con$ in $C$ do
        sendto($con$, $msg$)
    end for
end procedure
procedure PROCESSREQUEST($con$, $msg$, $DS$)
    $DS$ is local datastore
    $con$ is connection to requester
    for all $post$ in $DS$ do
        if $post.uid$ is in $msg.uids$ and $post.seq >$ $msg[uid].seq$ and $post.time$ is in $range$ then
            sendto($con$, $post$)
        end if
    end for
end procedure
```

sequence number received from the particular user. Upon receiving the pull request, the node scans its local datastore for any messages from the list of uids in the request, and sends back messages with timestamps that fall within the range of the request (Algorithm 2).

3.4. Message Recovery

Our microblogging service is designed to use a best-effort datagram messaging system because it allows the most flexibility and the least amount of maintenance. We assume a messaging layer where packet drops occur or packets arrive out-of-order. Therefore, both our push-based publishing and our pull-based retrieval of messages are vulnerable to packet loss. The resulting effect of deploying our system on a best-effort medium is gaps in a user's timeline. As previously described, each generated post contains a sequence number which increments with each new post created. Therefore, it is easy to identify missing messages by looking for gaps in the timeline of a particular user. To counteract the message lost, our algorithm periodically scans the local datastore for missing posts. When one is found, a pull request is sent directly to the creator of the missing post if they are online. If the creator of the post is offline, the pull request is sent to one of the creator's followers since there will be a high likelihood that they have the missing post cached.

4. Prototype Implementation

In order to determine the feasibility of our design, we implemented a prototype. To deal with peer-to-peer connectivity, peer discovery, and key distribution, we chose the SocialVPN project (http://socialvpn.org) which provides of all these services. With SocialVPN, users are able to create IP connections by emailing each other their P2P address and the hash of their public key. Alternatively, they can automate that exchange of credentials through the XMPP protocol. Because SocialVPN virtualizes the peer-to-peer layer into a virtual private network, our microblogging service uses the well-known Berkley sockets API to communicate with peers. This also means that our implementation works in a LAN or private network as long as multicast is enabled.

When publishing a post, that message is sent using IP multicast over SocialVPN. Since messages are limited to 140 characters, they are able to fit into one UDP message without IP fragmentation. Therefore, when Alice creates a post, she is able to send it to all her followers by simply encapsulating the message in a UDP datagram and sending it a multicast IP address (i.e. 239.192.1.100, the chosen address for our protocol). As we all know, IP multicast is always a best-effort protocol, meaning that there are no guarantees against packet drops.

As part of our push/pull model, the local Litter node periodically checks for updates from its local connections through the use of IP multicast as well. In our current implementation, a pull request for new messages is done every 5 minutes to check whether the follower has missed any messages. Furthermore, our message recovery mechanism searches for gaps in the sequences of messages every 30 seconds. Whenever a gap is found, the user sends a request to only the nodes that may have that message cached. The pull request also helps each Litter node determine common friends by comparing the list of uids in the request with the user's own list of friends. The prototype has been tested internally in our lab on both the LAN and over SocialVPN. We also made posts through the use of mobile devices such as the iPhone.

5. Related Works

There have been a few proposals for decentralizing microblogging systems. Sandler et. al presented FETHR [2] with a design for a decentralized microblogging service based on an unstructured social graph. Messages were propagated with a gossip mechanism to all followers or application-level multicast trees. Instead of sequence numbers, messages are chained by a list of message signatures. The Cuckoo project [3] seeks to combine both a structured and unstructured peer-to-peer design with a hybrid approach. Similar to our approach, new posts are pushed to followers through direct links and also stored in a DHT. Followers also periodically pull messages from nodes they are following if they have a direct connection with these nodes. If not, they retrieve new messages through the DHT. In contrast, Litter does not have to rely on a DHT if there is enough clustering among social nodes and it implements a message recovery mechanism to deal with lost packets.

6. Conclusion and Future Work

In this paper, we propose a distributed microblogging design which leverages existing middleware, specifically peer-to-peer, virtual private networks. Our approach reuses the peer discovery and key exchange mechanism available in today's peer-to-peer system along with the small-world properties of the social graph. We also described a functional, open-source prototype implementation that can be downloaded and tested to show the feasibility of the system. Future work will focus on getting real live measurements with real users to verify our simulated predictions.

7. Acknowledgements

This research is sponsored by the National Science Foundation under grant IIP-0758596, CCF-0622106, the Amazon Web Services Academic Resource Credits, the South East Alliance for Graduate Education and the Professoriate, the Florida Education Fund under the McKnight Doctoral Program. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] M. Arrington. (2008, May) Twitter can be liberated. [Online]. Available: http://techcrunch.com/2008/05/05/twitter-can-be-liberated-heres-how/

[2] D. R. Sandler and D. S. Wallach. Birds of a fethr: Open, decentralized micropublishing.

[3] T. Xu, Y. Chen, J. Zhao, and X. Fu, "Cuckoo: towards decentralized, socio-aware online microblogging services and data measurements," in *Proceedings of the 2nd ACM International Workshop on Hot Topics in Planet-scale Measurement*, ser. HotPlanet '10. New York, NY, USA: ACM, 2010, pp. 4:1–4:6.

[4] P. Saint-Andre. (2011, June) Microblogging over xmpp. [Online]. Available: http://xmpp.org/extensions/xep-0277.html

[5] "Hamachi - instant, zero configuration vpn." https://secure.logmein.com/products/hamachi/vpn.asp?lang=en.

[6] "Wippien p2p vpn," http://wippien.com/, 2010. [Online]. Available: http://wippien.com/

[7] "Socialvpn," http://socialvpn.org/, 2010. [Online]. Available: http://socialvpn.org/

[8] B. Wang, X. Wen, S. Yong, and Z. Wei, "A novel nat traversal mechanism in the heterogeneous environment," in *Proceedings of the 2009 Eigth IEEE/ACIS International Conference on Computer and Information Science*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 161–165.

[9] S. Guha, Y. Takeda, and P. Francis, "Nutss: a sip-based approach to udp and tcp network connectivity," in *Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, ser. FDNA '04. New York, NY, USA: ACM, 2004, pp. 43–48.

[10] P. St. Juste, D. Wolinsky, P. Oscar Boykin, M. J. Covington, and R. J. Figueiredo, "SocialVPN: Enabling wide-area collaboration with integrated social and overlay networks," *Computer Networks*, January 2010.

[11] I. Clarke, S. Oskar, O. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," in *In Workshop on Design Issues in Anonymity and Unobservability*, 2000, pp. 46–66.

[12] A. V. Ramachandran and N. Feamster, "Authenticated out-of-band communication over social links," in *Proceedings of the first workshop on Online social networks*, ser. WOSN '08. New York, NY, USA: ACM, 2008, pp. 61–66.

[13] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "iplane: an information plane for distributed services," in *Proceedings of the 7th symposium on Operating systems design and implementation*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 367–380.

[14] Y. A. Wang, C. Huang, J. Li, and K. W. Ross. Queen: Estimating packet loss rate between arbitrary internet hosts.