Balancing Gossip Exchanges in Networks with Firewalls*

João Leitão
INESC-ID/IST
jleitao@gsd.inesc-id.pt

Robbert van Renesse
Cornell University
rvr@cs.cornell.edu

Luís Rodrigues
INESC-ID/IST
ler@ist.utl.pt

Abstract

Gossip protocols are an important building block of many large-scale systems. They have inherent load-balancing properties as long as nodes are deployed over a network with a "flat" topology, that is, a topology where any pair of nodes may engage in a gossip exchange. Unfortunately, the Internet is not flat in the sense that firewalls and NAT boxes block many peer-wise interactions. In particular, nodes that are behind a firewall can initiate communication with nodes on the public Internet, but not vice versa. This may easily unbalance the number of gossip exchanges in which nodes are involved. In particular, nodes in well connected regions of the network tend to participate in many more interactions than other nodes and may suffer from resource exhaustion.

In this paper we present and evaluate a new approach to balance gossip exchanges in networks with firewalls. Our solution requires only local information and has no coordination overhead, allowing nodes to participate in a similar number of gossip exchanges independent of the network topology.

1. Introduction

A gossip protocol is a protocol where every participant periodically selects a peer at random to perform an exchange of information. This process is executed repeatedly such that information spreads among all the participants. Gossip protocols have been used to implement a wide range of scalable services.

Most gossip-based protocols have inherent load-balancing properties as long as nodes are deployed over a network with a "flat" topology, that is, a topology where any pair of nodes may engage in a gossip exchange. Most gossip protocols attempt to ensure that when a node selects another node to gossip, that peer is selected uniformly at random from the entire population. This can be easily achieved if each node knows the complete membership [1]. However, even if nodes only have a partial view of the entire membership, this can be achieved if the local views are uniform samples of the population [6].

Unfortunately, in realistic settings, not all interactions are possible. For instance, in the Internet, a large portion of nodes are behind firewalls or boxes that execute Network Address Translation (so-called NAT boxes, which in the remainder of this paper we will consider firewalls as well). This limits the communication patterns that can be established during gossip. If two nodes are behind the same box, we say that they belong to the same *confinement* domain. Also, we denote a node that can be directly accessed from any point in the Internet, without any sort of restrictions, as an *unconfined* node. In practice, a node in a confinement domain can only initiate communication with other nodes in its confinement domain or unconfined nodes. Unconfined nodes can only contact other unconfined nodes. This results in an unbalanced participation of nodes in gossip. As we shall see, such an imbalance can be problematic.

Much work on efficient gossip protocols focuses on reducing bandwidth requirements. The most significant source of bandwidth use is when two peers are gossiping to reconcile their respective states. In the original Clearinghouse paper [3], the authors propose an iterative reconciliation technique, where nodes compare their internal states using hash functions, and exchange the most recent updated they performed until their states become reconciled. Byers, Considine, and Mitzenmacher [2] improve on this design by combining Bloom filters, Merckle trees, and Patricia tries. Trachtenberg, Minsky, and Zippel [14] propose a method based on characteristic polynomials.

While effective in reducing bandwidth, these techniques require significant CPU resources to reconcile state. Besides the computations involved in state reconciliation itself, there is often a non-negligible amount of work required to serialize and deserialize objects that are transmitted, as well as signing and/or encryption in case any level of security is required. For example, in a commercial Java-based deployment of Astrolabe [16], a gossip-based aggregation service that uses Bloom filters and Merckle trees for reconciliation, nodes spend approximately 3% CPU time on all these operations. In a Planetlab deployment of Fire-

*João Leitão and Luís Rodrigues were partially supported by project "Redico" under the FCT grant (PTDC/EIA/71752/2006). Robbert van Renesse was supported in part by NSF TRUST, AFRL Castor, and AFOSR grants.

flies [9], a secure gossip-based overlay network that uses the reconciliation technique of Trachtenberg et al. [14], as well as public key cryptography, nodes use approximately 10% CPU time.

While significant enough, it is worth considering what would happen if there were a relatively small number of unconfined nodes in the presence of a large number of "islands" with confined nodes behind a firewall. The unconfined nodes would have to act as gateways for all information that is gossiped between all participants. In doing so, either unconfined nodes will experience a much larger networking and CPU load than ordinary peers (likely becoming CPU saturated), or by equalizing loads, dissemination times would be significantly increased compared to a "flat" network in which no nodes are confined.

In this paper we present and evaluate a new approach to balance gossip exchanges in networks with firewalls. Our solution requires only local information and has no coordination overhead, allowing all nodes in the system to participate using a similar rate of gossip exchanges, independent of the network topology.

The rest of this paper is organized as follows. Section 2 describes our approach to balance gossip exchanges in networks with firewalls. Section 3 presents experimental results that validate our approach. Section 4 discusses related work and Section 5 concludes this paper.

2. Balancing Gossip

In this section we describe the rationale for our approach and provide a full description of our gossip protocol.

2.1. Rationale

Our approach stems from the observation that the only way to convey information from a node p (in a confinement domain) to another node q (in a different confinement domain) is via an unconfined node u. That is, since p and q cannot communicate directly, it is unavoidable that some unconfined node u acts as a mediator. One way for u to act as a mediator is to engage in two full gossip exchanges. In detail, if node p first engages in a gossip exchange with u and, subsequently, node q engages in another gossip exchange with u, u is able to convey information from p to q. It is possible to alleviate the extra load on u if it merely serves as a router of messages exchanged between p and q, instead of executing the full gossip operation (as we have seen, full gossip can consume a substantial portion of CPU resources).

Therefore, the key idea of our approach is that nodes should have a dual operation mode: they sometimes participate in complete gossip exchanges (to update and propagate their own state) and sometime participate only has routers, forwarding messages exchanged between nodes that would otherwise not be able to communicate. There are several challenges in the implementation of this strategy:

i) Nodes should not be required to figure out if they are confined or unconfined. Ideally, all nodes would simply execute the same algorithm, and the emergent behavior of the system would ensure that a balanced participation in full gossip exchanges would happen.

ii) Nodes should use a localized algorithm to decide when to accept to participate in a full gossip exchange or when to merely serve as a router.

iii) An unconfined node u should be able to route messages between confined nodes p and q despite the fact that it cannot be the initiator of communication to p or to q.

These challenges are addressed in our protocol using the following two complementary techniques:

i) Nodes keep track of how many gossip exchanges they initiated and how many gossip exchanges they have accepted (initiated by other peers). In a balanced network, on average, every node participates in the same number of gossip exchanges initiated by itself and by other peers. Therefore, we define a quota that limits the number of gossip exchanges initiated by other nodes, in excess of the gossips initiated by itself, in which a node participates. The quota is increased when the node initiates a gossip exchange and decreased when it accepts a gossip exchange. When a node runs out of quota it no longer accepts gossip exchanges and simply acts as a router of the gossip request.

ii) Nodes that accept a connection keep the connection to that node open[1]. For instance, if an unconfined node u receives a gossip request from node q, it keeps the connection to q open until it receives another gossip request directly from some other peer. In this way, if node u is later contacted by another node p and its quota has been exhausted, u can route p's request to q.

2.2. Protocol

Algorithm 1 presents pseudo-code for our protocol. We assume that each node in our system has access to a view that contains identifiers of other nodes in the system with whom it can engage in direct communication (i.e., unconfined nodes or nodes within the same confinement domain). The contents of the view is managed by an external peer sampling service, such as [17, 13], whose implementation details are not relevant to our approach.

As noted before, each node owns a quota value, initially set to 1, of gossip interactions it can accept from other peers. Additionally, each node keeps a single-entry *cache* of the connection to the last peer from which it received a gossip messages directly; the connection allows contacting that peer regardless of other connectivity constraints.

When out of quota, our protocol forwards gossip requests. We limit the maximum number of times that a message can be forwarded using a protocol parameter denoted TTL. This avoids network congestion scenarios due to the

[1] The mechanism to maintain this connection open depends on the transport protocol being used.

Algorithm 1: Protocol

```
Internal data:
    cache ⟵ ⊥
    quota ⟵ 1
    view //Managed by an external membership protocol

1:  every △ T do
2:      p ⟵ random(view)
3:      Send(GOSSIP(∅ ∪ me,1),p)
4:      quota ⟵ quota +1

5:  upon Receive(GOSSIP(path,hop)) do
6:      if quota > 0 or hop = TTL or cache = ⊥ then
7:          quota ⟵ quota −1
8:          trigger Deliver(GOSSIP)
9:          n ⟵ last(path)
10:         Send(GOSSIPREPLY(path \ n), n)
11:     else
12:         path ⟵ path ∪ me
13:         Send(GOSSIP(path,hop+1), cache)
14:     if hop = 1 then
15:         cache ⟵ last(path)

16: upon Receive(GOSSIPREPLY(path) do
17:     if path = ∅ then
18:         trigger Deliver(GOSSIPREPLY)
19:     else
20:         n ⟵ last(path)
21:         Send(GOSSIPREPLY(path \ n), n)
```

accumulation of messages in the system whose processing keeps being postponed.

Periodically (line 1) every node tries to initiate a gossip exchange with a peer selected at random from its local view (lines $2-3$) by sending a GOSSIP message. The node also increases its quota (line 4), which will enable engagement in an additional gossip exchange initiated by another peer.

Upon receiving a GOSSIP message from a peer (line 5) a node engages in the gossip exchange if at least one of the following conditions is true: i) it has available quota (i.e. its quota value is above zero); ii) the GOSSIP message has been already forwarded TTL times; or iii) the cache of the node does not contain a connection that can be used to route the GOSSIP message.

If none of the above conditions is met, the node simply routes the GOSSIP message using the connection in its cache (i.e., to the last peer from which it received a gossip messages directly). Notice that the node adds its own identifier to the path associated with this message. This is required to allow the bi-directional gossip exchange between nodes in distinct confinement domains, as the GOSSIPREPLY message has to traverse the inverse path in the network (lines $16-21$)[2].

Whenever a node receives a GOSSIP message directly from its source (i.e, a GOSSIP message that has not been routed, line 14), it updates its cache. This means that the next time the node needs to route a GOSSIP message it will send to a different peer. GOSSIP messages that have been routed or GOSSIPREPLY messages do not update the cache.

We note that there is an interesting symbiosis between the cache and the quota mechanisms that helps in having routed GOSSIP messages quickly accepted. When node u adds to its cache a connection to p, p's quota is known to be greater than 0, as it has just initiated a gossip exchange; therefore, p is likely to still have a positive quota when a GOSSIP request is routed to it.

[2]This requires nodes to keep these connections open for some time, by using an additional cache outside our protocol's scope.

3. Experimental Evaluation

In this section we evaluate the efficacy of our approach. In particular we want to validate that the number of gossip exchanges in which peers engage is balanced across all peers, and assert the costs in terms of dissemination latency and the message forwarding overhead.

3.1. Experimental Setup

We start by describing the experimental setup that we employed, and provide motivation for the network model used in the simulations. We conducted extensive simulations in the Peersim simulator [8], using its event driven engine. In our experiments we simulated 12800 nodes distributed in a variable number of distinct confinement domains that ranges from 1 (the equivalent of a flat network topology) to 12100. For simplicity, we model all unconfined nodes as belonging to domain 1. In each experiment, we ensure that each confinement domain has at least one node, and then we distribute the remaining nodes at random among all domains.

We configured each node with a static view that contains all other nodes in its own domain plus all nodes in domain 1 (i.e., all unconfined nodes). We have evaluated our approach using several values for the TTL parameter in order to assert its impact. In particular, we performed simulations for TTL values of 1, 2, 5 and 10. Notice that a TTL value of 1 prevents a message from ever being routed, and corresponds to a classic gossip protocol, used as a baseline.

The state of the nodes is modeled by a single bit, initially set to 0. When an experiment begins, a random node sets its state to 1. Then, every node periodically gossips this value. When gossiping, nodes execute a simple anti-entropy protocol, in which they set their value to the largest value, between their own and the value received from its peer. This process models, in an abstract manner, the propagation of information in the population.

In each simulation each node initiates 500 gossip exchanges. Every node gossips its value every 10 time units, therefore a simulation takes 5000 units. Each link has a random latency between 2 and 7 simulator time units. All results reported in this paper are an average of 100 independent simulations. Confidence intervals reported in figures where calculated to a confidence of 95%.

3.2. Experimental Results

Figure 1 depicts the maximum number of gossip interactions in which a node participates for the scenarios de-

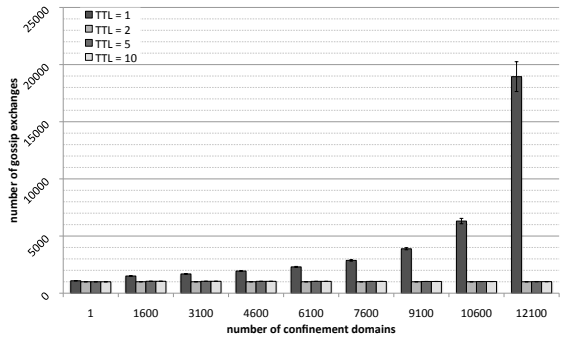

Figure 1. Max gossip exchanges / node.

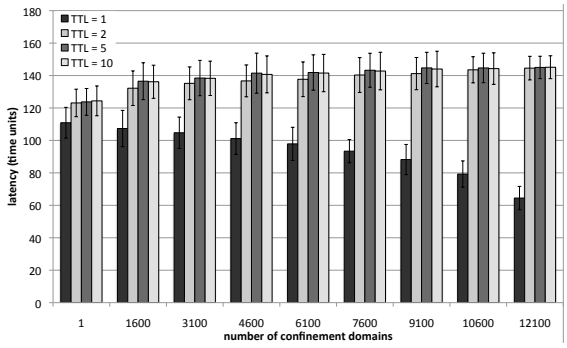

Figure 2. Maximum Latency.

scribed above. Notice that with a flat network (i.e., a single confinement domain) both the baseline (TTL = 1) and our approach behave similarly (for all tested TTL values). The maximum number of gossip interactions in which a single node participates is approximately 1000, which reflects a perfect balance: 500 interactions initiated by the node itself and another 500 that are initiated by other peers.

However, in scenarios with several confinement domains, the maximum number of gossip exchanges in which a single node may participate starts to rise with the baseline protocol. This is expected, as nodes that are in the Internet domain are selected for more gossip exchanges. As the number of domains increases, this effect becomes more visible. This happens because the number of nodes in the Internet domain decreases (given that we maintain the total number of nodes constant in our experiments). In a scenario with 12100 domains, the number of interactions in which a single node may be requested to participate approximates 20000. In sharp contrast, our approach, in all tested scenarios, is able to maintain a constant value which is very close to 1000. This shows that our approach effectively succeeds in balancing gossip interactions in networks with firewalls.

Figure 2 presents how long it takes to infect the entire population. With a flat network our approach presents an increase in latency of approximately 10 time units, that is, a single gossip round. This is observed for all TTL values. As the number of confinement domains increases, the latency increases slightly. This happens because more requests need to be routed, resulting in additional latency in gossip exchanges. Notice however that maximum increase in latency is only of 30 time units (roughly 3 gossip rounds).

Interestingly, as the number of confinement domains increases, the latency of the classic gossip approach decreases. This happens because the number of nodes in the Internet domain also decreases, which leads the system to behave like a centralized architecture. Therefore, the dissemination becomes very fast, by first contaminating the central Internet domain and then, having nodes in all other domains pulling the value from that domain. This is achieved at the cost of overloading unconfined nodes.

Figure 3 depicts results for the maximum number of messages forwarded by a single node. This is a measure of the communication overhead that is imposed by our approach. In the flat network topology our approach presents a negligible overhead, given that there are few requests that need to be routed. As expected, when the number of confinement domains increases, the maximum number of forwarded messages by a single node also increases, as the nodes in the Internet domain are forced to route more requests to avoid being overloaded. Considering that the CPU overhead imposed by forwarding a message is low (the node does not need to deserialize the payload, check signatures, and so on), we believe this overhead is acceptable.

In our experiments the efficacy is mostly unaffected by the TTL configuration parameter. This is because with high probability most requests are accepted in their second hop.

4. Related Work

In this section we discuss various prior work on dealing with the fact that the Internet is not flat. There are essentially two approaches. One approach is to exploit the structure of the Internet, while the other tries to find ways to hide it. In the first approach, many overlay networks, structured and unstructured, have introduced the notion of *superpeers*. Superpeers are nodes that have static, globally addressable IP addresses, are well-connected and exhibit little churn, and are altruistic, generously providing their re-

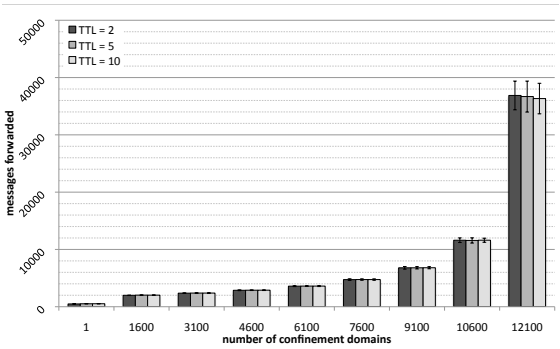

Figure 3. Max forwarded messages / node.

sources for the good of the entire overlay. The popular file sharing service Kazaa (www.kazaa.com) is a good example of an unstructured P2P network that uses superpeers. [18] explores how best to use superpeers in an unstructured network. Various others explored the use of superpeers in structured networks. These works can be subdivided into structured overlay networks that exploit heterogeneity but hide it to users, such as [15], and those overlays that expose the heterogeneity in the network, such as [16].

The other approach is to try to hide the structure of the Internet, so that all peers can directly communicate with one another. Some firewalls support explicit protocols for tunneling, such as [12]. Since this is not widely supported, another option is hole-punching through NAT boxes [10]. Based on hole-punching, Nylon is a gossip-based service that provides each peer with a random sample of nodes that it can communicate with [11]. [4] finds that hole-punching works for UDP in about 80% of cases, and for TCP in about 65% of cases.

Our work can be thought of as combining advantages of both approaches. We use a superpeer approach that does not require any special features of firewalls, but the only extra work that the superpeers do is forwarding traffic. Otherwise, all peers are equal participants. This hybrid approach does not prevent the overlay protocol from exploiting heterogeneity or proximity. Thus protocols that try to exploit heterogeneity such as HEAP [5] can take advantage of our approach to overcome the presence of firewalls, while allowing hosts with high capacity, even confined ones, to do more work than others.

Finally, previous works (such as [7]) have evaluated the performance of gossip protocols over random graphs. Typically these works assume either a regular graph, or that every pair of nodes can exchange messages directly. In our work we are studying the behavior of gossip protocols in scenarios that impose limitations to which nodes can interact directly.

5. Conclusion

We have presented a new approach to balance gossip exchanges in networks with firewalls and NAT boxes. When compared with classic gossip protocols, our approach is able to ensure that all nodes in the system participate in a similar number of gossip exchanges independent of the network topology. Moreover, we have presented experimental results showing that the increase in latency imposed by our solution is acceptably low, and that the communication overhead is acceptable for gossip-based applications that are CPU intensive.

References

[1] K. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM TOCS*, 17(2), May 1999.

[2] J. Byers, J. Considine, and M. Mitzenmacher. Fast approximate reconciliation of set differences. Technical Report 2002-019, CS Dept., Boston University, July 2002.

[3] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proc. of the 6th ACM Symp. on Principles of Distributed Computing*, pages 1–12, Vancouver, BC, Aug. 1987.

[4] B. Ford, D. Kegel, and P. Srisuresh. Peer-to-peer communication across network address translators. In *Proceedings of the 2005 USENIX Technical Conference*, 2005.

[5] D. Frey, R. Guerraoui, A.-M. Kermarrec, B. Koldehofe, M. Mogensen, M. Monod, and V. Quéma. Heterogeneous gossip. In *Middleware '09: Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, pages 1–20. Springer-Verlag, 2009.

[6] A. Ganesh, A.-M. Kermarrec, and L. Massoulié. SCAMP: Peer-to-peer lightweight membership service for large-scale group communication. In *3rd Int. Workshop on Networked Group Communication*, London, UK, Nov. 2001.

[7] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The peer sampling service: experimental evaluation of unstructured gossip-based implementations. In *Middleware '04: Proc. of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 79–98, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

[8] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris. The Peersim simulator. `http://peersim.sf.net`.

[9] H. Johansen, A. Allavena, and R. van Renesse. Fireflies: Scalable support for intrusion-tolerant network overlays. In *Eurosys 2006*, Leuven, Belgium, 2006.

[10] D. Kegel. NAT and peer-to-peer networking, 1999. `http://www.alumni.caltech.edu/~dank/peer-nat.html`.

[11] A.-M. Kermarrec, A. Pace, V. Quéma, and V. Schiavoni. NAT-resilient gossip peer sampling. In *Int. Conf. on Distributed Computing Systems (ICDCS'09)*, pages 360–367, Los Alamitos, CA, 2009. IEEE Computer Society.

[12] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones. rfc1928: SOCKS protocol version 5, 1996. `http://www.ietf.org/rfc/rfc1928.txt`.

[13] J. Leitao, J. Pereira, and L. Rodrigues. HyParView: A membership protocol for reliable gossip-based broadcast. In *Proc. of the 37th DSN*, Edinburgh, UK, 2007.

[14] Y. Minsky, A. Trachtenberg, and R. Zippel. Set reconciliation with nearly optimal communication complexity. *IEEE Trans. Inf. Theory*, 49(9):2212–2218, Sept. 2003.

[15] A. T. Mizrak, Y. Cheng, V. Kumar, and S. Savage. Structured superpeers: Leveraging heterogeneity to provide constant-time lookup. In *The Third IEEE Workshop on Internet Applications*, 2003.

[16] R. Van Renesse, K. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM TOCS*, 21:2003, 2001.

[17] S. Voulgaris, D. Gavidia, and M. Van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2):197–217, June 2005.

[18] B. Yang and H. Garcia-Molina. Designing a super-peer network. *Int. Conf. on Data Engineering*, 0:49, 2003.