Adapting to NAT timeout values in P2P Overlay Networks

Richard Price and Peter Tino
School of Computer Science
University of Birmingham
Birmingham, United Kingdom
Email: $\{R.M.Price, P.Tino\}$@$cs.bham.ac.uk$

***Abstract*—Nodes within existing P2P networks typically exchange periodic keep-alive messages in order to maintain network connections between neighbours. Keep-alive messages serve a dual purpose, being used to detect node failures and preventing idle connections being expired by NAT devices. However, despite keep-alive messages being widely used the interval between messages is typically fixed below the timeout value of most NAT devices based upon crude rules of thumb.**

Furthermore, although many studies have been conducted to traverse NAT devices while others seek to improve failure detection in P2P overlay networks; the limitations of NAT devices have received little research attention. This paper explores algorithms which allow nodes to adapt to the timeout values of individual NAT devices and investigates the resulting trade-offs.

I. Introduction

Using stateful translation tables that map multiple private addresses onto a single public address, Network address translation (NAT) devices allow several nodes within a private network to share a single public IP address. This allows private home networks and business intranets to interface with public networks such as the Internet. The widespread use of NATs has helped to alleviate the IPv4 network address shortage problem.

However due to limited resources and the vulnerability to denial of service attacks, NAT devices cannot indefinitely hold the state of their translation tables. As a result, idle connections are eventually expired and connection states removed after a NAT *timeout period*.

To avoid connections becoming idle, connected nodes must periodically exchange *keep-alive messages* at an interval shorter than the timeout period. As a result, keep-alive messages are widely used throughout all types of networks to maintain connections between nodes. The keep-alive period k defines the maximum interval that a connection between two nodes can remain inactive. Keep-alive messages are exchanged if no message has been sent within a keep-alive period to ensure the corresponding node is both still online and to avoid the connection being removed by a NAT device.

Keep-alive messages therefore serve a dual purpose, firstly keep-alive messages are used to detect the departure of ungraceful nodes. When a node that is part of the network receives a keep-alive message it responds by returning an acknowledgement message. Nodes that have left the network do not respond allowing failed connections to be detected and replaced. By proactively replacing failed connections nodes can ensure they remain well connected to a network overlay.

The second purpose of keep-alive messages is to prevent connections from becoming inactive and, as a result, being removed by NAT devices. Every time a packet is sent through a connection the NAT device at the other end restarts the timeout period. Keep-alive messages therefore serve as artificial packets, forcing NAT devices into resetting the timeout period and keeping the connection alive.

The keep-alive period is typically a fixed periodic interval uniform across all nodes in the network. The size of this keep-alive period interval is often determined by hand by application developers and is selected to fall within the timeout values of most NAT devices. The designers of BitTorrent [1] for example have set the default keep-alive period to $k = 120$ seconds. However RFC 1122 [2] recommends TCP stacks should wait for at least 2 hours between sending TCP keep-alive packets, accordingly NAT devices should not expire a TCP connection within this time. As a result a node in a BitTorrent network may be sending sixty times more keep-alive messages than is strictly necessary.

In this paper we explore and empirically analyse for the first time algorithms that can efficiently adapt keep-alive intervals to match the timeout values of NAT devices. and investigate the trade-offs of extending keep-alive intervals. Specifically the contributions of this paper are:

- We formally explain and analyse iGlance an existing algorithm that roughly estimates the timeout values of NAT devices. Accordingly we propose a more accurate algorithm based upon traditional binary search that can efficiently find the timeout value of NAT devices.
- We evaluate the proposed algorithms using real network data from the RedHat9 BitTorrent distribution. Using our trace driven simulation platform we compare the adaptive algorithms to the standard periodic approach commonly used throughout P2P networks.
- Using distinct evaluation metrics we identify the trade-offs in terms of bandwidth and the incurred failure detection delay when aligning keep-alive intervals to timeout values.
- Finally we show by augmenting all the algorithms with a simple gossip mechanism it is possible to extend keep-alive intervals, thereby reducing bandwidth spent on maintenance without timing out connections while retaining a reasonable average failure detection delay.

The rest of the paper is organised as follows. Section two provides an overview of existing work. Section three explains the well established standard keep-alive mechanism. Section four describes our binary search based approach while our experimental methodology is described in section five. Section six presents the results of the simulated experiments, comparing our algorithm with the standard keep-alive algorithm. Finally, section seven concludes the paper.

II. Related Work

While research often focuses upon making P2P network overlays flexible, efficient and robust [3], [4], the research community has identified reducing the cost of maintenance as an important open problem [5]. While a number of studies attempt to improve network failure detection; to the best of our knowledge, the problem of adapting to timeout values of NAT devices has not yet been fully addressed in this context.

The closest work to our own is iGlance [6], authored by David Barrett. iGlance an open-source Voice-Over-IP (VOIP) application allows clients to adapt the size of the keep-alive interval to an approximate NAT timeout period. Each client creates two connections with the server, one for live traffic and the other as a test connection. The live traffic connection always operates on a known safe keep-alive period k_{live}, while the test connection experiments with an alternative keep-alive interval k_{test}. If a corresponding node acknowledges a keep-alive message through the test-connection the interval k_{test} is doubled. Furthermore k_{live} is updated to reflect the new safe keep-alive period k_{test} if k_{test} is larger than the current value of k_{live}.

Should the corresponding node fail to acknowledge the keep-alive message, it is presumed to have timed out by the NAT device and the interval is halved. We implement a slight modification of this approach, by sending a keep-alive message through the live connection as well as the test connection when k_{test} period expires. Should the only the test connection fail to respond we can safely assume a NAT device has expired the connection, whereas should both connections fail it's likely the corresponding node has left the network.

By experimenting with increasingly large keep-alive intervals iGlance can approximate the NAT timeout period and reduce the number of keep-alive messages sent, whilst also allowing the live connection to operate within a safe interval. However, the simple process of doubling and halving the keep-alive interval may never find the exact timeout value of a particular NAT device. Furthermore, iGlance's adaptive algorithm is only a small part of much larger overall VOIP application. The results of adapting to the timeout values of NAT devices, as far as we know, have never been empirically tested. In this paper, we implement and evaluate the iGlance algorithm comparing it against the standard periodic approach and our own algorithm[1].

A number of other studies attempt to reduce the number of keep-alive messages needed to check corresponding nodes are still online without considering the limitations of NAT devices. One approach is not to send any keep-alive at all and reactively recover from failed connections as messages timeout. While this completely removes the cost of maintenance, failures are only detected when connections are needed. Rhea et al. in [4] explain how this approach is ineffective as message must timeout before they can be resent incurring potentially long network delays. Furthermore, they also show reactive recovery may exacerbate existing problems by adding to network congestion through a positive feedback loop.

Our previous work in [7], introduced three algorithms that determined the required frequency of keep-alive messages by using the current uptime to predict the remaining uptime of nodes. By prioritising keep-alive messages to nodes that are more likely to fail, the expected delay between failures occurring and their subsequent detection can be reduced. Our experiments show compared to the standard periodic approach our predictive algorithms can reduce the median failure detection delay by around 20%.

So and Sirer in [8] analyse the tradeoff between resource consumption and detection latency when detecting node failures. They describe two approaches that minimise the failure detection delay and bandwidth spent respectively, given that the average session time of each individual node is known. Dedinski et al [9] use cooperation between nodes with mutual neighbors to effectively reduce maintenance overhead by sending keep-alive messages in sequence rather than in parallel. Five distributed failure detectors are studied empirically in [10], showing that the average detection delay can be significantly and effectively reduced by sharing information of failures as they are detected between nodes.

III. Approach

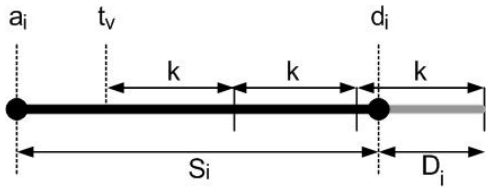
This section first describes the current state of the art approach of detecting failures used by the majority of P2P networks. We then describe our Binary search based approach that is able to accurately and efficiently find the timeout value of NAT devices.

A. Standard Keep-alive Algorithm

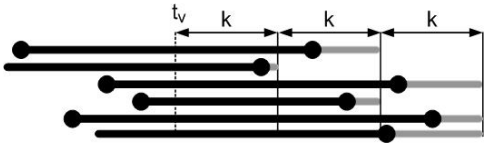
The Standard Keep-Alive (SKA) algorithm is widely used to detect the departure of ungraceful nodes and prevent connections being removed by NAT devices. A node assumes an entry in it's routing table to be online in the network for a duration of time defined by the keep-alive period k. Therefore, the time that a connection between two nodes can remain inactive is defined by the keep-alive period. If no message has been exchanged within a keep-alive period, keep-alive messages are exchanged to ensure the corresponding node is still online.

In a network with N nodes with an average degree D there are $(N \cdot D) \cdot 2$ keep-alive messages sent and subsequently acknowledged every k seconds. Keep-alive messages are also small, around 40 bytes which is equivalent to header of a IP packet containing a TCP segment of size 0. If a node leaves the network it does not respond to keep-alive messages, these unacknowledged messages are typically re-sent multiple times at short intervals to minimise the risk of false positives where keep-alive messages have been somehow lost. As ungraceful

[1]We would like to thank David Barrett for providing us extensive details of the iGlance adaptive algorithm.



(a) A node's session time $S_i = d_i = a_i$, where D_i is the failure detection delay period



(b) The mean and median failure detection delay period is $k/2$

Fig. 1. The standard keep-alive algorithm

nodes do not inform incoming connections upon leaving a network, these neighbours are left ignorant to changes in network topology.

Figure 1a as shown in [7] illustrates node i arriving in the network at time a_i and departing the network at time d_i. A node's session time S_i, the amount of time node i spends in the network is given by $S_i = d_i - a_i$. At time t_v node v connects to node i and begins sending periodic keep-alive messages with an interval of k seconds. At time d_i node i departs the network ungracefully, node v does not learn of this departure until the subsequent keep-alive that is sent but goes unacknowledged. As a result there is period D_i during which node v falsely believes that node i is still present within the network. We call this period the *failure detection delay*.

The size of the failure detection delay period is directly proportional to the keep-alive period k. As node's can connect to one another at at any point during a nodes session time with equal probability, departure time d_i of neighbours falls uniformly within any single keep-alive interval. Therefore the mean and median failure detection delay between a node leaving a network and subsequently being detected is $k/2$ using a periodic keep-alive mechanism as illustrated in Figure 1b.

The larger the failure detection delay the longer a node incorrectly considers a entry in it's routing table to be present within the network. While application developers may want to reduce the failure detection delay as much as possible, using the SKA approach this can only be done at the cost of sending more keep-alive messages. While other techniques [7], [8], [9], [10] improve upon the performance of the SKA approach the fundamental trade-off remains the same; extending the keep-alive period increases the incurred failure detection delay while reducing the cost in terms of bandwidth.

B. Binary Search

To improve on the iGlance algorithm that doubles and halves the test-interval we implement a Binary Search algorithm. As in iGlance the Binary search approach creates a live and a test connection, and updates the k_{live} and k_{test} intervals through positive and negative feedback. The binary search approach locates the NAT timeout period t by finding upper and lower bounds of t and selecting the middle value of these two points to progressively divide the search space in half.

In our context, the test connection sets a keep-alive period of k_{test} seconds. Each time the corresponding node successfully responds through the test connection, we learn the NAT timeout period t is greater than k_{test}. Accordingly we set k_{live} and min equal to current value of k_{test}. To begin with there currently is no upper bound on t, so the interval k_{test} is doubled. When the k_{test} interval exceeds t the NAT device removes the connection and the keep-alive message eventually fails, at this point we know t is lower than the current value of k_{test} so the upper bound max is set to k_{test}.

At this point the NAT timeout interval t must between lower and upper bounds min and max respectively, the binary search approach is to set k_{test} to the the middle of these two points $\frac{min+max}{2}$. If the keep-alive message is acknowledged after k_{test} seconds min and k_{live} are updated, otherwise the test connection has timed out and upper bound max is updated. Finally k_{test} is set to $\frac{min+max}{2}$ and the process is repeated. This implementation of Binary search should find a NAT timeout period t in at most $2[log_2 t]$ steps.

C. Gossiping Failures

To further reduce the failure detection delay we augmented all algorithms with a simple gossip mechanism. This mechanism shares information regarding node failures with their mutual neighbours.

Each time that node *X* sends a probe to node *Y* it also learns of all Y's neighbours $n(Y)$. If node Y subsequently does not return node X's keep-alive messages, X then informs the other neighbours $n(Y)$ of this failure. These mutual neighbours then immediately probe Y themselves to ensure news of the failure is correct. Although $n(Y)$ may be outdated, nodes that are not informed of Y's failure will eventually detect it themselves. Further examples of alternative and more complex sharing-based, gossip-based and flooding-based mechanisms are evaluated in [9], [10].

While gossip-based and similar mechanisms reduce the failure detection delay this comes at the cost of increased control overhead. Additional messages are required to inform mutual neighbours, who themselves check a node has failed. However as previous studies have shown this additional overhead is relatively small as the majority of maintenance consists of keep-alive messages with gossip messages only being sent when a failure is detected. The next section details our experimental methodology which we use to analyse and compare the algorithms described above.

IV. Experimental Methodology

To evaluate and compare the algorithms described above our simulations are based on a publicly available [11] BitTorrent tracker log. Our simulations are trace-driven based on a portion

of the five month logged period of the RedHat9 BitTorrent network.

New nodes join a BitTorrent network by connecting to the random subset of nodes already present in the network provided by a BitTorrent tracker. Periodically nodes update their status and, if they leave gracefully, inform the tracker upon departing the network. By automatically logging all this data, BitTorrent trackers provide us with the arrival and departure times of peers to the nearest second. Enabling us to model the complex process of churn accurately and realistically.

In some respects compared to crawler based techniques that actively probe a subset of nodes within a network at regular intervals, BitTorrent tracker logs are a more accurate source of network data. As crawlers progressively probe the network; increasing the number of nodes a crawler incorporates also increases the interval between successive probes, this interval currently ranges between four and thirty minutes [12], [13], [14]. Crawler based techniques cannot accurately capture session times smaller than the granularity of the network trace. While BitTorrent trackers passively monitor the network, registering to the nearest second nodes that contact it upon joining, periodically and upon departure. Of course the nodes that do not contact the tracker cannot be logged in a passive fashion.

The arrival and departure time of any graceful node can be accurately determined by processing a tracker log. Although the departure time of ungraceful nodes are not listed within tracker logs, as all nodes periodically update their progress at thirty minute intervals we can assume they leave at most thirty minutes after their last update. Accordingly we can add a uniformly random time of at most thirty minutes to the last progress update of ungraceful nodes to determine their simulated departure time. Furthermore, all our experiments utilise a fail-stop model in which all nodes graceful and ungraceful do not inform their neighbours upon departing the network.

In this paper we simulate an unstructured network, trace-driven upon the RedHat9 BitTorrent tracker log data. Whilst online each simulated node creates and maintains a fixed number of connections *D* with other existing nodes selected at random, we experiment with a range of node degree values setting $D = 20, 30$ and 40. As we only simulate maintenance messages we believe this work is general enough to be applied to any type of P2P network overlay regardless of it's structure.

In addition we also simulation the expiration of connections via NAT device timeouts. Each node x has a individual NAT timeout period t, if any connection to x is idle for longer than t seconds that connection is expired. In order to accurately simulated the NAT timeout values we generate our simulated timeout values according to [15] with a minimum imposed timeout value of two minutes. Guha and Francis in [15], find that only 35.6% of NAT devices keep an idle connection open for at least two hours. Furthermore, 21.8% of NATs expire idle connections after less than fifteen minutes, with the remaining 42.6% of NATs having a timeout period in somewhere between two hours and fifteen minutes. While connections in a unstructured network are unidirectional, data may flow both ways as a result we use expire a connection according to the lower of the two timeout values of two connected nodes.

For each experiment the simulation begins cold, i.e without any peers. The first twelve hours of the network then act as a warm-up period as nodes populate and leave the network according the events given by the trace. Once the warm-up period is finished each node then creates D connections with existing nodes and we report the maintenance of these connections over the subsequent twelve simulated hours. Each experiment is repeated five times with the results averaged over these runs and standard deviation shown in the next section.

V. Results

We evaluate each mechanism based upon two main criteria:

- **Cost:** The average bandwidth consumed per node per second online. Formally the cost C is equal to $\frac{(s+a)\cdot p}{T}$; where s and a are the number of keep-alive messages sent and acknowledged respectively, p is the size of a keep-alive message and T the sum of all node session times.
- **Failure detection delay:** The mean and median time that elapses between a failure occurring and subsequently being detected.

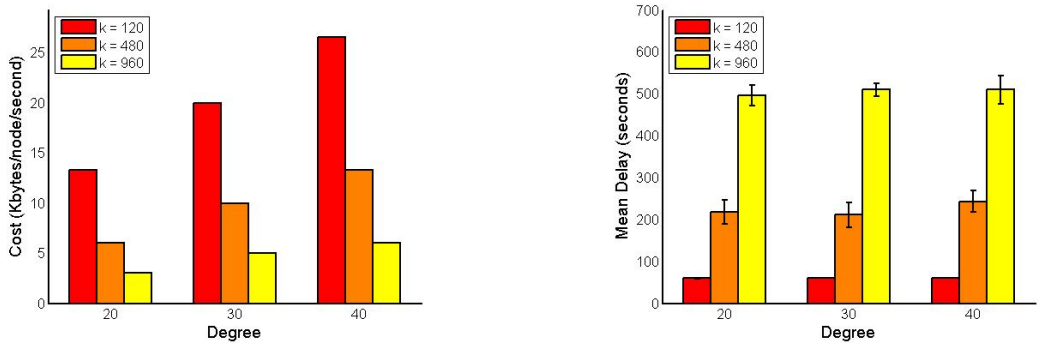
Cost and mean failure detection delay have been used as performance metrics other evaluations of failure detection algorithms including [7], [8], [10].

In this paper we compare the standard keep-alive algorithm (SKA), iGlance algorithm, Binary search algorithm against one another using the above metrics. Furthermore we also implement an imaginary 'optimal' algorithm called the Omni approach that has global knowledge of the timeout interval of NAT devices and matches the keep-alive period to that value. As we assume NAT devices do not know their own timeout period, clearly the global knowledge possessed by the Omni algorithm would be impossible to obtain. The purpose of the Omni algorithm is to establish the baseline minimum amount of traffic possible by extending the keep-alive period as far as possible without expiring the connection.

Figure 2, shows Binary search approach reduces the cost of maintenance to approximately the same level as the standard periodic algorithm with a keep-alive interval of $k = 960$. Despite the additional overhead incurred by using two connections per neighbour, the average number of keep-alive messages sent and received is around five times lower than the default parameter of BitTorrent [1]. By adapting to NAT timeout values the live connection is able to use increasingly large and safe keep-alive intervals. As the live connection always uses the largest known safe keep-alive interval there is no risk of a NAT device expiring the connection before it is needed.

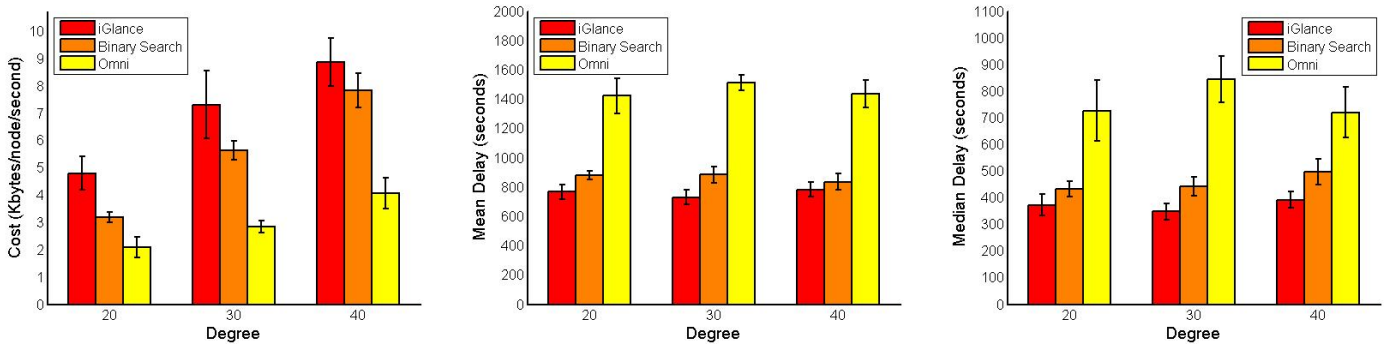
Figure 3a shows the iGlance algorithm by only doubling and halving the k_{test} interval it cannot accurately approximate many NAT timeout values and cannot achieve the lower cost values of the more accurate Binary Search approach.

Figure 3b and 3c shows there is a inherent tradeoff, by extending the keep-alive intervals we unavoidably increase



(a) (b)

Fig. 2. The performance of the standard keep-alive (SKA) algorithm in terms of cost and mean failure detection delay.



(a) (b) (c)

Fig. 3. Comparison of the iGlance, Binary Search and Omni approaches in terms of cost, mean and median failure detection delay.

the average failure detection delay. The adaptive iGlance and Binary search approaches only consideration is to find the NAT timeout values of nodes present in the network, as these values may be as large as two hours the resulting failure detection delay is also large. Periodic keep-alive algorithms generally set the keep-alive interval to be lower than the majority of NAT device's timeout values as a consequently the average failure detection delay is also very low. By finding these timeout values the iGlance, Binary Search and Omni algorithms incur large failure detection delays when nodes eventually leave the network. Figure 3 shows that the Omni approach is only optimal in terms of minimising bandwidth. By matching the timeout values immediately the Omni approach incurs the largest failure detection delays as node must wait until the end of the next keep-alive period to detect a failure.

While the mean and median detection delay are equivalent for the periodic approach, the the median delay is significantly lower for the adaptive algorithms than mean failure detection delay incurred. This is due to some timeout values being low resulting in a short keep-alive interval being found and nodes failing before the while the live connection is using relatively small keep-alive intervals. While application designers wanting to create low overhead networks may find these delays acceptable, designers of high churn networks should be wary of incurring such high failure detection delays as undetected failed neighbours could potentially occupy routing table entries for long periods of time.

However, it should be highlighted that our experiments assume a fail-stop model, meaning all nodes leave the network ungracefully. In real networks the majority of nodes will leave gracefully, informing their neighbours upon departure and incurring no failure detection delay as a result. Our experiments therefore simulate the worst-case scenario maximising the incurred failure detection delay.

To rectify the unreasonably large failure detection delays incurred, further experiments shown in Figure 4, show a simple gossip mechanism as discussed in [9], [10] can be used to significantly reduce the mean and median failure detection delay of all the algorithms with little additional bandwidth cost. As the degree of node is increased the more mutual neighbours nodes share further reducing the incurred failure detection delay. However gossip mechanisms are not designed to prevent connections expiring, only to inform mutual neighbours of the departure of neighboring nodes. As a result, the adaptive algorithms with gossip prevent connections becoming idle and keep the mean failure detection delay acceptably low. Of course the standard periodic approach can also be augmented with gossip further reducing the average failure detection delay, as shown in [10], but cannot ensure connections do not expire.

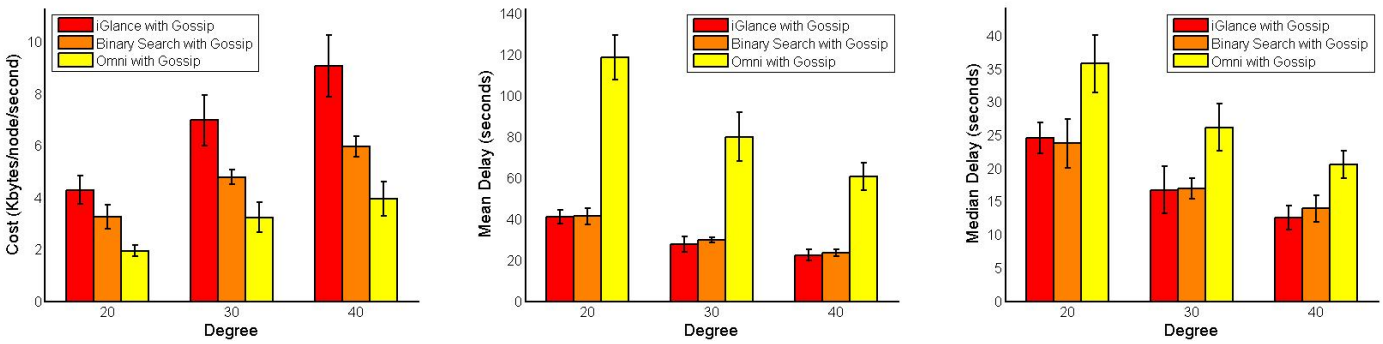


Fig. 4. Comparison of the iGlance, Binary Search and Omni approaches with Gossip in terms of cost, mean and median failure detection delay.

VI. Future Work

In the future, we would like to implement the algorithms proposed in this paper to a real system. While the simulated results of adaptive approaches appear promising how they behave in real network may highlight further areas of interest.

Furthermore, we would like to augment the binary search algorithm by adding collaborative and randomized features. As many nodes may be connected to a single node x, the mutual neighbours of x can work together to find x's NAT timeout value. One possible drawback of a collaborative feature is neighbours could experiment with the same k_{test} values in parallel, to avoid this we could also implement a version of the collaborative algorithm that randomises the k_{test} interval uniformly within the range min and max.

We would also like to extend our previous work presented in [7] to improve failure detection within structured networks. As structured networks must maintain the topology of the overlay they present a different constraints to the unstructured networks examined previously. Furthermore we would like to investigate more robust ways of determining the current uptime of a node.

VII. Conclusion

This paper presented a simple yet novel algorithm based upon Binary Search that adapts itself to the timeout period of NAT devices to ensure the connection for live traffic does not become idle. When compared to the existing iGlance algorithm and the widely deployed standard periodic approach the Binary Search approach reduces the cost of maintenance incurred by nodes significantly without live connections expiring. In doing so, the failure detection delay is significantly increased as keep-alive intervals match the potentially large timeout values. However by augmenting these algorithms with a simple gossip gossip mechanism we showed that the failure detection delay can be reduced to more acceptable levels.

Overall, this paper has shown it is possible to improve performance, in terms of bandwidth spent on maintenance, by adapting to the timeout value of NAT devices. However tuning keep-alive intervals presents an inherent trade-off between the bandwidth spent and the incurred failure detection delay. All our of the adaptive algorithms presented here reduce the cost of maintenance at the expense of increased failure detection delays. However, by adapting to NAT timeout values these algorithms ensure keep-alive messages are only sent until strictly necessary and no live connections are expired due to idleness. In conclusion this paper has shown that adaptive mechanisms and out-of-bound connections can be successfully used to reduce the cost of keep-alive messages and there is significant potential for future work.

References

[1] I. BitTorrent, "Bittorrent," World Wide Web, http://www.bittorrent.com/.

[2] R. Braden, "RFC 1122: Requirements for Internet Hosts Communication Layers," OCT 1989.

[3] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable Peer-To-Peer lookup service for internet applications," in *Proceedings of the 2001 ACM SIGCOMM Conference*, 2001, pp. 149–160.

[4] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling churn in a dht," in *ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2004, pp. 10–10.

[5] S. El-Ansary and S. Haridi, "An overview of structured overlay networks," in *Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless and Peer-to-Peer Networks*, 2005.

[6] D. Barrett, "iGlance," in *Presented at CodeCon*, 2006.

[7] R. Price and P. Tino, "Still Alive: Extending Keep-Alive Intervals in P2P Overlay Networks," in *Fifth International Conference on Collaborative Computing*, 2009.

[8] K. C. W. So and E. G. Sirer, "Latency and bandwidth-minimizing failure detectors," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, pp. 89–99, 2007.

[9] I. Dedinski, A. Hofmann, and B. Sick, "Cooperative keep-alives: An efficient outage detection algorithm for p2p overlay networks," in *P2P '07: Proceedings of the Seventh IEEE International Conference on Peer-to-Peer Computing*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 140–150.

[10] S. Zhuang, D. Geels, I. Stoica, and R. Katz, "On failure detection algorithms in overlay networks," in *Proceedings IEEE INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, 2005.

[11] M. Izal, "Bittorrent traces and tools," World Wide Web, March 2009, http://mikel.tlm.unavarra.es/ mikel/bt_pam2004/.

[12] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *IMC '06: Proceedings of the 6th ACM SIGCOMM on Internet measurement*. New York, NY, USA: ACM Press, 2006, pp. 189–202.

[13] R. Mahajan, M. Castro, and A. Rowstron, "Controlling the cost of reliability in peer-to-peer overlays," in *In IPTPS*, 2003.

[14] S. Saroiu, K. P. Gummadi, and S. D. Gribble, "Measuring and analyzing the characteristics of napster and gnutella hosts," *Multimedia Syst.*, vol. 9, no. 2, pp. 170–184, 2003.

[15] S. Guha and P. Francis, "Characterization and measurement of TCP traversal through NATs and firewalls," in *Proceedings of the Internet Measurement Conference (Berkeley, CA), October*, 2005, pp. 49–681.