Using Adaptive Heartbeat rate on Long-lived TCP Connections

M Saifur Rahman, Md. Yusuf Sarwar Uddin, M Sohel Rahman and M Kaykobad
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology
Email: {mrahman, yusufsarwar, msrahman, kaykobad}@cse.buet.ac.bd, saifur80@gmail.com

***Abstract*—In this paper, we propose techniques for dynamically adjusting heartbeat or keep-alive interval of long-lived TCP connections, particularly the ones that are used in push notification service in mobile platforms. When a TCP connection between a client behind a NAT (or any other middle-box) and a server is idle for a long time, it may get torn down due to TCP binding timeout. In order to keep the connection alive, the client device needs to send keep-alive packets through the connection when it is otherwise idle. To reduce resource consumption, the keep-alive packet should preferably be sent at the farthest possible time within the NAT binding timeout. We refer to this interval as the *Optimal Keep-alive Interval*. Due to varied settings of different network equipments, optimal keep-alive interval will not be identical in different networks. Hence, the heartbeat rate used in different networks should be changed dynamically. We propose a set of iterative probing techniques, namely binary, exponential and composite search, that detect optimal keep-alive interval with varying degree of accuracy; and in the process, keeps improving the keep-alive interval used by the client device. We also analytically derive performance bounds of these techniques. To the best of our knowledge, ours is the first work that systematically studies several techniques to dynamically improve keep-alive interval. To this end, we run experiments in simulation as well as make a real implementation on Android to demonstrate the proof-of-concept of the proposed schemes.**

I. Introduction

Smart phones, tablet PCs and other customized PDAs try to provide the user with fresh data. This includes the user's emails, social news feed etc. Real time communications, such as voice and video calls, are also performed through such devices. Since the devices in question have limited battery life, they cannot frequently poll for information updates, incoming call notifications etc. Rather, they rely on change notifications being pushed by a notification server. Mobile platforms like iPhone, Android, Windows Phone, Black Berry etc. provide a *Notification Service* which, at a high level, can be modeled as shown in Figure 1. Rather than polling different services to check if data needs to be downloaded, the user's device only maintains a TCP connection to a notification server. This connection is called a *Notification Channel*. When the user's social network service wants to send recent activity feeds, it sends a new activity notification to the notification server. This is delivered to the device through the notification channel. Based on this notification, the device opens a new connection to the social network service, downloads the activity feed and closes the connection. Similar workflow is executed for downloading unread emails, receiving voice and video calls etc. Therefore, to enable these seemingly *always connected* scenarios, the device must keep the notification channel alive at all times, even during low power modes. Such a connection is quiet for the most part, with traffic flowing only when the server has an update to share. When the device is behind a Network Address Translator (NAT), firewall or any other stateful middle-box, the connection would be subjected to binding timeouts. That means, the connection state may be cleared by the middle-box if no data passes through it for some specified amount of time.

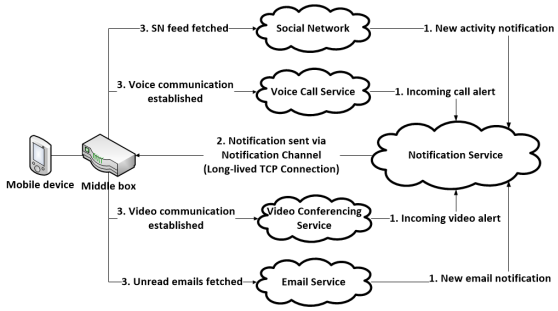


Fig. 1. A model of notification service used in different mobile platforms.

Studies have shown that the NAT binding timeouts vary dramatically amongst the commercially available home gateways [1]. To safeguard against this timeout, the device needs to periodically probe the other end of the connection when the connection is otherwise idle. This is called a keep-alive [2] or a heartbeat; with the interval being referred to as keep-alive interval or KA interval in short. To balance the battery life constraint with the necessity of keeping the connection alive, it is preferred that the keep-alive probe is made at the farthest possible time within the NAT binding timeout. This interval is termed as optimal keep-alive interval. Such an interval needs to be computed via experiment in the network environment that the device is currently in.

In this paper, we present several iterative probing approaches to dynamically improve the KA interval of a long-lived TCP connection. These include binary search, exponential search and composite search. Composite search combines

978-1-5090-0203-0/16/$31.00

different aspects of binary and exponential search techniques. We analyze the proposed techniques theoretically and validate them through simulation. We show that composite search technique results in the least number of KA messages sent during several study durations of different lengths.

The rest of the paper is organized as follows. Section II reviews some earlier work that is relevant to our research. Section III describes several techniques to dynamically change the keep-alive interval of TCP connection. Section IV provides analytical bounds on several properties of these techniques. Section V describes the simulation experiments and their results. Finally, Section VI concludes this paper.

II. Related Work

We organized the literature review in four sections. Firstly, we look at a study that suggests binding timeout of TCP connections varies widely amongst different networking equipments. This motivates the need for sending keep-alive packets periodically to retain the connection. Then we review use of KA packets in existing systems. We also review some literature on impact of KA interval on power consumption. This motivates the need for detecting and using longer KA intervals when possible. Finally, we review some earlier works that uses iterative probing for measuring different network parameters.

A. Study of NAT Binding Timeouts

In [1], Hätönen et al. experimentally analyzed the TCP binding timeouts of different home gateways. The shortest timeout observed was about 4 minutes; median was about an hour. While the Internet Engineering Task Force (IETF) recommends a timeout of 124 minutes [3], more than 50% of the devices did not comply.

On the other hand, some of the devices retained TCP bindings for considerably longer; they did not timeout the TCP connection even after 24 hours of idleness. As the binding timeout varies widely, a mobile device in real network environment may also need to perform testing to find longer keep-alive interval and send keep-alives using that period. While protocols such as NSIS [4] or SIMCO [5] do exist to explicitly negotiate the binding timeout with the middle-box, equipments currently used by mobile operators do not usually support these protocols.

B. Keep-alive in Existing Systems

The *Direct Push* feature of Exchange ActiveSync (EAS) protocol uses long-standing HTTPS requests to maintain a channel with the service [6]. Each request is 'parked' at the server for a time specified by the request, if there is no update on the server to share. After the time elapses, the server returns a '200 OK'. If, however, the underlying network has a smaller binding time out, then no response is received from the server. Then the client sends another HTTPS request with a lesser amount of wait time specified. Detailed discussion on the keep-alive interval adjustment, along with the configurable parameters, can be found in [7].

Android, iPhone, Windows Phone etc. platforms maintain a persistent connection between the client (device) and a notification service [8], [9], [10], [11]. It is expected that each of these eco-systems employ exchange of keep-alive messages periodically and has mechanism to tune the keep-alive interval to obtain good battery life performance. Patents granted to Microsoft Corporation indicate use of a test connection to dynamically detect an efficient keep-alive interval for communication between client and server via middle-box [12], [13]. However, the exact strategy used in choosing the test intervals is not called out in the patents.

The Heartbeat Extension [14] of Transport Layer Security (TLS) protocol [15] also uses keep-alive messages for ensuring liveness of peers.

C. Impact of KA Interval on Power Consumption

Haverinen et al. showed in [16] that battery life is significantly influenced by the frequency of keep-alive messages. They performed real power measurements in two different 3G networks, as well as, in 2G GPRS network.

A comprehensive survey on general solutions for energy efficiency on mobile devices, published between 1999 and May 2011, is available in [17]. Here the authors classify and provide a short summary of the various efforts on studying, modeling and reducing energy consumption in mobile devices. We discuss one of the power models, due to Balasubramanian et al. [18] here. In this energy model, the energy spent to download or upload x kilobytes of data over the cellular network consists of three components: ramp energy, transmission energy and tail energy. $R(x)$ denotes the sum of the ramp energy and the transfer energy to send x bytes of data. Tail energy is represented by E per second. For WiFi, there is no ramp energy. In this case, $R(x)$ denotes the sum of the transfer energy and the energy for scanning and association. Tail energy is 0 in this case. The total energy to transmit a packet further depends on the time the interface is on. The energy consumption to keep the interface on is represented using M, the maintenance energy per second. Finally, T denotes the tail-time. Since the keep-alives are sent with a period that is longer than the tail-time, each keep-alive incurs the overhead of the tail-time, if the device is not transferring any other data at that time. In that case, by reducing the number of keep-alives, we can reduce the overall power consumption.

D. Iterative Probing to Measure Network Parameters

Iterative probing has been widely used in the literature for measuring different network parameters like end-to-end available bandwidth (avail-bw), its variability, TCP congestion window size etc. The congestion window size in TCP doubles up in each round trip iteration in the slow start mode. In the subsequent additive increase mode, it is increased by one segment in every round trip iteration. Whenever a packet loss is detected, the slow start threshold is set to be half of the congestion window size and the entire process is restarted [19].

Jain et al. used iterative probing to measure end-to-end avail-bw. Their measurement methodology, Self-Loading

Periodic Streams (SLoPS), was implemented in a tool called pathload [20]. Other iterative techniques for avail-bw measurements include Bfind [21], PTR [22], TOPP [23], pathChirp [24] etc. The variability in the available bandwidth has also been measured by Jain et al. using iterative probing [25].

III. Techniques to Dynamically Improve Keep-alive Interval

In this section, we describe our proposed techniques for improving keep-alive interval of TCP connections. All schemes use an iterative probing that repeatedly send keep-alive packets at progressively longer intervals, until a bound is found beyond which the connection can't be kept alive. This bound is referred to as the optimal keep-alive interval. The schemes vary on how these test intervals are chosen one after another.

Figure 2 shows a flow chart of the steps taken in detecting the optimal keep-alive interval. The first step is to open a separate TCP connection with the target service. If the probes were done in the data connection (e.g. notification channel), the connection may suffer disruption, which may cause adverse effect on the ongoing service. Instead, we conduct the probing on a separate connection. We refer to this connection as *test* connection.

As the testing reveals improved intervals, the new intervals can be applied on the notification channel right away. Initially, however, a conservative keep-alive interval is used. This is the maximum keep-alive interval that is already known to work. The same interval is also used as the lower bound of the search range for searching a better interval. We also specify a higher bound on the search space. The higher bound is 1 minute less than the minimum keep-alive interval that is already known to not work. Note that we use intervals in minute boundaries only. In the figure, the search range has been represented as the interval $[low, high]$.

Starting from the lower bound, we try to improve by guessing new keep-alive intervals. Once a guess is made, the connection is kept silent for that much time. Afterwards a keep-alive is sent to check whether the connection is alive or not. If the connection is alive, it means our guessed KA interval is able to keep the connection alive. Now, a higher KA interval is guessed and tested. On the other hand, if the connection was dropped, then we need to lower the guess and perform the test again. This process is continued until the difference between 2 consecutive guesses is less than 1 minute.

We explore three different techniques for optimal KA interval detection. These techniques vary in how they select the next KA interval to test. The techniques are: binary search, exponential search and composite search. Intuitively, we first applied binary search to this problem. This takes the least amount of time to find the optimal KA interval. The problem with this approach, however, is that the first probe is made after a long period of waiting during which time no improvements can be made to the KA interval in the data connection. As such, we subsequently examined exponential search, where probes are made with shorter wait times initially; Improvements could immediately be made to KA interval in the data connection. However, exponential search takes very long time to detect the optimal KA interval. Therefore, we finally combined the two approaches into what we call the composite search and were able to get good results.

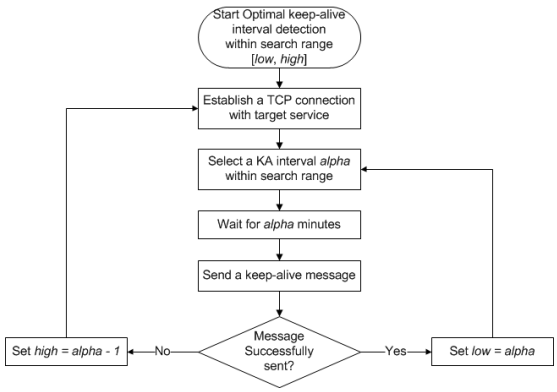


Fig. 2. Flow chart of KA interval detection technique.

For each of these techniques, we initialize low with 1 minute. This is a reasonable choice, since it is smaller than the smallest keep-alive interval in a home gateway, as found from the data presented in [1]. Exponential and composite search techniques do not need to specify a higher bound on the search range. In other words, we initialize $high = \infty$. For binary search, we initialize $high$ with 128 minutes. This is a power of 2 and is slightly higher than the IETF recommendation.

A. Binary Search

At each iteration, the mid-point of the search space is chosen as the next KA interval to be tested. After the test is completed, the search space gets halved. If the test was successful, then we search the second half of the initial search space to see if a better interval can be obtained. If, on the other hand, the test failed, then we continue searching in the first half of the initial search space. And this process is repeated. As an example, if the binding timeout of a NAT (or another middle-box) is 24 minutes, and the search range is $[1, 128]$, the sequence of intervals tested is:

$$ProbeSeq_{24}^{binary} = \{65^*, 33^*, 17, 25^*, 21, 23, 24\}$$

The probes that failed are annotated with '*' mark.

B. Exponential Search

In this approach, the difference between successive tested intervals follows geometric progression with a ratio of 2. So, the first few test intervals are $2, 4, 8, 16, 32$ minutes and so on. If the next interval to be tested goes beyond $high$, then there is no need to test that interval. Instead, low is increased to the last successfully tested interval; and the progression of the interval differences is restarted at 1. On the other hand, if the next tested interval overshoots the binding timeout, then the connection is lost. In this case, in addition to the updates

mentioned above, the $high$ is reduced to 1 minute less than the failed test interval. This process is repeated until the optimal KA interval is reached. For example, if the binding timeout is 24 minutes, the sequence of intervals tested would be

$$ProbeSeq_{24}^{exp} = \{2, 4, 8, 16, 32^*, 17, 19, 23, 31^*, 24, 26^*, 25^*\}$$

C. Composite Search

Composite search is a combination of binary and exponential searches. Initially, it acts like the exponential search. The difference between successive tested intervals follows geometric progression with ratio 2. When a tested interval overshoots the actual binding timeout, the test connection will get terminated. At that point, low is increased to the last successfully tested interval; and $high$ is reduced to 1 minute less than the failed test interval. Subsequently, binary search is performed in the new search range. As an example, if the network timeout is 24 minutes, the sequence of intervals tested would be:

$$ProbeSeq_{24}^{comp} = \{2, 4, 8, 16, 32^*, 24, 28^*, 26^*, 25^*\}$$

IV. Analytical Bounds of KA Schemes

We have analytically derived two metrics, namely the number of probes performed in the test connection to detect the optimal KA interval (Probe Count), and total time taken to do so (Convergence Time). Our schemes can be compared based on these.

Let α denote the optimal keep-alive interval. Let the search range be $[1, h]$. For binary search, $h = 2^k$ for some $k \geq 1$. For exponential and composite search, $h = \infty$. Let $N(\alpha)$ denote the number of test probes needed to detect the optimal KA interval. The best, worst and average case values of $N(\alpha)$ are represented respectively by N_{best}, N_{worst} and N_{avg}. Let $T(\alpha)$ denote the time it takes to detect the optimal KA interval α. This is called the convergence time. The best, worst and average case convergence time for the specified search range is represented by T_{best}, T_{worst} and T_{avg} respectively. In what follows, we obtain analytical bounds on these properties for the different detection techniques. Due to space limitation, we only show the derivations for binary search. For the other techniques we only record the results here. The reader is referred to [26] for details.

A. Probe Count

In binary search,

$$N_{best}^{binary} = N_{avg}^{binary} = N_{worst}^{binary} = \lg h \tag{1}$$

In exponential search, we have:

$$N_{best}^{exp} = 1 \tag{2}$$

$$N_{worst}^{exp} = \begin{cases} \lg h + 2 & \text{when } h \leq 4 \\ \frac{\lg h(\lg h+1)}{2} & \text{otherwise.} \end{cases} \tag{3}$$

$$N_{avg}^{exp} = \frac{\lg h(1 + \lg h)}{4} + 1 + \frac{1}{h} \tag{4}$$

In composite search, we have:

$$N_{best}^{comp} = 1 \tag{5}$$

$$N_{worst}^{comp} = 1 + 2\lg h \tag{6}$$

$$N_{avg}^{comp} = 2\lg h + \frac{2}{h}(2 + \lg h) - 3 \tag{7}$$

Probe count in different search techniques has been compared in Table I.

TABLE I
PROBE COUNT COMPARISON AMONG TECHNIQUES TO DYNAMICALLY IMPROVE KA INTERVAL.

	Best	**Average**	**Worst**
Binary search	$O(\lg h)$	$O(\lg h)$	$O(\lg h)$
Exponential search	1	$O(\lg^2 h)$	$O(\lg^2 h)$
Composite search	1	$O(\lg h)$	$O(\lg h)$

B. Convergence Time

In binary search, the first probe takes $1 + 2^{k-1}$ unit time. if $\alpha \geq 1 + 2^{k-1}$, then the second probe takes $1 + 2^{k-1} + 2^{k-2}$ unit time after completion of first probe. On the other hand, if $\alpha \leq 2^{k-1}$, then the second probe takes only $1 + 2^{k-2}$ unit time. Let $a_{k-1}a_{k-2}...a_1a_0$ denote the binary representation of $\alpha - 1$. Therefore, we can write the following expression:

$$T^{binary}(\alpha) = (h + \lg h - 1) + \sum_{i=1}^{\lg h - 1} a_i i 2^i \tag{8}$$

The best case convergence time happens when α is 1 or 2. And that value is: $2^k + k - 1$ unit time. Therefore,

$$T_{best}^{binary} = h + \lg h - 1 \tag{9}$$

The worst case convergence time occurs for $\alpha = 2^k$ or $\alpha = 2^k - 1$.

$$\begin{aligned} T_{worst}^{binary} &= (2^k + k - 1) + \sum_{i=1}^{k-1} i 2^i \\ &= h \lg h + \lg h - h + 1 \end{aligned} \tag{10}$$

The convergence time on average is:

$$\begin{aligned} T_{avg}^{binary} &= \frac{1}{2^k} \sum_{\alpha=1}^{2^k} T(\alpha) \\ &= (\frac{h}{2} + 1) \lg h \end{aligned} \tag{11}$$

In exponential search, we have:

$$T_{best}^{exp} = 2 \tag{12}$$

$$T_{worst}^{exp} = \begin{cases} 5h - 1 & \text{when } h \leq 8 \\ \frac{\lg^2 h - 3lgh + 12}{2} h - \frac{\lg^3 h + 11 \lg h + 36}{6} & \text{otherwise.} \end{cases} \tag{13}$$

$$T_{avg}^{exp} = \frac{\lg^2 h - 3\lg h + 24}{8} h - \frac{\lg^3 h + 23\lg h - 24}{24} \quad (14)$$

In composite search, we have:

$$T_{best}^{comp} = 2 \quad (15)$$

$$T_{worst}^{comp} = h\lg h + 5h - 3 \quad (16)$$

$$T_{avg}^{comp} = (\frac{h}{2} + 1)\lg h + \frac{2}{3}h + 3 - \frac{5}{3h} \quad (17)$$

TABLE II
CONVERGENCE TIME COMPARISON AMONG TECHNIQUES TO DYNAMICALLY IMPROVE KA INTERVAL.

	Best	**Average**	**Worst**
Binary search	$O(h)$	$O(h\lg h)$	$O(h\lg h)$
Exponential search	2	$O(h\lg^2 h)$	$O(h\lg^2 h)$
Composite search	2	$O(h\lg h)$	$O(h\lg h)$

Table II show the convergence time comparison of the search approaches.

V. EXPERIMENTS

We have implemented the different techniques to detect the optimal keep-alive interval on Omnet++ simulation platform [27]. We made an experimental setup with a client connected to a server through a single middle-box. Although in practice a connection may pass through a series of NAT boxes and firewalls with different timeout bindings, in terms of connection timeout the smallest interval along the path applies. In that, the series of middle-boxes can effectively be replaced by that particular middle-box with smallest binding timeout. The topology of the simulation setup is shown in Figure 3.

The delay from a node to the middle-box is set to 10ms. Between middle-box and the server, the delay was set to 100ms. These choices have been based on the observed round trip time (RTT) to the gateway and to prominent cloud services respectively, when connected to the network of the Department of CSE, BUET through different access points. As long as these delays are much less than 1 minute, our results will continue to hold.

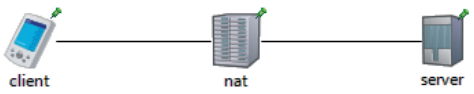


Fig. 3. Simulation topology.

A. Correctness of Analytical Bounds

Figure 4a plots the probe count of binary search, as found through simulation, against binding timeout. The theoretical curve is also put in the same plot. As can be seen, they are identical. This is expected, since the probe count in a given search range is independent of any specific binding timeout.

Figure 4b and 4c respectively plot the probe count of exponential and composite search techniques as a function of binding timeout. In both plots, observe that the experimental curve is off from the theoretical curve by 1 minute. That is, the observed probe count for any α is equal to the theoretical probe count for $\alpha-1$. This is due to network delay τ. While the middle-box times out a binding after α unit of time, if a node sends keep-alive after α unit time of silence, the KA packet reaches the middle-box after $\alpha + \tau$ unit time of quietness, where $0 < \tau \ll \alpha$. Therefore, the connection gets dropped. So, from the node's perspective, the binding timeout is $\alpha - 1$. For the same reason, the experimental convergence time curves (not shown for brevity) are also off from their theoretical counterparts by 1 unit along the Y-axis.

B. Keep-alive Interval with Tunable Accuracy

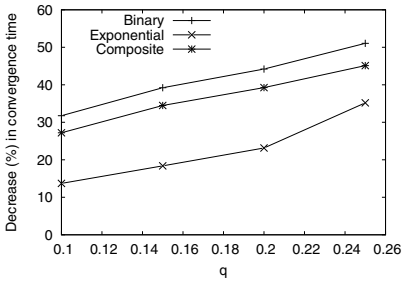
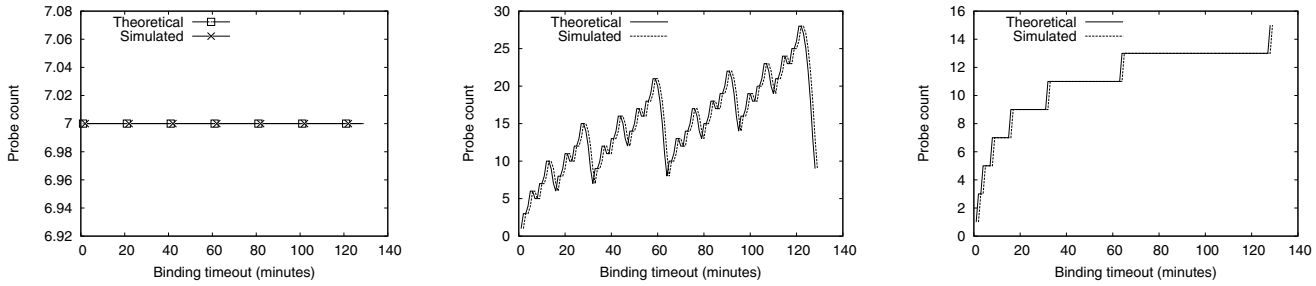


Fig. 5. Average decrease in convergence time due to relaxing accuracy of detected KA interval.

Long convergence time of the iterative probing techniques may result in increased load on the server. To mitigate that, we can sacrifice accuracy of the detected KA interval to reduce the convergence time. We introduce a tunable parameter q in our algorithms for this purpose. Let α be the optimal keep-alive interval. In effort to reduce the convergence time, let us settle on a sub-optimal keep-alive interval α'. We want to ensure that it admits an error that is no more than q fraction of α, for some $0 \leq q < 1$. That is: $\alpha' \geq (1-q)\alpha$. Let us continue testing so long as $\frac{high-low}{high} > q$. In that case, when the testing is completed, we can write:

$$\begin{aligned} \alpha' &= low \\ &\geq (1-q)high \\ &\geq (1-q)\alpha \end{aligned}$$

Thus with this simple modification, we can settle to a sub-optimal keep-alive interval with an assurance that it admits error no more than q fraction of the actual optimal keep-alive interval. We implemented this change and ran our algorithms with different values of q to see the impact on convergence time. As can be seen from Figure 5, the average decrease in



(a) Binary search. (b) Exponential search. (c) Composite search.

Fig. 4. Probe count of different search techniques in presence of network delay.

convergence time shows a linear relationship with q. Binary search has the most reduction in convergence time.

It is not necessary to hard code the value of q into the algorithm. Rather, it could be tuned dynamically. The server could send different values of q at different times to different nodes, if it needed to balance some load. The value could be embedded into the response to the KA message sent by the client.

C. Number of Keep-alives Sent

From the commencement of testing, we counted the total number of keep-alive packets sent through the data and test connection over a period of time. The smaller this number, the better it is. Because, each time a keep-alive is sent, there is the cost of bringing the radio to high power state, if the device was otherwise idle. We conducted this experiment for several different time durations that reflect some real world scenarios, where a device remains in the same network settings for some amount of time.

Firstly, we performed a 30 minute and a 1 hour run. These are typical durations of meetings in any organization.We also simulated a 2 hour run. This matches the duration of graduate classes, seminars or mini workshops. We plot the number of keep-alives sent over the data and the test connection against the binding timeout of middle-box. The resulting curves are shown in Figure 6. No testing is performed in case of the curve marked as 'Default'. One keep-alive packet is sent every minute in this case. On the other hand, 'Oracle' curve represents the behavior of a system that knows the optimal keep-alive interval apriori. Observe that the curve for binary search matches the Default curve in Figure 6a and Figure 6b. This is because the wait time for sending the first probe over the test connection is longer (65 minutes) than the duration of the scenario. As such, no improvements could be offered by the binary search technique. Figure 6c shows reduction in number of keep-alive sent using binary search technique. The duration of this run permitted one or more probes, which results in improved keep-alive interval.

Let us take a look at the behavior of composite and exponential search, on the contrary, in Figure 6. The curves for these approaches look identical. Both these techniques are able to reduce the number of keep-alive packets sent significantly. This is because, the initial improvements to the keep-alive intervals during the testing happens much earlier in these approaches. The curves are within 20 units (along Y-axis) of the Oracle curve in Figure 6c.

TABLE III
AVERAGE REDUCTION (%) OF NUMBER OF KEEP-ALIVES SENT WHEN TESTING IS PERFORMED TO IMPROVE THE KEEP-ALIVE INTERVAL FROM THE CONSERVATIVE DEFAULT VALUE.

Experiment duration	Binary search	Exponential search	Composite search
30 minutes	0	64.30	64.25
1 hour	0	77.79	77.83
2 hours	26.79	84.92	85.05
6 hours	71.62	90.88	91.06
8 hours	77.52	91.74	92.09
12 hours	83.47	92.84	93.13
24 hours	89.61	94.12	94.39

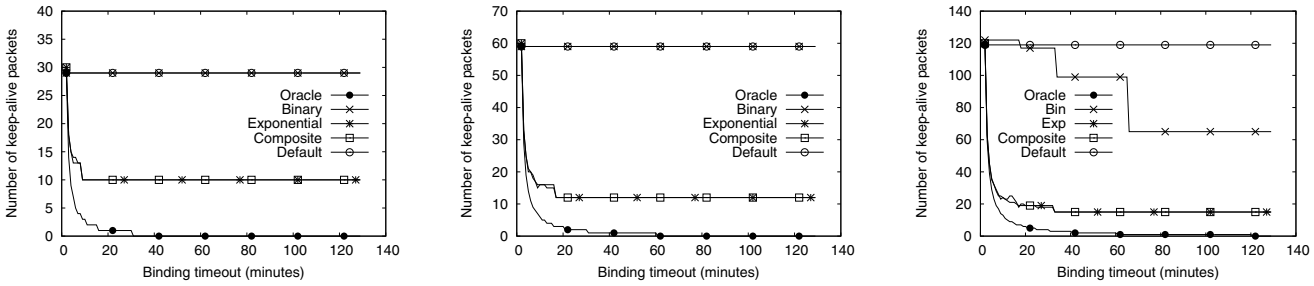
Next, we experimented with 8, 12 and 24 hour long runs. The number of keep-alives sent in all these cases, for the different search techniques are shown in Figure 7. In all these runs, composite search and exponential search curves are almost identical and they approach the 'Oracle' curve with increasing study durations.

For each different binding timeout, we calculated the percentage reduction in the number of KA packets sent using the different search techniques, compared to no testing. Then we averaged this over the entire range of binding timeouts used in the experiments: [1, 128]. This average percentage reduction in number of KA packets sent is listed in Table III. Based on this result, it is clear that exponential and composite search techniques should be preferred to binary search. And since composite search has the better convergence time of the two, it is the best choice.

D. Impact of Packet Failure

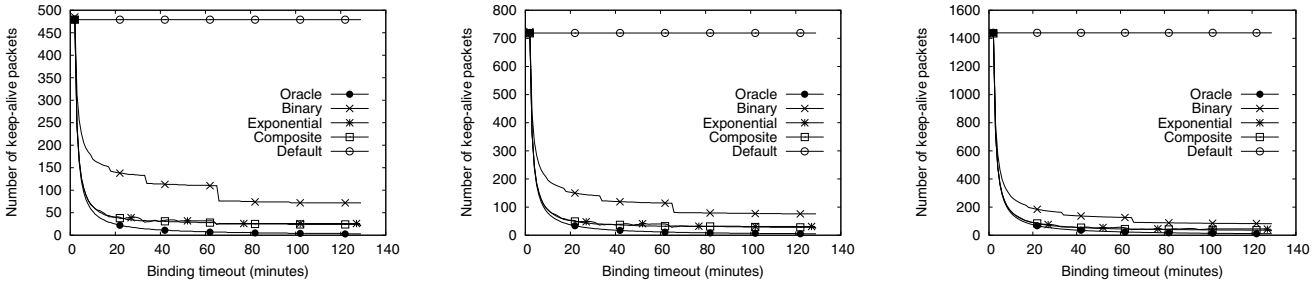
So far, when a keep-alive message fails over the test connection, we have assumed that the middle-box has dropped the connection due to too long an idleness. However, a packet could also be dropped for some transient issues in the network. The sender has no way to differentiate between the different causes of failure. As such, the technique we developed may not considerably improve KA interval.

In this experiment, we try to simulate transient network failures and observe the behavior of the different search tech-



(a) 30 minute run. (b) 1 hour run. (c) 2 hour run.

Fig. 6. Number of keep-alive packets sent during meeting or seminar scenario.



(a) 8 hour run. (b) 12 hour run. (c) 24 hour run.

Fig. 7. Number of keep-alive packets sent during longer durations.

niques. To model the transient failures, we define a parameter p that represents the probability that a keep-alive message will fail. The message failures are independent of each other. For different values of p, we observe the detected binding timeout. For each value of p, we repeated our experiment 20 times and averaged the results over those runs. Figure 8 plots the detected binding timeouts against the actual binding timeouts. The curve marked as 'No Error' represents the detected binding timeout when there is no error involved. From these curves, it is clear that, impact of packet failure cannot be neglected.

E. Retry on Packet Failure

We modified each algorithm as follows: When keep-alive message fails, we re-establish the TCP connection and test the same interval again. It is unlikely that both packets will encounter the transient failure. So, if the latter attempt fails too, we conclude that we have overshoot the optimal KA interval. With this modification, we re-ran the same experiments. The results are shown in Figure 9. For $p = 0.02$ and $p = 0.05$, the error in the detected binding timeout is very negligible.

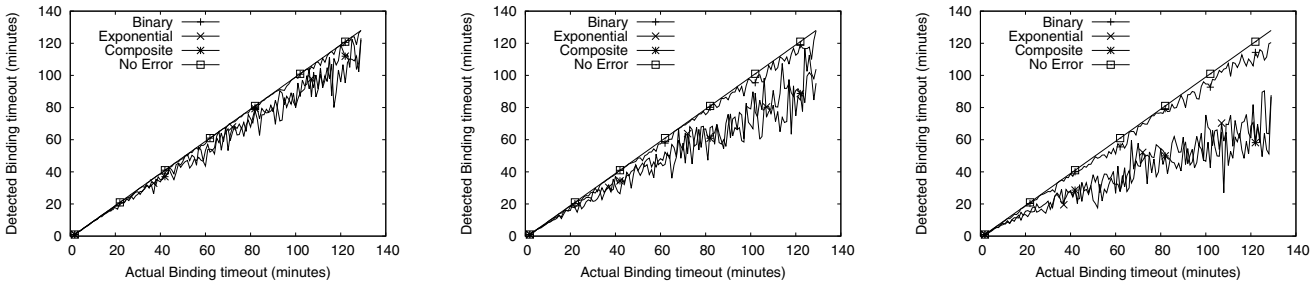
For $p = 0.10$, the retry scheme in binary search approach is quite successful in coping with transient network issues. On average, the error in detected binding timeout is less than 0.5%. In case of exponential search, the error remaining in detected binding timeout is around 5%, on average; and for composite search, it is around 3.5%. While the error is not negligible, we think this is within tolerable limit.

With single retry scheme, we encounter double the wait time for each valid keep-alive failure. Hence the convergence time grows significantly. Binary search encounters the most increase in convergence time: around 60% on average. Probe count in each technique is also increased due to the retry scheme. Since the error in detected binding timeout is in tolerable range and there is significant impact on convergence time and probe count for each number of additional retries, we would not implement more than single retry on packet failure.

With the retry scheme, we would now like to know how well do the different techniques perform in reducing the number of keep-alive packets sent over the data and the test connection combined. Like earlier, we conducted runs of different durations reflecting real world scenarios. The packet failure probability p was set to 0.02. q was set to 0.10. (Recall that q is the allowed error in the detected binding timeout, measured as a fraction of actual binding timeout.)

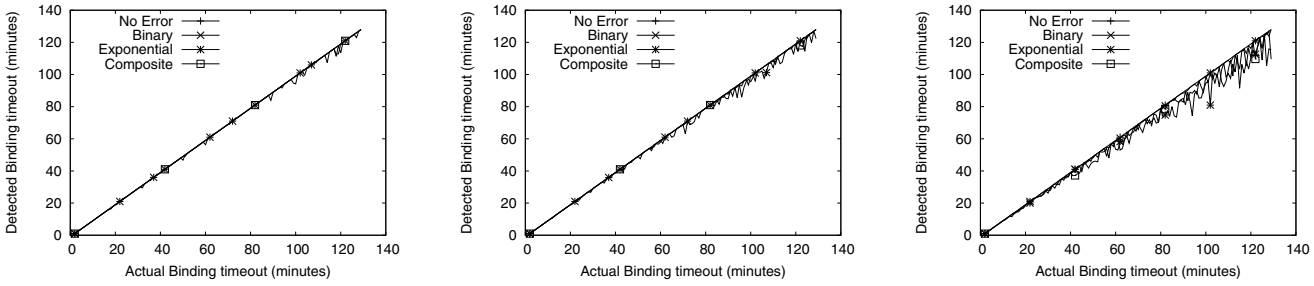
The result for each binding timeout was averaged over 20 repetitions. The results are shown in Figure 10 and Figure 11. These curves are very similar to the corresponding curves of Figure 6 and Figure 7. The average percentage reduction in number of KA packets sent is listed in Table IV. The values are comparable to the corresponding values in Table III.

Therefore, we can conclude that the retry scheme successfully coped with transient network failures and was able to reduce the number of keep-alive packets sent over the data and test connection considerably. Using the q parameter, some accuracy of detected binding timeout is sacrificed to reduce the convergence time. Even then, the reduction in number of keep-alive packets sent were very much comparable with earlier experiment. Overall, based on all the experiments, we conclude that composite search technique should be used to dynamically improve keep-alive interval.



(a) $p = 0.02$. (b) $p = 0.05$. (c) $p = 0.10$.

Fig. 8. Detected vs. actual binding timeout in presence of transient network errors.



(a) $p = 0.02$. (b) $p = 0.05$. (c) $p = 0.10$.

Fig. 9. Detected vs. actual binding timeout when retry scheme is applied in presence of transient network errors.

TABLE IV
AVERAGE REDUCTION (%) OF NUMBER OF KEEP-ALIVES SENT WHEN TESTING IS PERFORMED TO IMPROVE THE KEEP-ALIVE INTERVAL FROM THE CONSERVATIVE DEFAULT VALUE. HERE $p = 0.02$ AND $q = 0.10$.

Experiment duration	Binary search	Exponential search	Composite search
30 minutes	0	62.37	62.37
1 hour	0	75.44	75.58
2 hours	26.45	82.39	82.62
6 hours	71.59	88.19	88.56
8 hours	77.57	89.03	89.52
12 hours	83.59	90.10	90.56
24 hours	89.61	91.32	91.69

F. Proof of Concept Implementation

We implemented the composite search technique with no retry on an Andriod device. We deployed a server at kopottakha.cs.uiuc.edu on port 8080 and had the device connect to it. This proof-of-concept system only used a test connection to detect the optimal KA interval. Updating the KA interval on the data connection was not done - that would require changes in the OS code. We used our system in 2 different mobile operator's networks. The optimal KA interval detected were 9 and 10 minutes. The sequence of tests and the convergence time in the real implementation matched with our analytical bounds. We also ran the test over WiFi networks.

VI. CONCLUSION

In this research, we applied several iterative probing techniques to dynamically adapt the keep-alive interval of long-lived TCP connections. These include binary search, exponential search and composite search. We performed theoretical analysis as well as experiments on a simulation platform to compare these techniques. To the best of our knowledge, such analysis has not been done in any earlier work. We evaluated the performance of our techniques by varying different parameters and found composite search to be the best choice.

Other search techniques could be explored in future research work. In particular, searching with multiple test connections can reduce the convergence time significantly and also improve the keep-alive intervals quickly. However, care should be taken to not overload the server with too many connections. Occasionally, it is possible that due to changes in the network infrastructure, the binding timeout of the middle-box has decreased. In that case, the data connection will experience frequent disconnections. Experiments should be conducted to develop a strategy to bring the data connection out of this unstable state (frequent disconnections). Finally, work could be done to write a redistributable library that can be plugged into any device to improve the keep-alive interval. The API and protocol design for such a library remains to be investigated.

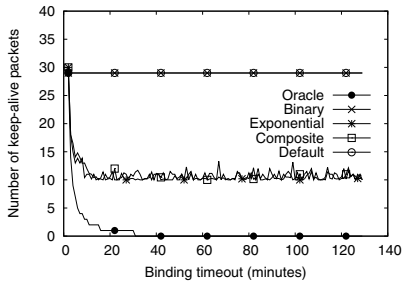
ACKNOWLEDGMENT

The authors would like to acknowledge all the anonymous referees for their valuable suggestions.

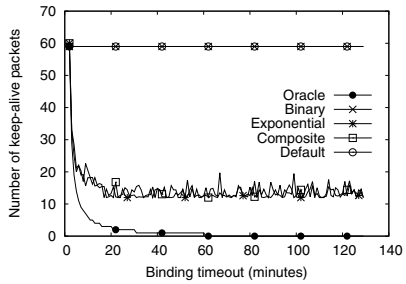
REFERENCES

[1] S. Hätönen, A. Nyrhinen, L. Eggert, S. Strowes, P. Sarolahti, and M. Kojo, "An experimental study of home gateway characteristics," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 260–266.

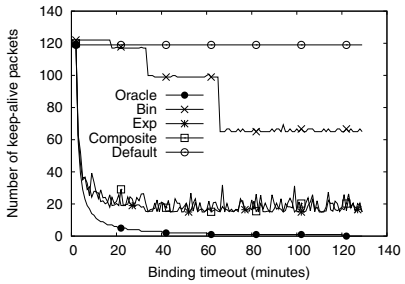
[2] R. Braden, "Requirements for Internet hosts-communication layers," October 1989, RFC 1122 (Standard).



(a) 30 minute run.

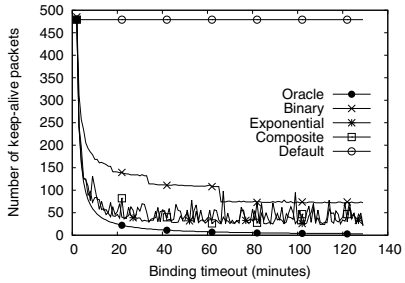


(b) 1 hour run.

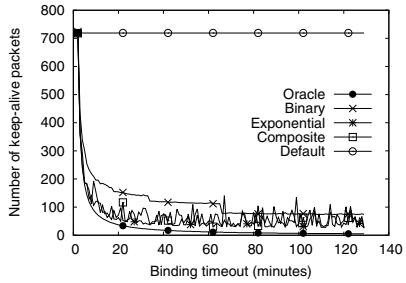


(c) 2 hour run.

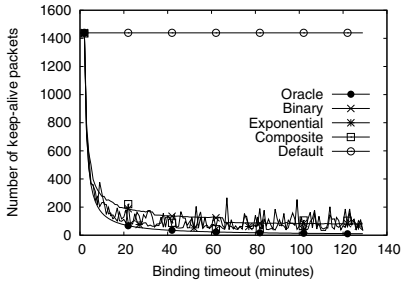
Fig. 10. Number of keep-alive packets sent during meeting or seminar scenario. ($p = 0.02$, $q = 0.10$).



(a) 8 hour run.



(b) 12 hour run.



(c) 24 hour run.

Fig. 11. Number of keep-alive packets sent during longer durations. ($p = 0.02$, $q = 0.10$).

[3] S. Guha, K. Biswas, B. Ford, S. Sivakumar, and P. Srisuresh, "NAT Behavioral requirements for TCP," October 2008, RFC 5382 (Best Current Practice).

[4] M. Stiemerling, E. Davies, C. Aoun, and H. Tschofenig, "NAT/Firewall NSIS Signaling Layer Protocol (NSLP)," October 2010, RFC 5973 (Experimental).

[5] M. Stiemerling, J. Quittek, and C. Cadar, "NEC's Simple Middlebox Configuration (SIMCO) Protocol Version 3.0," May 2006, RFC 4540 (Experimental).

[6] "Understanding Direct Push," http://technet.microsoft.com/en-us/library/aa997252(EXCHG.80).aspx, [Online; Last accessed on 22-Oct-2015].

[7] "Heartbeat Interval Adjustment," http://technet.microsoft.com/en-us/library/cc182270.aspx, [Online; Last accessed on 22-Oct-2015].

[8] "Apple Push Notification Service," https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html, [Online; Last accessed on 22-Oct-2015].

[9] "Google Cloud Messaging: Overview," https://developers.google.com/cloud-messaging/gcm, [Online; Last accessed on 22-Oct-2015].

[10] "Push Notifications (Windows Phone)," https://msdn.microsoft.com/en-us/library/hh221549.aspx, [Online; Last accessed on 22-Oct-2015].

[11] "Windows Push Notification Services (WNS) overview (Windows Runtime apps)," http://msdn.microsoft.com/en-us/library/windows/apps/hh913756.aspx, [Online; Last accessed on 22-Oct-2015].

[12] S. R. Gatta, K. Srinivasan, O. N. Ertugay, D. G. Thaler, D. A. Anipko, J. Vanturennout, M. S. Rahman, and P. R. Gaddehosur, "Keep alive management," November 2014, US Patent No. 8,892,710 B2.

[13] S. Herzog, R. Qureshi, J. Raastroem, X. Bao, R. Bansal, Q. Zhang, and S. M. Bragg, "Determining an efficient keep-alive interval for a network connection," February 2013, US Patent No. 8,375,134 B2.

[14] R. Seggelmann, M. Tuexen, and M. Williams, "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension," February 2012, RFC 6520 (Standard).

[15] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) protocol version 1.2," August 2008, RFC 5246 (Standard).

[16] H. Haverinen, J. Siren, and P. Eronen, "Energy consumption of always-on applications in WCDMA networks," in *Proceedings of IEEE Vehicular Technology Conference*. IEEE, April 2007, pp. 964–968.

[17] N. Vallina-Rodriguez and J. Crowcroft, "Energy management techniques in modern mobile handsets," *Communications Surveys & Tutorials, IEEE*, vol. 15, no. 1, pp. 179–198, 2013.

[18] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: a measurement study and implications for network applications," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*. ACM, 2009, pp. 280–293.

[19] D. J. Wetherall and A. S. Tanenbaum, "Computer Networks," 1996.

[20] M. Jain and C. Dovrolis, "End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput," in *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4. ACM, 2002, pp. 295–308.

[21] A. Akella, S. Seshan, and A. Shaikh, "An empirical evaluation of wide-area internet bottlenecks," in *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*. ACM, 2003, pp. 101–114.

[22] N. Hu and P. Steenkiste, "Evaluation and characterization of available bandwidth probing techniques," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 6, pp. 879–894, 2003.

[23] B. Melander, M. Bjorkman, and P. Gunningberg, "A new end-to-end probing and analysis method for estimating bandwidth bottlenecks," in *Global Telecommunications Conference, 2000. GLOBECOM'00*, vol. 1. IEEE, 2000, pp. 415–420.

[24] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell, "pathChirp: Efficient available bandwidth estimation for network paths," in *Proceedings of Passive and active measurement (PAM) workshop*, April 2003.

[25] M. Jain and C. Dovrolis, "End-to-end estimation of the available bandwidth variation range," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, no. 1. ACM, 2005, pp. 265–276.

[26] M. S. Rahman, "On Approaches to Detect Optimal Keep-alive Interval of TCP Connections." http://1drv.ms/1kktYFu, M.Sc. thesis, submitted to the Department of Computer Science and Engineering (CSE), Bangladesh University of Engineering and Technology (BUET). [Online; Last accessed on 22-Oct-2015].

[27] "Omnet++," http://www.omnetpp.org/, [Online; Last accessed on 22-Oct-2015].