Using Adaptive Heartbeat Rate on Long-Lived TCP Connections

M. Saifur Rahman, *Member, ACM*, Md. Yusuf Sarwar Uddin, *Member, IEEE, ACM*, Tahmid Hasan, M. Sohel Rahman, *Senior Member, IEEE, ACM*, and M. Kaykobad

***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 device connects to a server using TCP, often times the connection is established through some sort of middle-box, such as NAT, proxy, firewall, and so on. When such a connection is idle for a long time, it may get torn down due to binding timeout of the middle-box. 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 binding timeout. Due to varied settings of different network equipments, the binding timeout 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 the middle-box binding timeout 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.**

***Index Terms*— Keep-alive, middle-box, push notification, TCP, binding timeout, binary search, iterative probing.**

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 over Internet Protocol (IP), 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 getting timely updates using push notification technology [1]. 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 pushed 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.

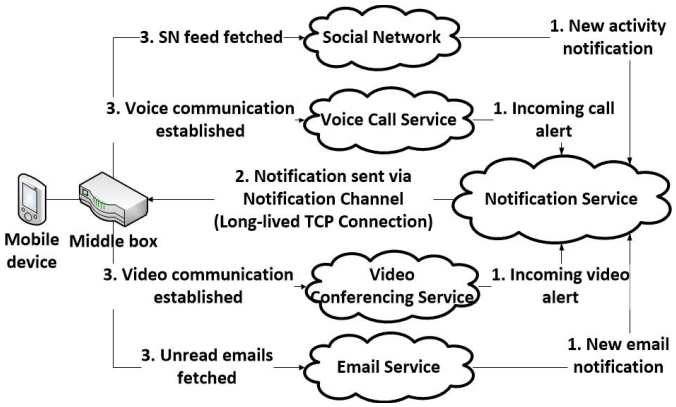

Fig. 1. A model of notification service used in different mobile platforms.

As an example of Voice over IP (VoIP) scenario, let us see how an Android user receives a *Viber* [2] call in his mobile phone. When a call is made through Viber network, the Viber service notifies the Google Cloud Messaging (GCM) Service [3]. Then an alert is sent to the callee's phone. The phone at that moment may be in "screen-off" state in the user's pocket. While a lay person may think the phone is turned off, in reality it is in low power mode, still capable of receiving notifications via the notification channel. When the notification is received in the phone, it moves to high power state and the Viber application installed in the device is able to generate the ring for the incoming call. As such, the user is now able to receive the call. If, however, the connection to the notification server was broken for some reason, the user would not be able to receive the Viber call.

Therefore, to enable these seemingly *always connected* scenarios, the device must keep the notification channel alive

Manuscript received April 2, 2016; revised November 30, 2016, March 14, 2017, and July 3, 2017; accepted November 2, 2017; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor S. Mascolo. Date of publication December 15, 2017; date of current version February 14, 2018. *(Corresponding author: M. Saifur Rahman.)*

The authors are with the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka 1205, Bangladesh (e-mail: mrahman@cse.buet.ac.bd; yusufsarwar@cse.buet.ac.bd; 1405004.th@ugrad.cse.buet.ac.bd; msrahman@cse.buet.ac.bd; kaykobad@cse.buet.ac.bd).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/TNET.2017.2774275

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 timeout. That means, the connection state may be cleared by the middle-box if no data passes through it for some specified amount of time.

Studies have shown that the NAT binding timeouts vary dramatically amongst commercially available home gateways [4]. 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 [5] or a heartbeat; with the interval being referred to as keep-alive interval or KA interval in short. A single KA transaction may consume between 0.15 mAh and 0.6 mAh energy in 3G network [6]. That means, a phone with 1800 mAh battery may spend upto 12% of its energy budget in a day just by KA transactions, if the KA interval is fixed to 4 minutes. The KA interval thus consumes a significant fraction of total power used by the device.

To balance the battery life constraint with the necessity of keeping the connection alive, it is preferred that the KA probe is made at the farthest possible time within the binding timeout. However, in the absence of knowledge about binding timeout in the middle-boxes for all sorts of edge networks a device can be put behind, the most obvious strategy is to use an optimistic but *fixed* lower interval for sending keep-alives. To the best of our knowledge, this strategy is indeed used in many production environments to date, and there are documented records of users complaining about losing notifications due to this fixed interval (when this interval is longer than the binding timeout) [7], [8]. This also leads to sub-optimal operation when the underlying middle-box can actually stand longer duration than the interval suggests. As a remedy, we can *dynamically* adjust this interval by probing at varying intervals. While trying at different intervals, we do not want to disturb the functional connection between the host and the server (which we refer to as *data connection*). Instead, we propose to use a separate *test connection* on which we probe at different intervals and once succeeded, we let the interval to be enacted on the data connection.

To this end, we propose several iterative probing strategies to dynamically adjust the KA interval of the data connection. These include binary, exponential and composite search. Composite search combines different aspects of binary and exponential search techniques. We analyze these 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.

Notably, an earlier checkpoint of this work was presented in NSysS'16 [9]. Since then, we have augmented the theoretical analysis parts and conducted further experiments in simulation platform as well as in real world implementation.

The rest of the paper is organized as follows. Section II reviews some earlier work relevant to our research. Section III describes several techniques to dynamically change the KA interval of TCP connection. Section IV provides analytical bounds on several properties of these techniques. Section V describes the simulation, real world 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 the middle-boxes. This motivates the need for sending KA 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 use iterative probing for measuring different network parameters.

A. Middle-Box Binding Timeout

Computers in a local network connect to the internet through stateful components like NAT, firewalls, proxy servers etc. These components maintain some state for each TCP connection. These are termed as middle-boxes, as they stay in the middle between a computer and the internet. A wide variety of middle-boxes are in place in today's enterprise and home networks. Sherry *et al.* [10] conducted the first large-scale survey of middle-box deployments. Their dataset included small, medium, large and very large networks. They categorized the middle-boxes into 8 different categories.

Hätönen *et al.* [4] focused on one particular type of middle-box—home gateway. A home gateway may perform many functionalities, spanning various link-layer technologies. It also performs higher-layer operations, such as network address translation (NAT), DNS proxy, DHCP and firewall functionalities. As such, it falls into many of the categories specified in [10], such as L3 router, L2 switch, IP firewall, application firewall, proxies, application gateway etc.

Hätönen *et al.* [4] 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 [11], 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.

While the home gateways cause idle timeout at the edge of home networks, components like load balancers can cause the same at the edge of the cloud. For example, Microsoft Azure's load balancer enforces an idle timeout of between 4 to 30 minutes for inbound connections [12].

Even though there is no literature on binding timeout of all possible types of middle-boxes, we can infer from the above discussion that the binding timeouts in all types of middle-boxes vary widely. As such a consumer device in real network environment may need to perform testing to find the binding timeout and send KAs using an interval smaller than that. While protocols such as NSIS [13] or SIMCO [14] 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 [15]. 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 timeout, then no response is received from the server. Then the client sends another HTTPS request with a lesser amount of wait time. Detailed discussion on the KA interval adjustment can be found in [16].

Android, iPhone, Windows Phone etc. platforms maintain a persistent connection between the client (device) and a notification service [3], [17]–[19]. Each of these eco-systems employs exchange of KA messages periodically and has mechanism to tune the KA interval to obtain good battery life performance. Patents granted to Microsoft Corporation indicate use of a test connection to dynamically detect an efficient KA interval [20], [21]. However, the exact strategy used in choosing the test intervals is not called out in the patents. The Heartbeat Extension [22] of Transport Layer Security (TLS) protocol [23] also uses KA messages for ensuring liveness of peers.

C. Impact of KA Interval on Power Consumption

Haverinen *et al.* showed in [6] that battery life is significantly influenced by the frequency of KA messages. They performed power measurements in two different 3G networks, as well as, in 2G GPRS network. Li et al. created a framework called MobileQoE [24] to improve the battery life for users' phones, as well as to reduce the cost of signaling. They provide a measurement approach for cellular radio resources consumed by KA messages. They also provide an SDK for android smart phones which can coordinate with different long-lived connections to reduce the total radio resource.

A comprehensive survey on general solutions for energy efficiency on mobile devices, published between 1999 and May 2011, is available in [25]. Here the authors provide a short summary of the various efforts on studying, modeling and reducing energy consumption in mobile devices. We focused on one of the power models, due to Balasubramanian *et al.* [26], in which the energy spent to transfer data over the cellular network consists of three components: ramp, transmission and tail energy. The amount of ramp and transmission energy consumed depends on the amount of data transfer. Once the data transfer is completed, the interface remains on for certain amount of time, called the tail-time. The energy cost during this time is called the tail energy. Since the KAs are sent with a period that is longer than the tail-time, each KA incurs the overhead of the tail-time, if the device is otherwise idle at that time. In that case, by reducing the number of KAs, 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. Jain et al. used iterative probing to measure end-to-end avail-bw. Their measurement methodology was implemented in a tool called pathload [27]. Other iterative techniques for avail-bw measurements include Bfind [28], PTR [29], TOPP [30], pathChirp [31] etc. The variability in the avail-bw has also been measured by Jain et al. using iterative probing [32].

The work of Price and Tino [33] is relevant to our work. They applied iterative probing in a separate (test) connection to adapt to the binding timeout of NAT devices in a P2P overlay network. Their approach of binary search to improve the KA interval is equivalent to what we call composite search. We, however, studied the problem more formally, proposed several alternative techniques and made mathematical characterization of those techniques in a more general setting. Additionally, we provided real implementation of adaptive KA scheme in mobile phone platform.

III. Adaptive Keep-Alive Techniques

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

The probing for improved KA intervals is conducted in a connection which we call a *test* connection. This is separate from the data connection (e.g. notification channel). This is done to avoid any disruption of service. As the probing in the test connection reveals improved intervals, the new intervals are applied on the data connection right away. Initially, however, the notification channel applies a conservative KA interval. 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 an upper bound on the search space, which is 1 minute less than the minimum KA interval that is already known to not work. Note that we use intervals in minute boundaries only.

A. Steps in Iterative Probing

Figure 2 shows a flow chart of the steps taken in dynamically improve the KA interval. The search range for the iterative probing has been represented as $[low, high]$. Initially, the data connection uses low as the KA interval. A separate TCP connection (*test connection*) with the target service is opened for iterative probing. Starting from the lower bound, we try to improve by guessing new KA intervals. Once a guess is made, the connection is kept silent for that much time. Afterwards a KA 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. The KA interval in the data connection is updated accordingly. Now, a higher KA interval is guessed and tested in the test connection. On the other hand, if the connection was dropped, then we lower the guess and perform the test again. This process is continued until the difference between 2 consecutive guesses is less than 1 minute.

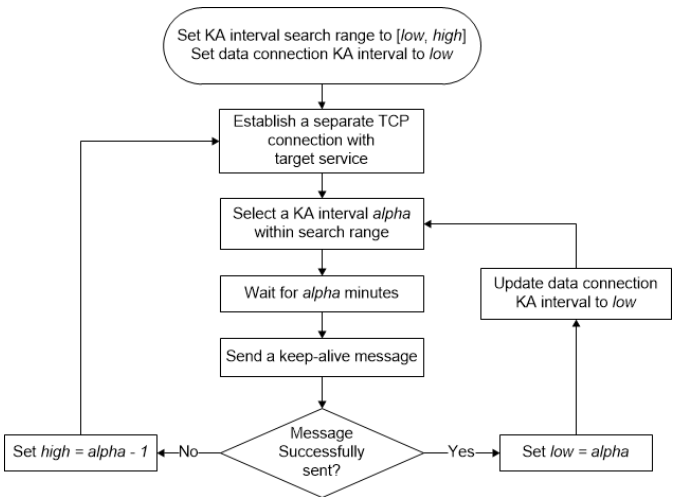

Fig. 2. Flow chart for dynamically improving the KA interval.

The probing sequence will be triggered whenever a device detects that it is under a new mobile network or WiFi access point. A device can look at the Access Point Name (APN) of the mobile network or the MAC address of the WiFi access point for this identification. This identifier can also be used as a key to cache improved KA interval found from prior testing. Therefore, even if the device is not continually under the same access point, incomplete tests can be resumed whenever the device returns.

We explore three different probing techniques. 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 binding timeout. The problem with this approach, however, is that the first probe is made after a long period of waiting during which 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, it takes very long time to detect the binding timeout. Therefore, we finally combined the two approaches into what we call the composite search and were able to get good results.

For each of these techniques, we initialize low with 1 minute. This is a reasonable choice, since it is smaller than the smallest KA interval in a home gateway, as found in [4]. Exponential and composite search techniques do not need to specify an upper 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.

B. 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: $\{65^*, 33^*, 17, 25^*, 21, 23, 24\}$. The probes that failed are annotated with '*' mark.

C. 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 low and $high$ becomes identical, at which point that value is reported as the binding timeout. For example, if the binding timeout is 24 minutes, the sequence of intervals tested would be: $\{2, 4, 8, 16, 32^*, 17, 19, 23, 31^*, 24, 26^*, 25^*\}$.

D. 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: $\{2, 4, 8, 16, 32^*, 24, 28^*, 26^*, 25^*\}$.

E. Mathematical Rationale Behind Composite Search

As mentioned earlier, binary search based probing initially has long period of waiting during which time no improvements can be made to the KA interval in the data connection. Exponential search, on the other hand, probes with shorter wait times initially. However, it takes very long time to detect the binding timeout. Therefore, we combined the two approaches into composite search and were able to get good results in our experimentation. While results from our experiments led us to composite search, we nonetheless are able to provide mathematical rationale behind its success as follows:

Let l and h respectively be the lower and upper end of the search range. Therefore, l is the largest known KA interval that works. We want to improve it further, by probing the test connection at an interval longer than l. Let this additional jump be t, such that $l+t \leq h$. If the probe at interval $(l+t)$ succeeds, we would get a longer KA interval, which gives us "reward" of saving the total number of KAs sent in longer run. The savings is in the order of $\left(\frac{1}{l} - \frac{1}{l+t}\right)$. Now, the choice of t has odds. Choosing a longer t means waiting for longer duration before knowing whether that attempted interval actually works

(which could even fail). On the other hand, choosing a shorter t, gives us quicker results but does not give much rewards. We are interested to find a jump that maximizes the reward per unit of wait time. So, the marginal reward function that we would like to maximize is:

$$R(t) = \frac{1}{l+t}\left(\frac{1}{l} - \frac{1}{l+t}\right)$$

By examining the first and second derivative of $R(t)$, we find that reward is maximized when $t = l$. And we can rewrite the constraint "$l + t \leq h$" as "$h \geq 2l$". Therefore, as long as the search range higher bound is at least twice as much the lower bound, we should always double up the current interval and probe for that. This is exactly what composite search does until the first probe failure occurs.

Once a probe fails, h is reduced and $h \geq 2l$ does not hold anymore. Now, with each probe, we want to reduce the size of this range as much as possible. The initial range or gap size is $d = h - l$. If the next probe is made with interval $l + t$, this may reduce the gap to t (if the probe fails) or $(d - t)$ if it succeeds. Since α is equally likely to be anywhere in the range, then $P\{\text{The probe fails}\} = P\{\alpha < l + t\} = \frac{t}{d}$.

Therefore, conditioned on whether the next probe at interval $l + t$ succeeds or fails, the expected next gap size is:

$$G(t) = t \times \frac{t}{d} + (d - t) \times (1 - \frac{t}{d}) = \frac{1}{d}\{t^2 + (d - t)^2\}$$

By examining the first and second derivative of $G(t)$, we find that the gap is minimized when $t = \frac{1}{2}d$. Hence, once an interval range has been identified, next probe should be made at its mid-point.

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 binding timeout (*Probe Count*), and total time taken to do so (*Convergence Time*). Our schemes can be compared based on these.

Let α denote the binding timeout. Let the search range be $[l, h]$. For binary search, we assume $h = 2^k$, for some $k \geq 1$. For exponential and composite search, $h = \infty$. For all our analysis we have taken $l = 1$. Let $N(\alpha)$ denote the number of test probes needed to detect the binding timeout. 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 binding timeout 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. As needed, superscript is added from {'binary', 'exp', 'comp'} to the notations to identify the KA schemes. For convenience, the notations have been tabulated in Table I. In the table and the subsequent derivations, we have conveniently used lg to represent base 2 logarithm (i.e., $\log_2$).

In what follows, we derive the exact expression for probe count and convergence time of the different detection techniques. For detailed derivations, the reader is referred to the Appendix 1 provided as supplementary material.

TABLE I
Notations Used in Analytical Expressions

Notation	Interpretation
l	Lower bound of the KA interval search range
h	Higher bound of the KA interval search range
k	$\lg h$ (i.e., $\log_2 h$)
α	Middle-box binding timeout
N	Probe count of a KA scheme
T	Convergence time of a KA scheme

A. Binary Search

In binary search, the total number of test probes for any value of α would be $\lg h$. Since it is independent of α, the maximum, minimum and average will also be $\lg h$. Therefore,

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

Let $a_{k-1}a_{k-2}\ldots a_1a_0$ denote the binary representation of $\alpha - 1$. We observe that, 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. Therefore, we can write:

$$\begin{aligned} T^{binary}(\alpha) &= \sum_{i=0}^{k-1}(1 + 2^i) + \sum_{i=0}^{k-1} a_i i 2^i \\ &= (h + \lg h - 1) + \sum_{i=1}^{\lg h - 1} a_i i 2^i \end{aligned}$$

Clearly, convergence time is a monotone increasing function of binding timeout. Therefore, the minimum or best case is:

$$T_{best}^{binary} = T^{binary}(1) = h + \lg h - 1 \approx O(h) \tag{2}$$

And the maximum or worst case would be:

$$\begin{aligned} T_{worst}^{binary} &= T^{binary}(2^k) \\ &= h \lg h + \lg h - h + 1 \approx O(h \lg h) \end{aligned} \tag{3}$$

We assume that the binding timeout is equally likely to be any discrete value in the search range. Therefore, the average convergence time is:

$$T_{avg}^{binary} = \frac{1}{2^k}\sum_{\alpha=1}^{2^k} T^{binary}\ (\alpha) = (\frac{h}{2} + 1)\lg h \approx O(h \lg h) \tag{4}$$

B. Exponential Search

Let p denote the number of probes upto the the first failure. Then, $p = 1 + \lfloor \lg(\alpha) \rfloor$. Afterwards, α is searched in the range $[2^{p-1}, 2^p - 1]$. As far as probe count is concerned, this is similar to as if searching for $\alpha + 1 - 2^{p-1}$ in the range $[1, 2^{p-1}]$. Let $N'(\alpha, k)$ denote the number of probes for searching α in the bounded search space $[1, 2^k]$. Then, $N'(\alpha, k)$ can be defined by the following recurrence:

$$N'(\alpha, k) = \begin{cases} k & \text{when } \alpha = 2^k \\ p + N'(\alpha + 1 - 2^{p-1}, p - 1) & \text{otherwise.} \end{cases}$$

Then, we can write:

$$N^{exp}(\alpha) = p + N'(\alpha + 1 - 2^{p-1}, p - 1) \tag{5}$$

From Equation 5, using the mathematics in the Appendix, we can write:

$$N_{best}^{exp} = 1 \tag{6}$$

$$N_{worst}^{exp} = \begin{cases} \lg h + 2 & \text{when } h \leq 4 \\ \frac{\lg h(\lg h+1)}{2} & \text{otherwise.} \end{cases} \tag{7}$$

$$\approx O(\lg^2 h)$$

$$N_{avg}^{exp} = \frac{\lg h(1+\lg h)}{4} + 1 + \frac{1}{h} \approx O(\lg^2 h) \tag{8}$$

Now, let us derive the expression for convergence time. As mentioned earlier, after the first failure occurs, α is searched in the range $[2^{p-1}, 2^p - 1]$. This subsequent search is similar to searching for $\alpha + 1 - 2^{p-1}$ in the range $[1, 2^{p-1}]$, except that each probe takes an additional time of $2^{p-1} - 1$.

Let $T'(\alpha, k)$ denote the convergence time, when searching α in the bounded search space $[1, 2^k]$. Then, $T'(\alpha, k)$ can be defined by the following recurrence:

$$T'(\alpha,k) = \begin{cases} 2^{k+1} - 2 & \text{when } \alpha = 2^k \\ 2^{p+1} - 2 & \text{otherwise.} \\ +T'(\alpha + 1 - 2^{p-1}, p-1) & \\ +(2^{p-1} - 1) & \\ \times N'(\alpha + 1 - 2^{p-1}, p-1) & \end{cases}$$

Then, we can write:

$$T^{exp}(\alpha) = 2^{p+1} - 2 + T'(\alpha + 1 - 2^{p-1}, p-1) + (2^{p-1} - 1)N'(\alpha + 1 - 2^{p-1}, p-1) \tag{9}$$

From equation 9, using the mathematics in the Appendix, we can write:

$$T_{best}^{exp} = 2 \tag{10}$$

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

$$\approx O(h \lg^2 h) \tag{11}$$

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

$$\approx O(h \lg^2 h) \tag{12}$$

C. Composite Search

As discussed before, the number of probes upto the first failed probe is $p = 1 + \lfloor \lg(\alpha) \rfloor$. Afterwards, α is searched in the range $[2^{p-1}, 2^p - 1]$, using binary search. The search range size is 2^{p-1}. As such, number of probes during binary search is $p - 1$. Therefore,

$$N^{comp}(\alpha) = p + p - 1 = 1 + 2\lfloor \lg(\alpha) \rfloor \tag{13}$$

Clearly, $N^{comp}(\alpha)$ is monotonic. We obtain (see Appendix for details):

$$N_{best}^{comp} = 1 \tag{14}$$

$$N_{worst}^{comp} = N^{comp}(h) = 1 + 2 \lg h \approx O(\lg h) \tag{15}$$

$$N_{avg}^{comp} = 2 \lg h + \frac{2}{h}(2 + \lg h) - 3 \approx O(\lg h) \tag{16}$$

TABLE II
COMPARISON AMONG TECHNIQUES TO ADAPT KA INTERVAL

Technique	N_{avg}	N_{worst}	T_{avg}	T_{worst}
Binary	$O(\lg h)$	$O(\lg h)$	$O(h \lg h)$	$O(h \lg h)$
Exponential	$O(\lg^2 h)$	$O(\lg^2 h)$	$O(h \lg^2 h)$	$O(h \lg^2 h)$
Composit	$O(\lg h)$	$O(\lg h)$	$O(h \lg h)$	$O(h \lg h)$

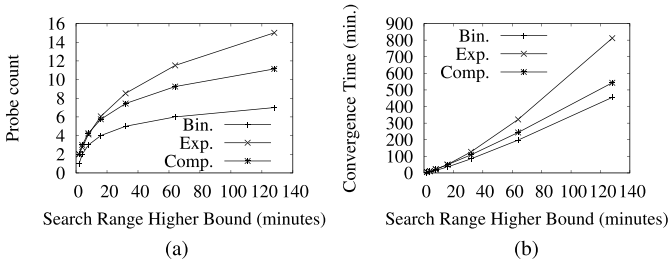

Fig. 3. Comparison of average case probe count and convergence time amongst different search techniques. The average is performed over different search ranges. (a) Probe count comparison. (b) Convergence time comparison.

Now, we would like to derive the convergence time $T^{comp}(\alpha)$. Let $t_e(\alpha)$ and $t_b(\alpha)$ be the time spent during the exponential and binary phases, respectively. Then,

$$t_e(\alpha) = 2 + 4 + 8 + \cdots + 2^p = 2^{p+1} - 2$$

Binary search for α is performed in the range, $[2^{p-1}, 2^p - 1]$. This is similar to searching for $(\alpha - 2^{p-1} + 1)$ in the range $[1, 2^{p-1}]$; however for each probe, there is an additional $(2^{p-1} - 1)$ wait time. Let $a_{p-1}a_{p-2}\cdots a_1 a_0$ is the binary representation of α. Therefore, we can write:

$$t_b(\alpha) = (2^{p-1} - 1)(p-1) + (2^{p-1} + p - 2) + \sum_{i=1}^{p-2} a_i i 2^i$$

Combining the above expressions, we can write:

$$T^{comp}(\alpha) = t_e(\alpha) + t_b(\alpha) = 2^{\lfloor \lg(\alpha) \rfloor}(5 + \lfloor \lg(\alpha) \rfloor) - 3 + \sum_{i=1}^{\lfloor \lg(\alpha) \rfloor - 1} a_i i 2^i \tag{17}$$

Like probe count, convergence time is also a monotonic function of α. We obtain,

$$T_{best}^{comp} = T^{comp}(1) = 2 \tag{18}$$

$$T_{worst}^{comp} = h \lg h + 5h - 3 \approx O(h \lg h) \tag{19}$$

$$T_{avg}^{comp} = (\frac{h}{2} + 1) \lg h + \frac{2}{3} h + 3 - \frac{5}{3h} \approx O(h \lg h) \tag{20}$$

Average and worst case probe count and convergence time of the different search techniques has been compared in Table II. The corresponding curves for the average case have been plotted in Figure 3a and 3b respectively.

V. EXPERIMENTS

We have implemented the different techniques to dynamically adjust the KA interval on Omnet++ simulation platform [34]. 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 binding timeout, in terms of

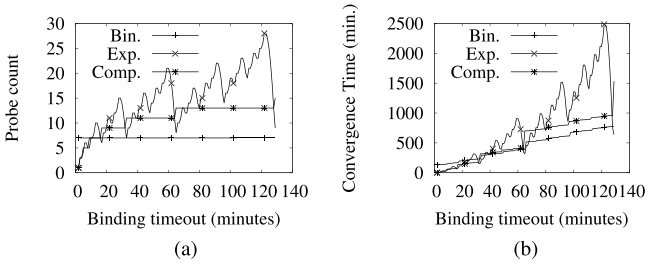

Fig. 4. Probe count and convergence time of different search techniques in presence of network delay. (a) Probe count. (b) Convergence time.

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 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.

A. Correctness of Analytical Bounds

Figure 4a plots the probe count of the different search techniques against binding timeout, as found through simulation. For the binary search, the curve is identical to theoretical curve and is a straight line parallel to the X-axis. This is expected, since the probe count in a given search range is independent of any specific binding timeout. The curve for composite search generally stays in the middle between that of binary and exponential search. Both the curves for exponential and composite search are off from their theoretical counterparts (not shown) 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 are also off from their theoretical counterparts by 1 unit along the Y-axis. The convergence time curves, as found through simulation, are shown in Figure 4b. Here, too, the curve for composite search generally stays in the middle between that of binary and exponential search.

The exact match of the simulation results with the analytical curves affirms that all of our close formed expressions of probe counts and convergence time are indeed accurate.

B. Keep-Alive Interval With Tunable Accuracy

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. In effort to reduce the convergence time, let us settle on a sub-optimal KA 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 as long as $\frac{high-low}{high} > q$. In that case, when the testing is completed, we can write:

$$\alpha' = low \geq (1-q)high \geq (1-q)\alpha$$

We implemented this change and ran our algorithms with different values of q. The average decrease in convergence time for each search technique is shown in the table below. Binary search had the most reduction in convergence time.

q	**Binary**	**Exponential**	**Composite**
0.10	31.76	13.71	27.21
0.15	39.22	18.39	34.48
0.20	44.20	23.16	39.25
0.25	51.06	35.19	45.12

Value of q can be tuned dynamically. The server can send different values at different times to different nodes, if it needs to balance some load.

TABLE III
AVERAGE CONVERGENCE TIME IN VARIATIONS OF COMPOSITE SEARCH. (AVERAGE TAKEN OVER THE SEARCH RANGE OF $[1, 128]$ MINUTES)

Search Technique	**Avg. Convergence Time**
Binary	7 hours and 35 minutes
Composite 4	7 hours and 38 minutes
Composite 8	7 hours and 43 minutes
Composite 16	7 hours and 51 minutes
Composite 32	7 hours and 59 minutes
Composite	9 hours and 3 minutes

C. Variations of Composite Search

Composite search acts like exponential search initially. Upon the first failed probe, it transitions to binary search. Here, we introduce another case for this transition—if the next interval to be tested is larger than a specified threshold r, exponential search phase is halted in that case too; and binary search is started thereafter.

The convergence time of this variation for different values of r is shown in Table III. Here, the search technique name has been annotated with the value of r used. By tuning r, upto 15% improvement could be achieved in the convergence time. Parameter r, too, can be tuned from the server.

D. Number of Keep-Alives Sent

From the commencement of testing, we counted the total number of KA packets sent through the data and test connection over a period of time. Note that this is different than the probe count, which refers to the number of KAs sent only through the test connection during the convergence time period. The smaller the number of KAs sent through the data and test connections combined, the better it is. Because, each time a KA 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 time.

Firstly, we performed a 30 minute and a 1 hour run. These are typical durations of meetings in any organization. We also

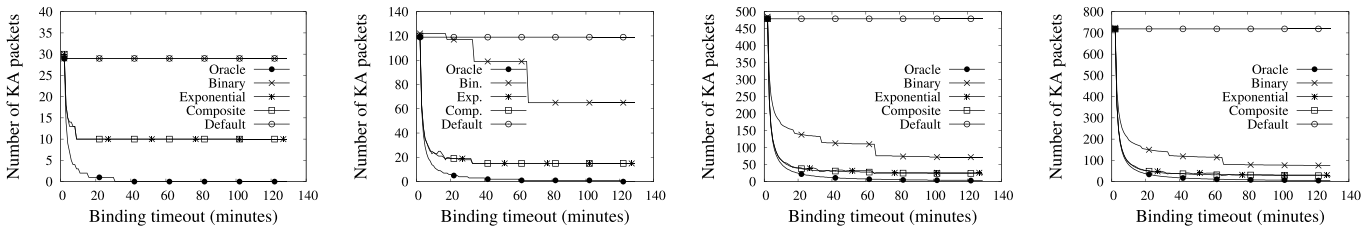

Fig. 5. Number of keep-alive packets sent during different time durations. (a) 30 minute run. (b) 2 hour run. (c) 8 hour run. (d) 12 hour run.

TABLE IV
AVERAGE REDUCTION (%) OF NUMBER OF KEEP-ALIVES SENT WHEN TESTING IS PERFORMED TO IMPROVE THE KEEP-ALIVE INTERVAL

Experiment duration	Binary search	Exp. search	Comp. ($r = 4$)	Comp. ($r = 8$)	Comp. search
30 minutes	0	64.30	61.26	64.25	64.25
1 hour	0	77.79	68.64	76.60	77.83
2 hours	26.79	84.92	75.77	82.23	85.05
6 hours	71.62	90.88	87.87	89.99	91.06
8 hours	77.52	91.74	89.69	91.29	92.09
12 hours	83.47	92.84	91.58	92.66	93.13
24 hours	89.61	94.12	93.66	94.20	94.39

simulated a 2 hour run. This matches the duration of graduate classes, seminars or mini workshops. Next, we experimented with longer runs: 8, 12 and 24 hours. We plot the total number of KAs sent during the study window against the binding timeout of middle-box. The resulting curves are shown in Figure 5. (Curves for 1 and 24 hour runs are left out for brevity). No testing is performed in case of the curve marked as 'Default'. One KA packet is sent every minute in this case. On the other hand, 'Oracle' curve represents the behavior of a system that knows the binding timeout apriori.

Observe that the curve for binary search matches the Default curve in Figure 5a. This is because the wait time for sending the first probe over the test connection is longer than the duration of the scenario. As such, no improvements could be offered by binary search. Figure 5b shows reduction in number of KAs sent using binary search. The duration of this run permitted one or more probes, which results in improved KA interval. In these as well as the longer runs, binary search is inferior to exponential and composite search. These other techniques, on the other hand, look identical. Both these techniques are able to reduce the number of KA packets sent significantly 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 IV. For composite search, data for the regular version as well as variations with different values of r is reported.

Based on this result, it is clear that exponential and composite search techniques should be preferred to binary search. Since composite search has the better convergence time of the two, it is the best choice. Furthermore, tuning the r parameter to reduce convergence time further has very little adverse impact on performance in terms of number of KA packets sent.

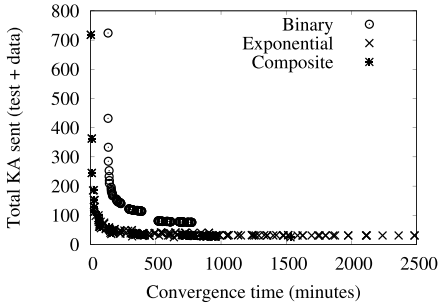

Fig. 6. For each binding timeout, a point is drawn to relate the convergence time to the total number of keep-alive packets sent (data and test connection combined) in a 12 hour run.

The superiority of composite search is demonstrated further in Figure 6. For each technique we plot the total number of KA packets sent (data and test connection combined) in a 12 hour run against the convergence time for each possible binding timeout. A technique which has bulk of the points closer to the origin would be preferable. The points for binary search are further away from the origin, compared to the other ones. For exponential search, significant number of points are away from origin along the X-axis. For composite search, however, bulk of the points form a cluster close to the origin, compared to the other techniques.

E. Impact of Packet Failure

So far, when a KA 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 simulate transient network failures and observe the behavior of the different search techniques. We define a parameter ρ that represents the probability that a KA message will fail. The message failures are independent of each other. For ρ set to 0.02, 0.05 and 0.10, we observe the detected binding timeout. For each value of ρ, we repeated our experiment 20 times and averaged the results over those runs. Figure 7 plots the detected binding timeouts against the actual binding timeouts. The curves for ρ = 0.05 have been left out for brevity. The curve marked as 'No packet failure' 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.

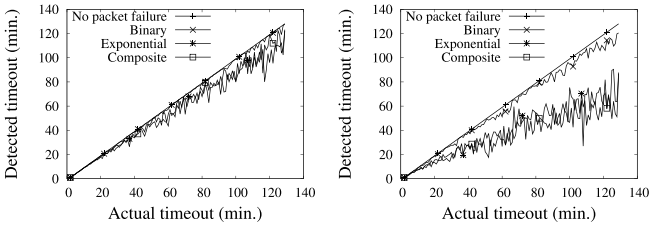

Fig. 7. Detected vs. actual binding timeout in presence of transient packet failure. (a) $\rho = 0.02$. (b) $\rho = 0.10$.

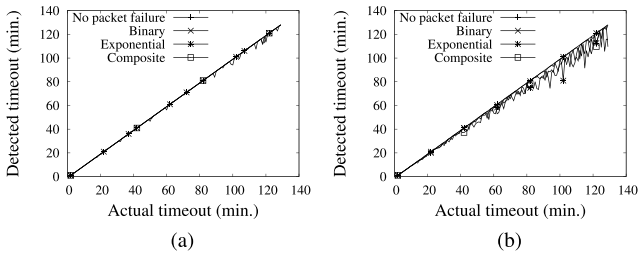

Fig. 8. Detected vs. actual binding timeout when retry scheme is applied in presence of transient packet failure. (a) $\rho = 0.02$. (b) $\rho = 0.10$.

F. Retry on Packet Failure

We modified each algorithm as follows: When KA message fails, we re-establish the connection and test the same interval again. It is unlikely that both packets will encounter the transient failure. If the latter attempt fails too, we conclude that we have overshot the binding timeout. With this modification, we re-ran the same experiments. The results are shown in Figure 8. For $\rho = 0.02$ and $\rho = 0.05$ (plot omitted), the error in the detected binding timeout is very negligible.

For $\rho = 0.10$, 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 KA 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 like to know how well do the different techniques perform in reducing the number of KA 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 ρ was set to 0.02, and q to 0.10. Recall that q is the allowed error in the detected binding timeout, measured as a fraction of actual binding timeout. Parameter r was not used in case of composite search. The result for each binding timeout was averaged over 20 repetitions. The results are shown in Figure 9. These curves are very similar to the corresponding curves of Figure 5. The average percentage reduction in number of KA packets sent is listed in Table V. The values are comparable to the respective values in Table IV.

TABLE V
AVERAGE REDUCTION (%) OF NUMBER OF KEEP-ALIVES SENT WHEN TESTING IS PERFORMED TO IMPROVE THE KEEP-ALIVE INTERVAL FROM THE CONSERVATIVE DEFAULT VALUE. HERE $\rho = 0.02$ 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

Therefore, we can conclude that the retry scheme successfully coped with transient network failures and was able to reduce the number of KA 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 KA packets sent were very much comparable with the earlier experiments. Overall, based on all the experiments, we conclude that composite search should be used to dynamically improve KA interval.

G. Simulation With Notifications

In this experiment we simulated notifications from the server, to observe how the notification delivery success rate is impacted by middle-box binding timeout and the KA interval being used in the device. Parameters ρ, q, r were not used here.

The arrival of notifications is modeled as a Poisson process with a rate of 10, 5, and 1 per hour. Note that the notification arrival rate is an aggregated demand for all applications (because there is one push notification channel for all), so the value may depend on the number of apps installed. The binding timeout was set to 5, 10 and 20 minutes. The KA interval used by the device was set to 4, 16, 24 and 32 minutes. We also simulated the 'Oracle' behavior where the system knows the binding timeout apriori and uses a KA interval that is just slightly smaller than that. Also, instead of using a fixed KA interval, we experimented with adaptive KA interval, using composite search. For each such setting, as described above, we conducted a 24 hour run. Each run was repeated 20 times and the results were averaged over those runs. Figure 10 shows how the notification success rate is impacted for different combinations of binding timeouts and KA intervals used.

The 'Oracle' bar is (obviously) at 100%. The composite search bar matches the oracle bar. On the other hand, whenever a fixed KA interval is larger than the binding timeout, there is certain amount of notifications failure. The larger this gap between KA interval and binding timeout, the larger is the number of notifications lost. Therefore it is clear that, to ensure high success rate of notification delivery, we should either use adaptive KA interval using composite search, or apply a very small fixed KA interval. In the later case, however, the number

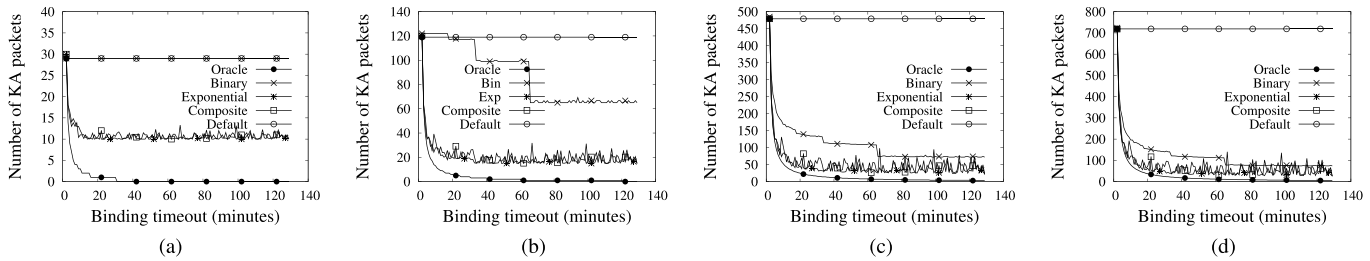

Fig. 9. Number of KA sent during different time durations. ($\rho = 0.02$, $q = 0.10$). (a) 30 minutes. (b) 2 hours. (c) 8 hours. (d) 12 hours.

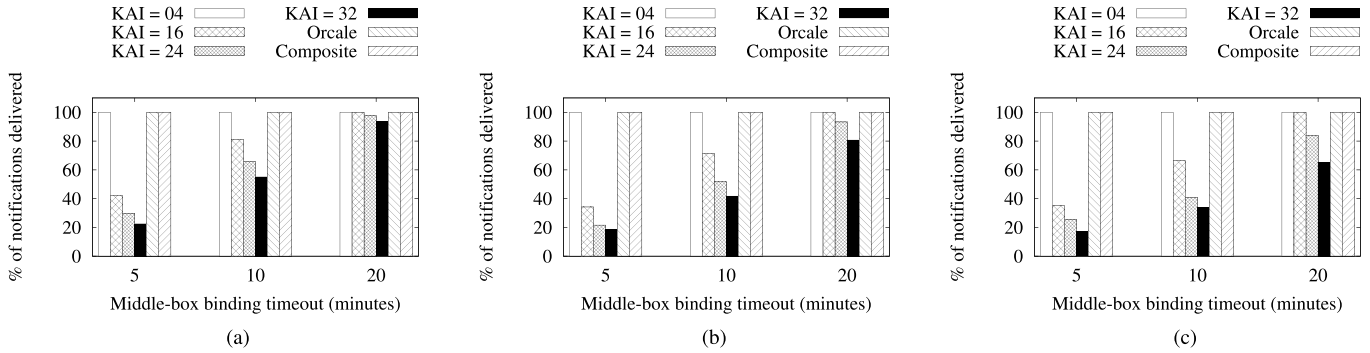

Fig. 10. Comparison of notification delivery success rates in varied settings of keep-alive interval used, middle-box binding timeout and notification rate. (a) Notification rate = 10/hour. (b) Notification rate = 5/hour. (c) Notification rate = 1/hour.

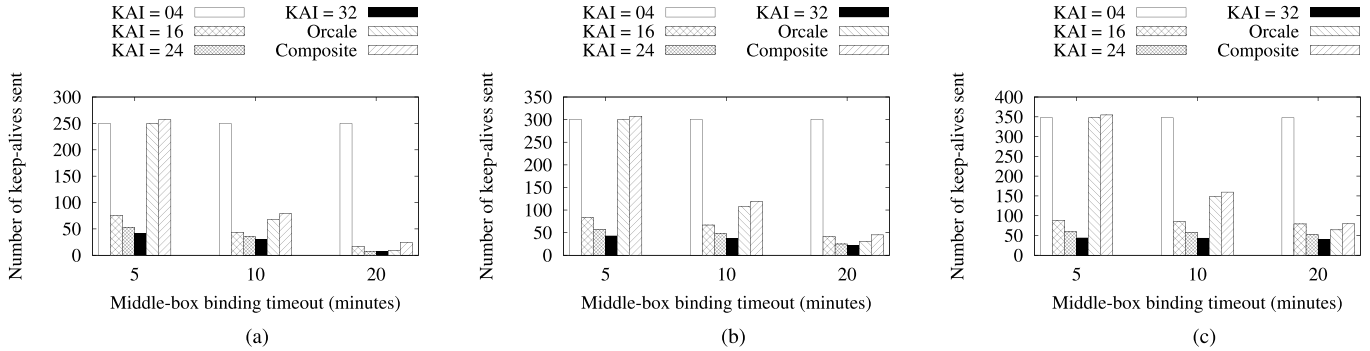

Fig. 11. Comparison of number of keep-alive packets sent in varied settings of keep-alive interval used, middle-box binding timeout and notification rate. (a) Notification rate = 10/hour. (b) Notification rate = 5/hour. (c) Notification rate = 1/hour.

of KA packets sent is very large. This is reflected in Figure 11. For example, when the binding timeout is 10, for ensuring 100% notification delivery, a device using fixed KA interval will send at least twice as much (and as many as 3 times as much) KA packets compared to using adaptive KA interval. And when the binding timeout is 20, this ratio is at least 4 (for low notification rate) and as high as 10 (for high notification rate). Since the number of KA packets sent has direct impact on power consumption of the device, the adaptive KA interval technique clearly turns out to be superior.

H. Real World Implementation

We conducted several real world experiments using the composite search technique. Parameters q, r were not used in these experiments. The components used, experimental setup, results etc. are discussed in this section.

1) Android Application: We implemented the composite search technique in an Android application called *GCMAdaptiveHeartBeater*. While we did not implement the retry scheme discussed in Section V-F, we used Android's *ConnectivityManager* API to detect if the device is connected to internet or not before sending a keep-alive. Also, when a KA failed, we re-ran the test for the current interval if we determined that the device lost connectivity during the operation.

Many user and developer forums complain about message delivery failure or delay when using Google Cloud Messaging (GCM). Particularly [8], discusses this issue and attributes it to heartbeat failures. According to this post, the client component of GCM uses a KA interval of 28 minutes on mobile connections and 15 minutes in WiFi. Indeed, we validated this in Android Gingerbread from log investigation and using the GTalk Service Monitor available in the device. As such, our adaptive scheme will need to adjust this interval as it makes progress through its testing. Our application used techniques discussed in [35] for updating GCM heartbeat interval. The other key components of Android platform that our application uses are *AlarmManager* [36], *Service* [37], *WakefulBroadcastReceiver* [38] etc.

2) Echo Server: We implemented a simple echo server so that GCMAdaptiveHeartBeater could connect to it, send

KA messages at varying intervals and examine whether the connection is maintained. The server operated at port 5228. We chose this port because GCM uses port in the range 5228-5230 [39].

3) Notification Generator: Since we are interested in measuring how notification delivery success rate is impacted by the interval, we must first know how many notifications were sent to a device by different services. However, we do not have a way to find out how many notifications were attempted through GCM for a specific device. As such, instead of using real application notifications, we used our own notification generator to generate different categories of notifications. All the notifications were targeted to GCMAdaptiveHeartBeater. We referred to [40] to build our notification model. The paper listed top 10 applications in terms of notifications generated per day and bucketized the apps into 4 categories: Messenger, Mail, Social and Calendar. The sum of average notifications per day for these categories is 44.9. Note that the study did not include notifications generated by VOIP applications like Skype and Viber. On the other hand, the notifications generated by the applications in the calendar category are mostly expected to be local (reminders) as opposed to notifications received from the cloud. Nonetheless, their reported value is a good starting point to conduct our real world experiment. We modeled the arrival of notifications as a Poisson process with the aforementioned rate. Using this model based notification generator, we pushed notifications through GCM and observed how many were received in our app.

4) Experiment With Firebase Cloud Messaging (FCM): We conducted preliminary experiments on devices running Android Gingerbread, KitKat and Lollipop versions. Gingerbread produced comprehensive amount of logging from which connectivity state (both internet connection as well as connection to the notification service) of the device could be inferred. The higher Android versions did not produce as much logging information. As such, we proceeded with further experiments using a Gingerbread device. We ran composite search in 4 different mobile operator's network of Bangladesh to identify the one that would require the most aggressive KA interval. We identified the operator to be Teletalk and found the binding timeout to be 4 minutes. We conducted 12 hour long runs in this network with Firebase Cloud Messaging (FCM), which is the newer version of GCM. We used fixed keep-alive intervals of $4, 8, 16, 24, 32$ minutes, as well as adaptive KA interval using composite search. In yet another run, we allowed Gingerbread's own heartbeat interval. For each scheme, we correlated the number of KA sent with the notification delivery success rate. As recorded in Table VI, observations made earlier from simulations generally held in practice. However, some notifications were lost due to intermittent network issues even when the heartbeat period was within the binding timeout.

Through log investigation, we realized that in the KAI $= 32$ model heartbeat was being sent every 28 minutes due to the already running KA timer of Gingerbread. Also, after the KA failed and connection was re-established, another KA would get sent within 4 minutes, due to the timer we had set up. We had no way of knowing when the FCM connection gets disconnected and re-established and as such no way to adjust our timer. We felt that we did not have full control over the environment we were experimenting in. Hence, we implemented our own notification service and conducted extensive experiments in that platform.

TABLE VI

COMPARISON OF NUMBER OF KEEP-ALIVES SENT WITH NOTIFICATION DELIVERY SUCCESS RATE IN TELETALK NETWORK USING FCM

KA Scheme	KA Count	Notifications Delivered (%)
Composite	181	100
Gingerbread	22	21
KAI = 4	180	95
KAI = 8	97	63
KAI = 16	46	21
KAI = 24	36	26
KAI = 32	30	21

TABLE VII

BINDING TIMEOUT DETECTED IN VARIOUS NETWORKS

Profile Id.	Internet Service Provider (ISP)	Binding timeout (Minutes)
	Mobile Operators:	
1	Grameenphone [41]	19
2	Robi [42]	9
3	Banglalink [43]	10
4	Teletalk [44]	4
	WiFi network:	
5	TP-Link [45] WiFi router (TL-WR720N), connected to Qubee [46] WiMax	9
6	D-Link [47] WiFi access point (DWL-3200AP), connected to CSE, BUET campus backbone, provided by BTCL [48]	29

5) Experiment With Custom Notification Service: Our custom notification server was bound to port 5229 so that it would be subjected to same type of firewall restrictions (if any) as FCM would in any organization. Notification generator was modified to push notification via this custom server.

We ran our system on Android versions Gingerbread, KitKat and Lollipop in 4 different mobile operators' networks of Bangladesh. We also experimented over WiFi networks, connected to 2 different internet service providers. Like before, we performed 12 hour long runs with different fixed KA intervals as well as using composite search.

From the experiment with adaptive scheme, we were able to identify the binding timeouts. These are recorded in Table VII. For all the experiments, we recorded the notification delivery success rate and total number of KAs attempted. For the runs in Teletalk, we collected server and client logs, correlated them and investigated them thoroughly. All the events in the real experiment followed our expectations.

Figure 12 shows the notification delivery success rates. The observations made earlier from simulations generally held in practice, with some anomalies. For example, adaptive scheme failed to achieve 100% delivery rate in Qubee and Banglalink. In both cases, logs indicated that connectivity issues resulted in the device remaining disconnected for some time, during which notification was sent from the server (determined by correlating server log). Also, sometimes the ConnectivityManager API indicated connected state, yet existing TCP

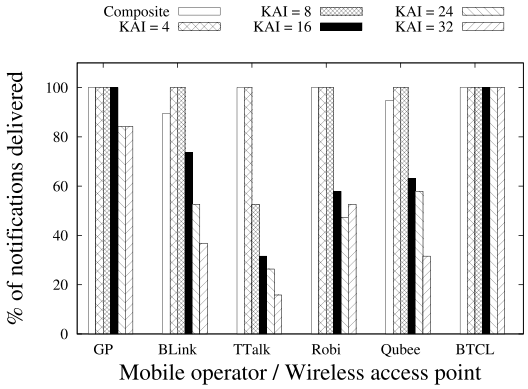

Fig. 12. Notification delivery success rates in real world scenario.

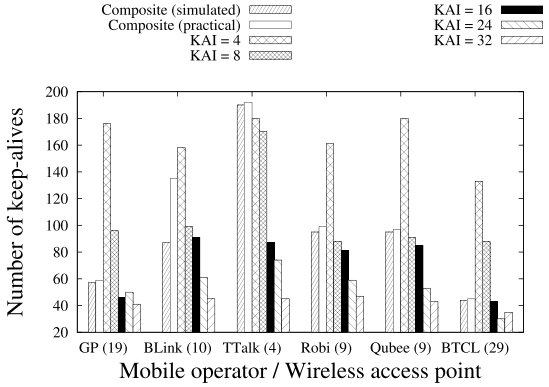

Fig. 13. Total number of keep-alive packets attempted in real world scenario. Each network is annotated with its detected binding timeout.

connections (connection to our notification service, as well as GCM connection) were lost.

We incremented the attempted KA count by 1 each time a KA was sent successfully. When a KA failed, a connection attempt followed immediately if the device had connectivity. Since there are 2 network transactions in this case, we incremented the count by 2. Figure 13 compares the total number of KAs attempted in the different approaches. In addition, it shows the KA count for composite search in a simulated environment (where there are no network issues). In all networks other than Teletalk, KA count in composite search is smaller than the fixed rate KA scheme with 4 minutes interval. Longer KA intervals result in lesser number of KAs, but lose notifications. In the Banglalink case, practical KA count for composite search is way higher than the simulated count. This indicates that the device encountered significant connectivity issues during the experiment, which resulted in many failed KA attempts followed by successful ones. As such the KA count became much higher than expected.

Another observation from the Banglalink experiments is worth noting. From log investigation, we found that frequent KA failures were experienced in this network. Even a 5 second delay beyond the 10 minutes in sending the KA would result in a broken connection. Since such delays are likely to occur in practice, we recommend using a KA interval slightly less (e.g., 10 seconds) than the detected one.

6) Experiment With PowerTutor: To examine impact of KA schemes on energy consumption, we conducted some experiments using PowerTutor [49], an Android application for power consumption monitoring.

TABLE VIII
ENERGY CONSUMPTION IN DIFFERENT KA SCHEMES

Keep-alive Scheme	Energy spent (mWh)	Notifications Delivered (%)	Keep-alive count
Gingerbread	15.59	53	-
Composite	16.54	89	86
KAI = 4	27.73	95	180

In Banglalink network, we conducted 12 hour runs on a Gingerbread device using FCM. We used composite, 4 minute fixed KA and Gingerbread's native KA models. In each case we collected power measurement traces from PowerTutor and recorded the energy consumption, excluding the display. The device we used had a 3.7 V Li-ion battery with 6.48 Wh capacity. Before each run, the battery was charged to full capacity. To account energy expenditure solely for KA activities, we purposefully avoided interactions with the phone (e.g. calls, browsing etc.). Only the system's background processes and our notification receiver app was running. This gave us fairly the same environment across all competing schemes. The results are recorded in Table VIII. Composite search resulted in slightly higher energy cost compared to Gingerbread's native KA scheme. However, the notification delivery success was improved significantly. Additionally, the energy cost was 67% less than when 4 minutes fixed interval is used.

7) Experiment With Newer Android Versions and FCM: As mentioned earlier, we conducted experiments in FCM using Android Gingerbread version because of the comprehensive logging that was available – we could analyze the logs and correlate theoretical as well as simulation results with the practical observations. However, Gingerbread is relatively old version of Android. Therefore, we decided to run experiments using KitKat and Lollipop as well, which together covers almost half of Android's market share [50]. Interestingly, with Android's native KA scheme in these versions, we were not losing as many notifications as we did in the experiments with Gingerbread. This suggested that some sort of adaptive scheme has been introduced in the newer versions of Android.

To better understand the scheme and make a comparison with ours, we intercepted all traffic coming and going out of an Android Lollipop device and looked for packets where the remote port was between 5228 and 5230 and the remote machine's IP address belonged to publicly known Google's IP blocks. Even though the FCM connection uses encryption, we were able to identify which TCP packets were due to KAs, by examining the packet size and period.

We conducted several 12 hour runs in the connection profile 5 of Table VII and observed that Lollipop indeed applies adaptive adjustment of KA intervals that attempts to catch up the binding timeout of the edge routers. One difference from our scheme is that the server periodically pushes information to the client, based on which the client adjusts its subsequent KA interval. The technique raises KA interval by a small amount after the current interval has worked for a certain number of times. On the other hand, it falls back to a smaller

TABLE IX
COMPARISON OF KA COUNT AND NOTIFICATIONS DELIVERY SUCCESS BETWEEN LOLLIPOP'S NATIVE KA SCHEME AND COMPOSITE SEARCH

Keep-alive Scheme	Notifications Delivered (%)	Keep-alive count
Lollipop (1st run)	84	99
Composite (1st run)	95	97
Lollipop (Subsequent run)	84	74
Composite (Subsequent run)	100	82

interval when the current interval fails twice in a row. The attempts to improve KA interval happen on the notification channel itself which is a contrast to our use of a separate test connection. As such, overshooting the binding timeout results in disconnection for short periods from time to time. For a detailed analysis, the reader is referred to Appendix 2.

It is not possible to do an energy consumption comparison between Lollipop's native adaptive scheme and composite search. This is because, in order to conduct the composite search on Lollipop, we would first have to disable its native KA scheme. We have not found a way to achieve that. However, we can compare the KA counts and notification delivery rate of Lollipop's native KA scheme to those of the composite search technique, as observed from the experiment with the custom notification service. As can be seen from Table IX, the energy cost in both schemes is similar (inferred from KA count) in the first run scenario behind an edge router. In the subsequent run on the same network, both schemes send less number of KA due to the cached improved KA interval. Lollipop has a slight edge over our scheme in this run. This is because our scheme only used minute boundaries for KA intervals whereas Lollipop was more granular. While our scheme used 9 minutes as the optimal KA interval, Lollipop converged to 10 seconds higher than that. From the perspective of notification delivery, our scheme was significantly superior in both scenarios. Even in the subsequent run scenario, Lollipop lost notifications because it periodically tried to increase the KA interval and got disconnected from FCM. The composite search, on the other hand, did not attempt any improvements after converging to the optimal interval and was able to receive *all* the notifications.

8) Code Availability: Code for GCMAdaptiveHeartBeater is available at https://github.com/srautonu/GCMAdaptiveHeartBeater. Notification generator, echo server and the notification server codes are available at https://github.com/srautonu/AdaptiveHeartBeat. These can be used by other researchers to conduct further experiments in different mobile network in different countries.

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, exponential 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. We subsequently implemented it in an Android application and performed further real world experiments.

Several new research directions can be proposed from this work. For example, once the binding timeout is detected, the notification channel sends KA using the optimal interval. Occasionally, it is possible that due to changes in the network infrastructure, the binding timeout has decreased. Then the notification channel will experience frequent disconnections. A strategy should be developed to bring the channel out of this unstable state. The most conservative approach would be to revert back to the default short KA interval and restart testing for better intervals. Another option is to remain cautiously optimistic: reduce the KA interval to 50% of the current value and restart testing. The former ensures stability of the connection right away but incurs more power cost. The latter tries to restrict the increase in power consumption but may fail to quickly stabilize the connection. Work should be done to analyze these approaches and run simulations to compare their behaviors.

Another future work could be to develop other adaptive KA strategies. Randomized techniques (e.g., a guided random walk) can be explored. Iterative probing with multiple test connections can be pursued to reduce the convergence time significantly and to improve the KA intervals quickly. How many connections should be run in parallel? While more connections will reduce convergence time, they would also increase load on the server. Another question is whether all the parallel connections use the same technique (i.e., composite search) or not. These questions make the idea of multiple test connections an independent research direction.

Finally the tools we provided can be used to conduct further experiments in mobile networks of different countries. Recording the wide variation of binding timeouts in a large scale will help in developing robust solutions to improve user experience in 'always connected' scenarios.

REFERENCES

[1] I. Warren, A. Meads, S. Srirama, T. Weerasinghe, and C. Paniagua, "Push notification mechanisms for pervasive smartphone applications," *IEEE Pervasive Comput.*, vol. 13, no. 2, pp. 61–71, Apr./Jun. 2014.
[2] Viber. Accessed on Mar. 25, 2016. [Online]. Available: https://www.viber.com/en/
[3] *Google Cloud Messaging: Overview*. Accessed on Apr. 1, 2016. [Online]. Available: https://developers.google.com/cloud-messaging/gcm
[4] S. Hätönen, A. Nyrhinen, L. Eggert, S. Strowes, P. Sarolahti, and M. Kojo, "An experimental study of home gateway characteristics," in *Proc. 10th ACM SIGCOMM Conf. Internet Meas.*, 2010, pp. 260–266.
[5] R. Braden, *Requirements for Internet Hosts-Communication Layers*, Standard RFC 1122, Oct. 1989.
[6] H. Haverinen, J. Siren, and P. Eronen, "Energy consumption of always-on applications in WCDMA networks," in *Proc. IEEE Veh. Technol. Conf.*, Apr. 2007, pp. 964–968.
[7] *GCM Heartbeat Manager Source Code*. Accessed on Mar. 30, 2016. [Online]. Available: https://chromium.googlesource.com/chromium/chromium/+/trunk/google_apis/gcm/engine/heartbeat_manager.cc
[8] *Push Notifications Delayed, Hearbeat Interval not Reliable*. Accessed on Nov. 29, 2016. [Online]. Available: https://productforums.google.com/forum/#!topic/nexus/fslYqYrULto
[9] M. S. Rahman, Y. S. Uddin, M. S. Rahman, and M. Kaykobad, "Using adaptive heartbeat rate on long-lived TCP connections," in *Proc. Int. Conf. Netw. Syst. Secur. (NSysS)*, 2016, pp. 16–24, doi: 10.1109/NSysS.2016.7400700.

[10] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 13–24, 2012.
[11] S. Guha, K. Biswas, B. Ford, S. Sivakumar, and P. Srisuresh, *NAT Behavioral Requirements for TCP*, Standard RFC 5382, Oct. 2008.
[12] Accessed on Mar. 26, 2016. [Online]. Available: https://azure.microsoft.com/en-us/documentation/articles/load-balancer-tcp-idle-timeout/
[13] M. Stiemerling, E. Davies, C. Aoun, and H. Tschofenig, *NAT/Firewall NSIS Signaling Layer Protocol (NSLP)*, Standard RFC 5973, Oct. 2010.
[14] M. Stiemerling, J. Quittek, and C. Cadar, *NEC's Simple Middlebox Configuration (SIMCO) Protocol Version 3.0*, Standard RFC 4540, May 2006.
[15] *Understanding Direct Push*. Accessed on Apr. 1, 2016. [Online]. Available: http://technet.microsoft.com/en-us/library/aa997252(EXCHG.80).aspx
[16] *Heartbeat Interval Adjustment*. Accessed on Apr. 1, 2016. [Online]. Available: http://technet.microsoft.com/en-us/library/cc182270.aspx
[17] *Apple Push Notification Service*. Accessed on Apr. 1, 2016. [Online]. Available: https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html
[18] *Push Notifications (Windows Phone)*. Accessed on Apr. 1, 2016. [Online] Available: https://msdn.microsoft.com/en-us/library/hh221549.aspx
[19] *Windows Push Notification Services (WNS) Overview (Windows Runtime Apps)*. Accessed on Apr. 1, 2016. [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/apps/hh913756.aspx
[20] S. R. Gatta *et al.*, "Keep alive management," U.S. Patent 8 892 710 B2, Nov. 18, 2014.
[21] S. Herzog *et al.*, "Determining an efficient keep-alive interval for a network connection," U.S. Patent 8 375 134 B2, Feb. 12, 2013.
[22] R. Seggelmann, M. Tuexen, and M. Williams, *Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension*, Standard RFC 6520, Feb. 2012.
[23] T. Dierks and E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.2*, Standard RFC 5246, Aug. 2008.
[24] Z. Li, K. Bian, T. Zhao, and X. Li, "Improving the QoE of mobile data access for long-lived connections in cellular networks," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, May 2015, pp. 456–461.
[25] N. Vallina-Rodriguez and J. Crowcroft, "Energy management techniques in modern mobile handsets," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 1, pp. 179–198, Jan. 2013.
[26] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: A measurement study and implications for network applications," in *Proc. 9th ACM SIGCOMM Conf. Internet Meas. Conf.*, 2009, pp. 280–293.
[27] M. Jain and C. Dovrolis, "End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, pp. 295–308, 2002.
[28] A. Akella, S. Seshan, and A. Shaikh, "An empirical evaluation of wide-area Internet bottlenecks," in *Proc. 3rd ACM SIGCOMM Conf. Internet Meas.*, 2003, pp. 101–114.
[29] N. Hu and P. Steenkiste, "Evaluation and characterization of available bandwidth probing techniques," *IEEE J. Sel. Areas Commun.*, vol. 21, no. 6, pp. 879–894, Aug. 2003.
[30] B. Melander, M. Bjorkman, and P. Gunningberg, "A new end-to-end probing and analysis method for estimating bandwidth bottlenecks," in *Proc. Global Telecommun. Conf. (GLOBECOM)*, vol. 1. Dec. 2000, pp. 415–420.
[31] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell, "PathChirp: Efficient available bandwidth estimation for network paths," in *Proc. Passive Active Meas. (PAM) Workshop*, Apr. 2003.
[32] M. Jain and C. Dovrolis, "End-to-end estimation of the available bandwidth variation range," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 33, no. 1, pp. 265–276, 2005.
[33] R. Price and P. Tino, "Adapting to NAT timeout values in P2P overlay networks," in *Proc. IEEE Int. Symp. Parallel Distrib. Process., Workshops Phd Forum (IPDPSW)*, Apr. 2010, pp. 1–6.
[34] *Omnet++*. Accessed on Apr. 1, 2016. [Online]. Available: http://www.omnetpp.org/
[35] *GCM: How to Send Heartbeat to GCM Server*. Accessed on Nov. 29, 2016. [Online]. Available: http://stackoverflow.com/questions/27471141/gcm-how-to-send-heartbeat-to-gcm-server
[36] *AlarmManager*. Accessed on Nov. 29, 2016. [Online]. Available: https://developer.android.com/reference/android/app/AlarmManager.html
[37] *Creating a Background Service*. Accessed on Nov. 29, 2016. [Online]. Available: https://developer.android.com/training/run-background-service/create-service.html
[38] *WakefulBroadcastReceiver*. Accessed on Nov. 29, 2016. [Online]. Available: https://developer.android.com/reference/android/support/v4/content/WakefulBroadcastReceiver.html
[39] *About Firebase Cloud Messaging Server*. Accessed on Nov. 29, 2016. [Online]. Available: https://firebase.google.com/docs/cloud-messaging/server
[40] A. Sahami Shirazi *et al.*, "Large-scale assessment of mobile notifications," in *Proc. SIGCHI Conf. Human Factors Computing Syst.*, 2014, pp. 3055–3064.
[41] Grameenphone. Accessed on Nov. 28, 2016. [Online]. Available: https://www.grameenphone.com/
[42] Robi. Accessed on Mar. 21, 2016. [Online]. Available: http://www.robi.com.bd/
[43] Banglalink. Accessed on Mar. 21, 2016. [Online]. Available: http://www.banglalink.com.bd/en/
[44] Teletalk. Accessed on Nov. 28, 2016. [Online]. Available: http://www.teletalk.com.bd/
[45] TP-Link. Accessed on Nov. 28, 2016. [Online]. Available: http://www.tp-link.com/en/
[46] Qubee. Accessed on Mar. 21, 2016. [Online]. Available: http://www.qubee.com.bd/
[47] D-Link. Accessed on Nov. 28, 2016. [Online]. Available: http://us.dlink.com/
[48] BTCL. Accessed on Mar. 21, 2016. [Online]. Available: http://www.btcl.net.bd/
[49] *PowerTutor*. Accessed on Mar. 12, 2017. [Online]. Available: http://ziyang.eecs.umich.edu/projects/powertutor/index.html
[50] *Android Dashboards*. Accessed on Jun. 29, 2017. [Online]. Available: https://developer.android.com/about/dashboards/index.html

M. Saifur Rahman received the bachelor's and master's degrees in computer science and engineering from the Bangladesh University of Engineering and Technology (BUET) in 2004 and 2014, respectively. He is currently an Assistant Professor with the Department of Computer Science and Engineering, BUET. His research interest includes distributed computing systems, networking, algorithms, and bioinformatics. He is a member of the ACM.

Md. Yusuf Sarwar Uddin (M'16) received the bachelor's and master's degrees in computer science and engineering from the Bangladesh University of Engineering and Technology (BUET) in 2004 and 2006, respectively, and the Ph.D. degree from the Department of Computer Science, University of Illinois at Urbana–Champaign, USA, in 2012. He is currently an Associate Professor with the Department of Computer Science and Engineering, BUET. His research interests include distributed computing systems, mobile computing, wireless and sensor networks, and distributed cyber-physical systems. He is a member of the ACM.

Tahmid Hasan is currently a sophomore with the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology. He is currently interested in statistical modeling and computational aspects of statistics.

M. Sohel Rahman (SM'16) received the bachelor's and master's degrees in computer science and engineering from the Bangladesh University of Engineering and Technology (BUET) in 2002 and 2004, respectively, and the Ph.D. degree from King's College London in 2008. He is currently a Professor with the Department of Computer Science and Engineering, BUET. He is a senior member of the ACM, and a member of the London Mathematical Society, the American Mathematical Society, and the Computability in Europe.

M. Kaykobad received the M.S. degree (Hons.) in engineering from Odessa State Maritime University in 1979, the M.S. degree in engineering from the Asian Institute of Technology, and the Ph.D. degree from the Flinders University of South Australia in 1988. He is currently a Professor with the Department of Computer Science and Engineering and the Dean of the EEE Faculty of the Bangladesh University of Engineering and Technology. His research interest is in the fields of algorithms and optimization. During his long involvement with the ACM ICPC, he was adjudged outstanding coach in 2002 with ICPC World Finals held at Honolulu. He also received the Senior Coach Award at the ICPC World Finals held at St. Petersburg in 2014. He was awarded the Gold Medal for excellence in research from the Bangladesh Academy of Sciences.