M.Sc. Engg. Thesis

On Approaches to Detect Optimal Keep-alive Interval of TCP Connections

by
Mohammad Saifur Rahman

Submitted to

Department of Computer Science and Engineering
in partial fulfilment of the requirments for the degree of
Master of Science in Computer Science and Engineering

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology (BUET)
Dhaka 1000

May 2014

The thesis titled "**On Approaches to Detect Optimal Keep-alive Interval of TCP Connections**", submitted by Mohammad Saifur Rahman, Roll No. 0412052070P, Session April 2012, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering and approved as to its style and contents. Examination held on May 14, 2014.

Board of Examiners

1. _____

Dr. Md. Yusuf Sarwar Uddin — Chairman (Supervisor)
Assistant Professor
Department of CSE
BUET, Dhaka.

2. _____

Dr. Mohammad Mahfuzul Islam — Member (Ex-officio)
Professor & Head
Department of CSE
BUET, Dhaka.

3. _____

Dr. A.K.M. Ashikur Rahman — Member
Associate Professor
Department of CSE
BUET, Dhaka.

4. _____

Dr. S. M. Farhad — Member
Assistant Professor
Department of CSE
BUET, Dhaka.

5. _____

Dr. Mohammad Nurul Huda — Member (External)
Professor
Department of CSE
United International University, Dhaka.

Candidate's Declaration

It is hereby declared that this thesis or any part of it has not been submitted elsewhere for the award of any degree or diploma.

Mohammad Saifur Rahman
Candidate

Contents

List of Figures

List of Tables

Acknowledgments

In the name of Allah, the most benevolent and the most merciful.

All praises and thanks are due to Him.

I express my heart-felt gratitude to my supervisor, Dr. Md. Yusuf Sarwar Uddin, for his constant supervision of this work. He is an awesome teacher, guide and a friend. He injected an amazing amount of energy and enthusiasm in this research work. Bouncing ideas with him and white-boarding in his office were memorable and fun experiences.

I would also want to thank the members of my thesis committee for their valuable suggestions. I thank Professor Dr. Mohammad Mahfuzul Islam, Dr. A. K. M. Ashikur Rahman, Dr. S. M. Farhad and specially the external member Professor Dr. Mohammad Nurul Huda.

Besides my supervisor and board members, I am specially thankful to my elder brother, Professor Dr. M. Sohel Rahman. He has acted as an in-house supervisor throughout every stage of my studies and career. I also thank Professor Dr. M. Kaykobad for being such an inspiration.

In this occasion, I commemorate my late mother. I missed her guidance and advices during my M. Sc. studies. I acknowledge my father, my wife Tasneem, our two sons Abrar and Ahmad, my sister in law Sara and my sweet niece Nazaha, for letting me time off from my social and family responsibilities to focus on my thesis. I also acknowledge Paresa and Keya for preparing great meals and snacks for me.

Abstract

When a TCP connection between a client behind a NAT 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, it is preferred that the keep-alive packet is sent at the farthest possible time within the NAT binding timeout. This interval is called the *Optimal Keep-alive Interval*. Due to varied settings of different network equipments, optimal keep-alive (KA) interval will not be identical in different networks. Hence, it needs to be dynamically detected. In this thesis, we employ several search approaches to dynamically detect the optimal KA interval. These include binary search, exponential search and hybrid search. Hybrid search combines different aspects of binary and exponential search techniques. We present theoretical analysis of different aspects of these techniques. We also conduct simulation based experiments to compare these techniques. Based on the theoretical studies and the experimental results, we conclude that hybrid search should be used in detecting the optimal keep-alive interval of a TCP connection.

Chapter 1

Introduction

Recently, power consumption of mobile devices has received significant research attention [18, 28, 30, 37]. Devices such as smart phones, tablet PCs and other customized personal digital assistants (PDA) try to provide the user with fresh data. This includes the user's emails, social news feed, corporate recent activity logs etc. Since the devices in question have limited battery life, they cannot frequently poll for information updates. Rather, they rely on change notifications being pushed by a notification server. The device remains connected to the server even during low power modes, typically via a Transmission Control Protocol (TCP) connection. 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) or any other 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 has passed through it for some specified amount of time. Studies have shown that the NAT binding timeouts vary dramatically amongst the commercially available home gateways [22]. 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 [17] 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 or KA interval. Such an interval needs to be computed via experiment in the network environment that the device is currently in.

In this thesis, we present several iterative probing approaches to dynamically detect the optimal keep-alive interval. We compare these techniques by analyzing several quantitative aspects. We also conduct simulation based experiments to validate the theoretical results as well as analyze the behavior of these techniques further. Based on the analysis, we identify the technique that has superior performance.

1.1 Motivation

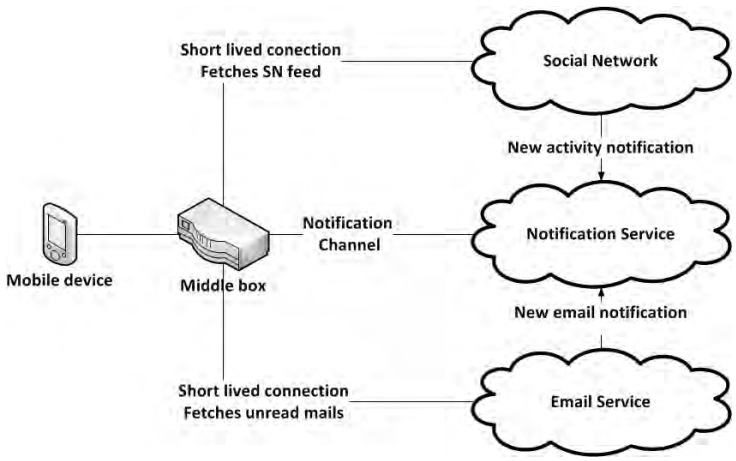


Figure 1.1: A model of notification service used in different mobile platforms.

As mentioned above, the motivation for this work is drawn from the scenarios enabled by the mobile platforms of today. iPhone, Android, Windows Phone, Black Berry provide the user a feeling that his mobile device is always on – fetching emails as soon as they arrive in the email server; updating the social news feeds as soon as there are activities in his social network. The same goes for tablet platforms like iPad, Android, Windows 8 and so on.

Each of these platforms provide a *Notification Service* which, at a high level, can

be modeled as shown in Figure 1.1. 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 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. In the same way, the device receives emails as soon as they arrive to the email service. Therefore, to enable these seemingly *always connected* scenarios, the device must keep the notification channel alive. This channel remains idle most of the time; only getting notification messages from the server when something happens. Therefore, periodic keep-alive messages must be sent through the notification channel to keep its state active in the middle-box.

Without better information, the keep-alives would be sent at a very conservative interval (e.g. 1 minute). Even when the device is in idle state, it needs to bring the CPU and the radio module to high power state to send the keep-alive every minute. This has a significant power consumption [23]. As such, it is important to use a higher interval. On the other hand, the binding timeout of middle-boxes varies widely [22]. Therefore, we cannot just guess a higher interval and hope it to be able to keep the notification channel alive. It is extremely important to keep this channel active all the time. Otherwise, we will miss notifications from the server and will not be able to fetch fresh data from different services. This motivates the need to detect the optimal keep-alive interval dynamically in the network environment that the device is in. This is done by testing higher keep-alive intervals in a separate connection.

1.2 Problem Statement

When a TCP connection is established through a stateful middle-box, it may get dropped by the middle-box if no data has passed through the connection for a long time. To

ensure that the state binding for the connection is not timed out, periodic heartbeat or keep-alive messages need to be sent. This has some impact on power consumption. As such, 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 binding timeout. In the optimal keep-alive interval detection problem, we try to find out this optimal interval between successive keep-alives. This is done by keeping the connection idle for different amount of times and then probing whether the connection is still open.

If an interval being tested overshoots the optimal KA interval, the middle-box clears the state of the connection. In such a case, we will be unable to receive data from the server for some amount of time. To overcome this issue, the testing is performed by opening a second connection to the server. The original TCP connection is used for sending and receiving important data between the client and the server. This is referred to as the *Data Connection*. The second connection, opened to perform the testing to detect optimal KA interval, is called the *Test Connection*.

Before testing starts, keep-alives over the data connection are sent with a conservative (small) interval so that the connection is not lost. As testing reveals better intervals, the data connection's keep-alive period is increased accordingly. We need to find a optimal keep-alive interval detection technique that would result in minimum number of keep-alive messages sent over a given amount of time.

1.3 Related Work

In this section we review some literature that is relevant to our 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 the optimal keep-alive interval. Finally, we review some earlier works that uses iterative probing for measuring different network parameters.

1.3.1 Study of NAT binding timeouts

In [22], Hätönen et al. experimentally analyzed characteristics of a substantial number of different home gateways. These characteristics include binding timeouts, queuing delays, throughput, protocol support etc.

One of the experiments in their work focus on finding the TCP binding timeouts of different home gateways. They used a modified binary search algorithm in their test bed to detect the binding timeout for each home gateway they tested. To speed up the test, multiple parallel connections were used. Testing was terminated if binding timeout of a device was longer than 24 hours. The results show that TCP binding timeout varies dramatically amongst the commercially available home gateways. The shortest timeout observed was about 4 minutes, while the median timeout was about an hour. More than 50% of the devices did not meet the IETF recommended timeout of 124 min [21]. 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 the optimal keep-alive interval and send keep-alives using that period. While protocols such as NSIS [35] or SIMCO [36] do exist to explicitly negotiate the binding timeout with the middle-box, equipments currently used by mobile operators do not usually support these protocols.

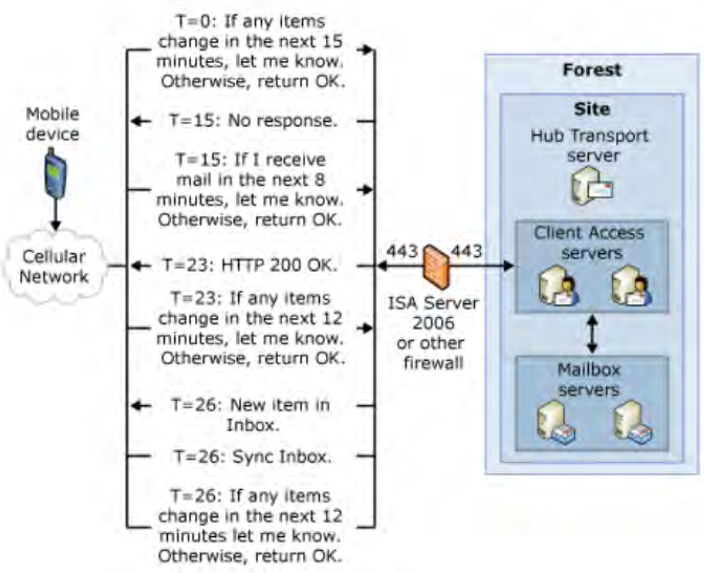


Figure 1.2: (Excerpted from [13]) Direct Push Network Design.

1.3.2 Keep-alive in Existing Systems

In this section, we review use keep-alive messages in some existing systems. We look at the Exchange Active Sync (EAS) protocol first. The Direct Push feature of EAS protocol is designed to keep a mobile device up to date over a cellular network connection [13]. It uses long-standing HTTPS requests to maintain a channel with the service. 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. Figure 1.2, excerpted from [13] shows an example run. Detailed discussion on the keep-alive interval adjustment, along with the configurable parameters, can be found in [6].

Windows Phone maintains a connection with Microsoft Push Notification Service (MPNS) for receiving notifications [11]. Windows 8 devices connect to Windows Notification Service (WNS) for the same purpose [14]. Similarly, Apple Push Notification Service (APNs) [3] serves iPad, iPhone while Google Cloud Messaging (GCM) serves An-

droid devices [5]. In all these cases a persistent connection must be maintained between the client (device) and the server (notification service). As such, 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. The details of the exact mechanism used are not available publicly. However, source code posted at [1] and discussion threads in Google product forums [2] indicate that GCM uses 15 minutes over WiFi network and 28 minutes over mobile or unknown network as the keep-alive interval. The forum threads further indicate that these intervals result in frequent disconnects.

Patent applications filed by 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 [20, 24]. We can infer, therefore, that MPNS and WNS clients test different keep-alive intervals through a test connection. However, the exact strategy used in choosing the test intervals is not called out in the patent applications.

The Transport Layer Security (TLS) protocol also uses keep-alive or heartbeat messages. TLS allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery [19]. Originally, there was no feature in the protocol to keep the connection alive without continuous data transfer. This limitation was overcome by the recently added Heartbeat Extension [33] – passing heartbeat messages for ensuring liveness of peers. The user can use the new *HeartbeatRequest* message, which has to be answered by the peer with a *HeartbeartResponse* immediately. Incidentally, at the time of writing this thesis, a missing bounds check in the OpenSSL library's implementation of heartbeat extension was exposed and received widespread attention. The bug was termed *Heartbleed* [7, 12].

1.3.3 Impact of KA Interval on Power Consumption

Haverinen et al. showed in [23] that battery life is significantly influenced by the frequency of keep-alive messages. They performed real power measurements in two different

Table 1.1: (Excerpted from [23]) Current consumption by keep-alive interval.

Interval [s]	**Avg. current 2G [mA]**	**Avg. current 3G [mA]**
20	29	34
40	16	24
150	9.1	16
300	7.3	14
Infinite	5.2	6.1

3G networks. They also conducted measurements in 2G GPRS network. From these measurements, they calculated the average energy consumption of a single keep-alive event. The consumption of a single keep-alive transaction varied between 0.15 mAh and 0.6 mAh in 3G, and between 0.11 mAh and 0.13 mAh in 2G. Table 1.1, excerpted from [23] shows how current consumption is reduced with increased keep-alive interval. This is observed in both 2G and 3G networks.

A comprehensive survey on general solutions for energy efficiency on mobile devices, published between 1999 and May 2011, is available in [38]. Here the authors classify and provide a short summary of the various efforts on studying, modeling and reducing energy consumption in mobile devices. One of the power models, proposed by Balasubramanian et al. [16], is briefly described in Section 2.4. This model too indicates that power consumption can be reduced by increasing keep-alive interval.

1.3.4 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 is initially set to at most four segments. For each segment that gets acknowledged, the sender adds one segments worth of bytes to the congestion window. Therefore, congestion window doubles up in each round trip iteration. This is called slow start and continues until a slow start threshold

is reached. Subsequently, TCP switches from slow start to additive increase. In this mode, the congestion window 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 [39].

When measuring avail-bw using iterative probing, several probing streams are used. For each stream, the input and output rates are compared. If the output rate is found smaller than the input rate, then the probing rate is larger than the avail-bw during the probing stream. The probing rate is varied either linearly or based on what happened in previous streams, until the probing process converges to an estimate of the average avail-bw. Jain et al. used iterative probing to measure end-to-end avail-bw. They termed their measurement methodology as Self-Loading Periodic Streams (SLoPS) [26]. They also implemented it in a tool called pathload and examined the relation between avail-bw and TCP throughput. Other iterative techniques for avail-bw measurements include Bfind [15], PTR [25], TOPP [29], pathChirp [32] etc.

The variability in the available bandwidth has also been measured by Jain et al. using iterative probing [27]. A number of N probing streams of duration τ and rate R are sent through a path. Each stream provides an indication of whether the avail-bw in the corresponding time interval is higher than R. The resulting N binary samples are used to estimate the precentile of the avail-bw distribution.

1.4 Our Contribution

The main task of this thesis is to design techniques to detect the optimal keep-alive interval of a TCP connection. We employ several search approaches to dynamically detect the optimal KA interval. These include binary search, exponential search and hybrid search. Hybrid search combines different aspects of binary and exponential search techniques.

We analyze the proposed techniques theoretically and validate them through simu-

lation. We make simulation based experiments using Omnet++ [8]. The results of the experiments, their explanations and conclusions drawn from them are stated in Chapter 4. We evaluate the performance of our techniques by varying various parameters (application level packet failure rate, acceptable error in the detected keep-alive interval etc.). Experiments reveal that hybrid and exponential search techniques result in the least number of keep-alive messages sent over the data connection during several study durations of different lengths. Additionally, the number of probe counts in the test connection and the time taken to detect the optimal KA interval are less in hybrid search than in exponential search. This is proved through theoretical analysis and is backed up by the simulation results. Therefore, we conclude that hybrid search should be used in detecting the optimal keep-alive interval of a TCP connection.

Implementing the KA interval detection techniques on a real mobile platform is left as future work. All observations are made based on theoretical analysis and simulations.

1.5 Assumptions

We made the following assumptions for the scope of work in this thesis. However, in Chapter 5, we have outlined how these assumptions could be removed.

1. We assume that once a keep-alive interval is detected by a device in a network, it does not change. This assumption should hold in the real world most of the time. However, If the middle-box is replaced or re-configured, then the binding timeout may become different. But, we do not expect the middle-boxes to be re-configured frequently. As such, we make the stated assumption.

2. The keep-alive interval detection will be triggered whenever a device detects that it is in a new network. We assume that a device can accurately determine whether it is connected to a new network or not. A device can look at the Access Point Name (APN) of the mobile network for this detection. In case of WiFi network, the access point's

Service Set Identifier (SSID) could be used. However, it is possible that 2 different access points in different locations have been configured with the same SSID. The APN could also be duplicated due to misconfiguration. In such cases, the assumption does not hold. In Section 5.2 we outline a mitigation for this situation.

1.6 Organization of the Thesis

The rest of the chapters are organized as follows. Chapter 2 gives preliminary concepts and terminologies that will be used in describing the contribution of this research in subsequent chapters. Chapter 3 contains the core work of this thesis. It introduces several techniques to detect the optimal keep-alive interval of TCP connection. Theoretical analysis and comparison among these techniques are also provided in this chapter. Chapter 4 contains setup of the simulation experiments and their results to further compare the performance of these techniques. Chapter 5 concludes this dissertation by summarizing the major contributions and providing some directions for further research in this field.

Chapter 2

Preliminaries

In this chapter, we review some preliminary concepts and terminologies that will be used in describing the contribution of this research in subsequent chapters.

2.1 Middle-box

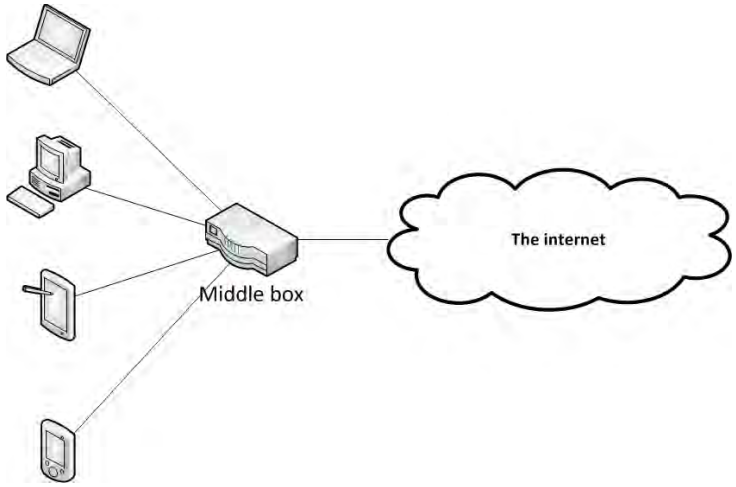


Figure 2.1: A middle-box connecting a PC, a laptop, a PDA and a mobile phone to the internet.

Computers in a local network connect to the internet through stateful components like firewalls, Network Address Translators (NATs) and proxy servers. 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. Figure 2.1 shows an example configuration. The figure is relevant for any type of middle-box.

2.1.1 Network Address Translator (NAT)

Network Address Translator (NAT) is a widely used component in the IPv4 based internet. In a typical NAT configuration, computers in the local network (i.e. behind the NAT) are configured with IP addresses from a private IP address range. The NAT in the local network is connected to the Internet with at least one publicly routable IP address. It may optionally have more than one public IP address.

When a computer from the local network sends some data to a host in the internet, the NAT translates the local IP address to one of its public address(es). The port number of the TCP or UDP connection may also be modified. The NAT keeps track of active connections so that it is able to translate inbound packets to the correct local addresses and ports.

In essence, a NAT multiplexes its public IP address for many computers in the local network which do not have public IP addresses of their own. As such, with the advent of IPv6, we expect the use of NAT will slowly diminish. However, they may still remain in use for purposes like protocol translation.

2.1.2 Firewall

Unlike NAT, a firewall do not typically make any changes to packets passing through them. Instead, it monitors the traffic to ensure no unwanted or malicious traffic comes into the local network. Therefore, it too needs to maintain some state about each connection that goes through it. Typically, only hosts within the local network are allowed to initiate new connections (i.e., cause firewall to create connection state). Packets coming from

outside of the local network are dropped unless they belong to an existing connection, or are otherwise explicitly permitted. While NATs are generally not needed for IPv6, the firewalls will still be very much needed.

2.1.3 Proxy Server

According to [10], *A proxy server is a server (a computer system or an application) that acts as an intermediary for requests from clients seeking resources from other servers. A client connects to the proxy server, requesting some service, such as a file, connection, web page, or other resource available from a different server and the proxy server evaluates the request as a way to simplify and control its complexity. Proxies were invented to add structure and encapsulation to distributed systems. Today, most proxies are web proxies, facilitating access to content on the World Wide Web and providing anonymity.*

A proxy can also be used for monitoring and filtering traffic. As such, it also needs to maintain some state about each connection that goes through it.

2.1.4 Binding Timeout

As discussed so far, all types of middle-boxes require to maintain some state for each TCP or UDP connection passing through them. This takes up some resources; as such when a connection is closed, the middle-box needs to detect it immediately. It would then be able to release all those resources.

When a connection is closed gracefully, the middle-box can release the corresponding resources right away. However, when a TCP connection is abruptly closed at one end the other end and middle-boxes may not realize it until an attempt is made to send another packet through that connection. Such connections may block resources of the middle-box indefinitely, although no communication over the connection will succeed. To work around this problem, the middle-box assumes that the connection is ungracefully closed,

if no packet has passed through that connection for a specified amount of time. This time is called the binding timeout.

A middle-box would create a timer for each connection and set it to fire after binding timeout has elapsed. The timer is reset whenever there is traffic through the connection, however. If the timer does fire, the corresponding connection is considered to be dead and the state maintained for that connection is cleared.

Studies have shown that the NAT binding timeouts vary dramatically amongst the commercially available home gateways [22]. We expect similar pattern for other middle-boxes like firewalls and proxy servers as well.

2.2 Keep-alive

If the binding timeout is small, a middle-box may incorrectly close down active connections with sparse traffic pattern. As such, when the expected traffic pattern is sparse, a client needs to ensure that middle-box does not think that the connection became idle. This is done by sending periodic dummy packets, even when there is no data to send. A dummy packet sent to reset the binding timeout timer of the middle-box is called a heart beat or a keep-alive (KA) packet.

Power consumption of a mobile device is significantly influenced by the frequency of keep-alive messages [23]. As such, increasing the interval between successive keep-alive packets without jeopardizing the connection integrity is of high significance.

2.2.1 TCP Keep-alive

The Transmission Control Protocol, TCP in short, is a widely used transport layer protocol. It provides reliable host-to-host communication in packet-switched computer networks, and in interconnected systems of such networks [31]. Some implementations of TCP support keep-alive packets, although this is not universally accepted [17]. When the

keep-alive timer fires, a keep-alive probe packet is sent by the TCP stack. It contains no data and has the ACK flag turned on. The remote endpoint responds with a packet that has the ACK set and contains no data. In an Ethernet network, a keepalive frame length is 60 bytes; the response is 54 bytes long.

However, note that ACK segments that contain no data are not reliably transmitted by TCP. If a TCP keep-alive fails, we may not reliably conclude that the connection has been closed. As such, it is preferable to implement the keep-alive feature at the application level.

2.2.2 Application Keep-alive

Application keep-alive is implemented in the application layer, instead of the transport layer. An example is HTTP keep-alive. Notification services also use application keep-alive to maintain the connection through middle-boxes. The client side component of the notification service implements the keep-alive.

2.2.3 Keep-alive Interval

Keep-alive interval is the time between successive keep-alive messages. This interval should be smaller than the binding timeout of middle-box. The keep-alive interval of an application needs to adapt to the widely varying binding timeouts of different middle-boxes. When a mobile device moves from one network to another, it needs to adjust any application level keep-alive interval.

2.2.4 Optimal Keep-alive Interval

As mentioned earlier, increasing the interval between successive keep-alive packets without jeopardizing the connection integrity is very much preferable. If the keep-alive packets are sent just when the binding timeout timer of the middle-box is about to expire, then the

timer will be reset instead and the connection will not get terminated. Such an interval is called optimal keep-alive interval. We denote it by α. Obviously, α is slightly smaller than the binding timeout of middle-box, due to propagation delay.

Since the binding timeout of different middle-boxes vary widely, the value of optimal keep-alive interval cannot be pre-defined. Instead it needs to be dynamically detected through testing in the network environment that the mobile device is in.

2.3 Mobile Access Point Name (APN)

An Access Point Name is the name of a gateway between a GPRS, 3G or 4G mobile network and the Internet. It identifies the packet data network (PDN) that a mobile user is communicating with.

A structured APN consists of two parts: a Network Identifier and an Operator Identifier. The network identifier is mandatory. It defines the external network to which the Gateway GPRS Support Node (GGSN) is connected. The operator identifier, on the other hand, is optional. It defines the specific operators packet domain network in which the GGSN is located.

APN could potentially be used by a mobile device to detect whether it has been to this network before or not. A mobile device can cache certain features of a network (e.g. optimal keep-alive interval), using the APN as a key.

2.4 Power Consumption Model

A power consumption model helps in predicting power consumption due to some specific events or during a scenario, without having to perform actual power measurements. Yang proposed a M/G/1 vacation model for UMTS Discontinuous Reception (DRX) [42]. Using this model, he shows how to select the best inactivity timer and DRX cycle values for

Table 2.1: (Excerpted from [16]) Energy model for downloading x kilobytes of data over 3G, GSM and WiFi networks. All values except the Maintenance values for 3G and GSM, are averaged over more than 50 trials.

	3G	**GSM**	**WiFi**
Transfer Energy ($R(x)$)	$0.025(x) + 3.5$	$0.036(x) + 1.7$	$0.007(x) + 5.9$
Tail energy (E)	0.62 J/sec	0.25 J/sec	NA
Maintenance (M)	0.02 J/sec	0.03 J/sec	0.05 J/sec
Tail-time (T)	12.5 seconds	6 seconds	NA

various traffic patterns to optimize power consumption of the mobile station.

Balasubramanian et al. proposed another power consumption model for 3G, GSM and WiFi networks [16]. Their model is shown in Table 2.1. 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.

From this model, we can say that the cost of sending a keep-alive packet over 3G is approximately 11.25 joules. 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. Naturally, by reducing the number of keep-alives, we can reduce the overall power consumption.

Zhang et al. described a novel automated power model construction technique in [43]. This technique uses built-in battery voltage sensors and knowledge of battery discharge behavior to determine component-level power consumption. This approach was used to

implement PowerTutor [9], a power estimation tool for Android platform. Several other power models can be found in [34, 40, 41].

Chapter 3

Keep-alive Interval Detection

In this chapter, we apply several search techniques to dynamically detect the optimal keep-alive interval of a network. The techniques are compared against each other based on different criteria. This is done based on theoretical analysis. We have also performed experiments on a simulation platform. The simulation results are discussed in the next chapter.

Figure 3.1 shows a flow chart of the steps taken in detecting the optimal keep-alive interval. The first step is to open a TCP connection with the target service. We apply a conservative keep-alive interval in that connection that should just work. This is the maximum keep-alive interval that is known to work. As such, we also use this 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 represents the maximum keep-alive interval that may potentially work. Essentially, this is 1 less than the minimum keep-alive interval that is already known to not work. In the figure, the search range has been represented as the interval $[low, high]$.

Starting from this 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.

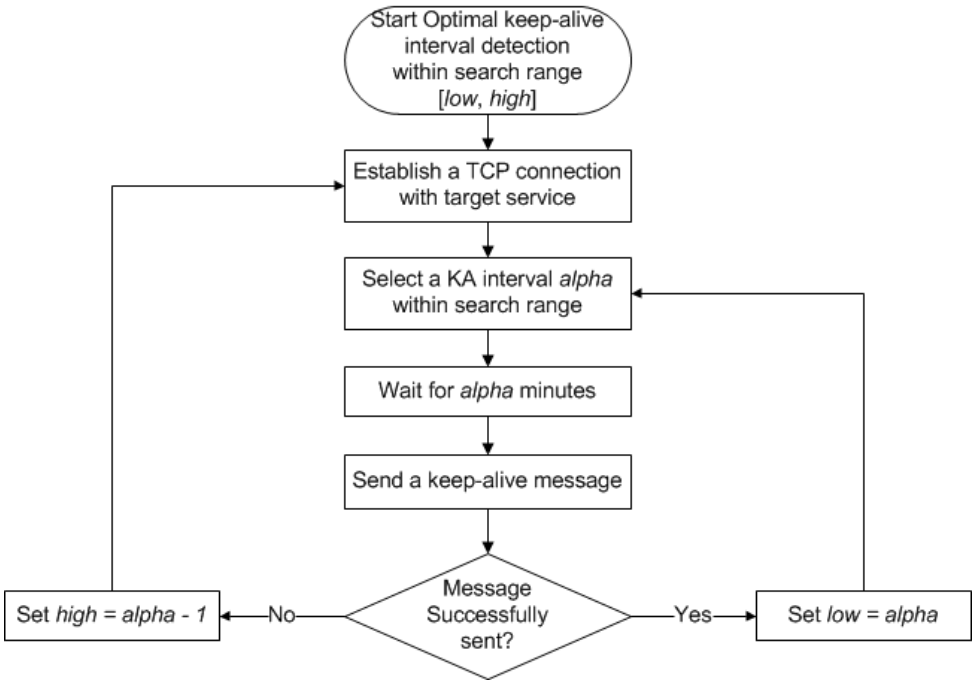


Figure 3.1: Flow chart of KA interval detection technique.

To ensure that the data connection does not get adversely affected during the search for optimal keep-alive interval, the search is performed on a separate connection. This connection is called a Test connection. A test connection is preferable to have more control over the testing as well – no data packet other than the keep-alive packets is sent over this connection.

While the testing is happening, the data connection needs to send keep-alive with conservative interval that would keep the connection alive in any home gateway scenario. As the testing reveals improved intervals, the new intervals can be applied on the data connection right away.

In the sections that follow, we explore three different techniques for optimal KA in-

terval detection. These techniques vary in how they select the *next* KA interval to test. The techniques are: binary search, exponential search and a combination of both. We refer to the latter approach as *Hybrid Search*. In our analysis and experiments, we have set $low = 1$ minute for each of these techniques. 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 [22]. Exponential and hybrid search techniques do not need to specify a higher bound on the search range. In other words, we set $high = \infty$. For binary search, we have used $high = 128$ minutes. That means, the maximum candidate interval that may work as keep-alive interval is 128 minutes. This is a power of 2 and is slightly higher than the IETF recommended timeout of 124 minutes [21] for any TCP connection.

3.1 KA Interval Detection Using Binary Search

As mentioned earlier, the lower bound of the search space is set to 1 minute in binary search. The higher bound is set to 2^k for some $k \geq 1$. At each step 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 $ProbeSequence(24) = \{65, 33, 17, 25, 21, 23, 24\}$. Algorithm 1 lists the steps for KA interval detection using binary search.

In next few sections, we analyze the performance of binary search technique. We find the number of tests performed and the time it takes to find the optimal keep-alive interval in best, worst and average cases.

Algorithm 1 KA Interval Detection by Binary Search

```
procedure KABinarySearch(low, high)
    optimal ← low                    ▷ optimal is a global parameter of the system
    Establish TCP connection with server
    while high − low ≥ 1 do
        test ← (low + high + 1)/2
        Wait for test units of time
        Send a keep-alive message to server
        if message was successfully sent then
            low ← test
            optimal ← test
        else
            high ← test
            Re-establish TCP connection with server
        end if
    end while
end procedure
```

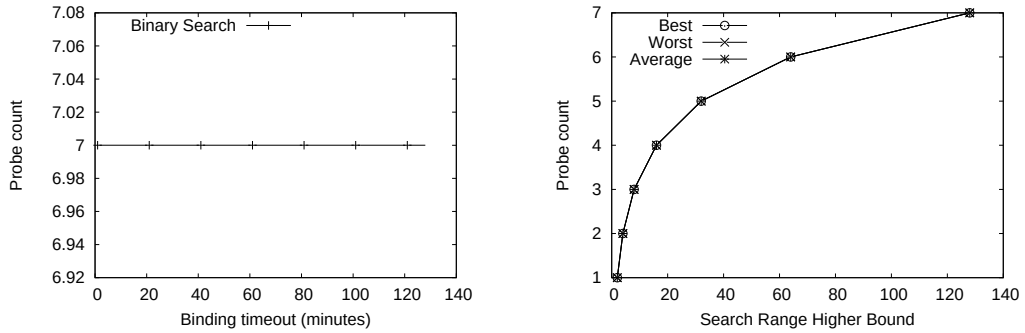
3.1.1 Number of Test Probes

Let α denote the optimal keep-alive interval. Let the search range be $[1, h]$ where $h = 2^k$ for some $k \geq 1$. Let $N(\alpha)$ denote the number of test probes needed to detect the optimal KA interval. With binary search, the total number of test probes for any value of α would be $\lg h$. Therefore,

$$N(\alpha) = \lg h \tag{3.1}$$

Since the above expression is independent of α, the maximum (N_{worst}), minimum (N_{best}) and average (N_{avg}) value of this function will also be $\lg h$. Therefore,

$$N_{best} = N_{avg} = N_{worst} = \lg h \tag{3.2}$$



(a) Probe count as a function of binding timeout.

(b) Best, worst and average case probe count as a function of search range.

Figure 3.2: Probe count of binary search.

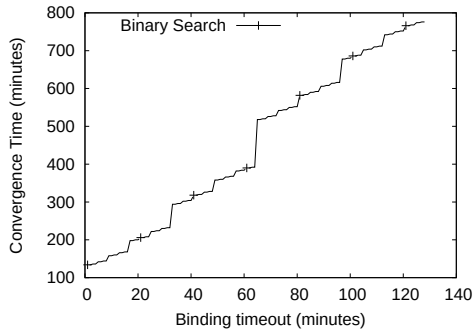
Figure 3.2(a) shows the probe count for different values of α, when the search range is $[1, 128]$. Figure 3.2(b) shows how the probe count increases with the search range higher bound.

3.1.2 Convergence Time

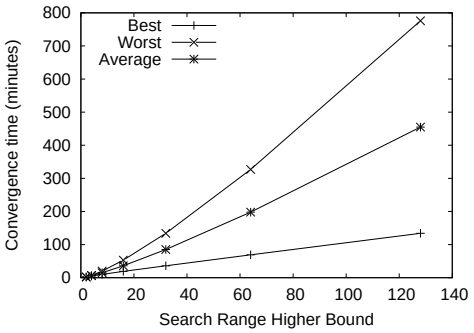
Let $T(\alpha)$ denote the time it takes to detect the optimal KA interval α. This is called the convergence time. The first probe takes $1 + 2^{k-1}$ unit time. if $\alpha \geq 1 + 2^{k-1}$, then the second probe takes $1 + 2^{k-1} + 2^{k-2}$ unit time after completion of first probe. On the other hand, if $\alpha \leq 2^{k-1}$, then the second probe takes only $1 + 2^{k-2}$ unit time. Similarly, third probe will take at least $1 + 2^{k-3}$ unit time. If $\alpha \geq 1 + 2^{k-2}$, then it takes 2^{k-2} unit more time. If $\alpha \geq 1 + 2^{k-1}$ then it takes a further 2^{k-1} unit more 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:

$$\begin{aligned} T(\alpha) &= \sum_{i=0}^{k-1}(1 + 2^i) + \sum_{i=0}^{k-1} a_i i 2^i \\ &= (2^k + k - 1) + \sum_{i=1}^{k-1} a_i i 2^i \\ &= (h + \lg h - 1) + \sum_{i=1}^{\lg h - 1} a_i i 2^i \end{aligned} \tag{3.3}$$



(a) Convergence time as a function of binding timeout.



(b) Best, worst and average case convergence time as a function of search range higher bound.

Figure 3.3: Convergence time of binary search.

As an example, let us take the search range $[1, 64]$. Therefore, $h = 64$ and $\lg h = 6$. Let's take $\alpha = 56$. The bit pattern of $\alpha - 1$ is 110111. Therefore,

$$\begin{aligned} T(56) &= (64 + 6 - 1) \\ &\quad + 1 \times 5 \times 2^5 + 1 \times 4 \times 2^4 + 0 \times 3 \times 2^3 \\ &\quad + 1 \times 2 \times 2^2 + 1 \times 1 \times 2^1 \\ &= 69 + 160 + 64 + 8 + 2 \\ &= 303 \end{aligned}$$

On the other hand, if we take the probe sequence to converge to the optimal keep-alive time of 56, it would be: $ProbeSequence(\alpha) = \{33,\ 49,\ 57,\ 53,\ 55,\ 56\}$

Therefore, the total time for the testing is:

$$\begin{aligned} T(\alpha) &= 33 + 49 + 57 + 53 + 55 + 56 \\ &= 303 \end{aligned}$$

Thus, we have verified equation 3.3.

Figure 3.3(a) shows the convergence time for different values of α, when the search range is $[1, 128]$. Clearly, convergence time is a monotone increasing function of binding timeout. Convergence time is minimum, when all the bits, except the Least Significant

Bit (LSB), of $\alpha - 1$ is 0. So, the best case convergence time happens when α is 1 or 2. And that value is: $2^k + k - 1$ unit time. Therefore, convergence time in best case is:

$$\begin{aligned} T_{best} &= 2^k + k - 1 \\ &= h + \lg h - 1 \\ &\approx O(h) \end{aligned} \tag{3.4}$$

For $T(\alpha)$ to reach its maxima, all the bits, except the LSB, in $\alpha - 1$ must be set. The LSB may or may not be set. So, the worst case convergence time occurs for $\alpha = 2^k$ or $\alpha = 2^k - 1$.

$$\begin{aligned} T_{worst} &= T(2^k) \\ &= (2^k + k - 1) + \sum_{i=1}^{k-1} a_i i 2^i \\ &= (2^k + k - 1) + \sum_{i=1}^{k-1} i 2^i \\ &= (2^k + k - 1) + 2^k(k-2) + 2 \\ &= 2^k(k-1) + k + 1 \\ &= h(\lg h - 1) + 1 + \lg h \\ &= h \lg h + \lg h - h + 1 \\ &\approx O(h \lg h) \end{aligned} \tag{3.5}$$

Note that the following equality has been used in simplifying the above expression. This will be used many times in our future calculations, so we call it out here:

$$\sum_{i=1}^{k} i 2^i = 2^{k+1}(k-1) + 2 \tag{3.6}$$

What would be the convergence time on average? We assume that the binding time-out is equally likely to be any discrete value in the search range. Therefore, if we sum the convergence time for all values of α and divide that by 2^k, we can get the average convergence time in the specified search range.

$$\begin{aligned} T_{avg} &= \frac{1}{2^k}\sum_{\alpha=1}^{2^k} T(\alpha) \\ &= \frac{1}{2^k}\sum_{\alpha=1}^{2^k}(2^k + k - 1) + \frac{1}{2^k}\sum_{\alpha=1}^{2^k}\sum_{i=1}^{k-1} a_i i 2^i \\ &= (2^k + k - 1) + \frac{1}{2^k}\sum_{i=1}^{k-1}\sum_{\alpha=1}^{2^k} a_i i 2^i \end{aligned}$$

In order to evaluate the second term of the right hand side of this expression, let us look at $\sum_{\alpha=1}^{2^k} a_i i 2^i$. This can be rewritten as $\sum_{\alpha-1=0}^{\alpha-1=2^k-1} a_i i 2^i$. Now, the i^{th} bit is set for exactly 2^{k-1} values of $\alpha - 1$. Therefore, this reduces to $2^{k-1} i 2^i$. As such, we can rewrite the expression as:

$$\begin{aligned} T_{avg} &= (2^k + k - 1) + \frac{1}{2^k}\sum_{i=1}^{k-1} 2^{k-1} i 2^i \\ &= (2^k + k - 1) + \frac{1}{2}\{2^k(k-2) + 2\} \\ &= (2^k + k - 1) + 2^{k-1}(k-2) + 1 \\ &= 2^{k-1}k + k \\ &= (2^{k-1} + 1)k \\ &= (\frac{h}{2} + 1)\lg h \\ &\approx O(h \lg h) \end{aligned} \tag{3.7}$$

Figure 3.3(b) plots the best, worst and average case convergence time for different search range higher bounds. While the lower bound of the search range in these figures are set to 1, we can easily extend the above equations for search range $[\beta + 1, \beta + h]$, for some $\beta \geq 0$. We observe that, the size of the range is still the same; hence the probe count, on average, will be $\lg h$. For each probe, there will be an additional wait time of β. Therefore,

$$T_{avg} = (\frac{h}{2} + 1 + \beta)\lg h$$

Based on the above calculations for convergence time and probe counts in binary search, we can now write the following theorems.

Theorem 3.1.1. *When detecting the optimal keep-alive interval using binary search technique in the range* $[1, h]$ *for some* $h = 2^k$*,* $k \geq 1$*, the number of tests performed in best, worst and average case is* $O(\lg h)$*.* □

Theorem 3.1.2. *When detecting the optimal keep-alive interval using binary search technique in the range* $[1, h]$ *for some* $h = 2^k$*,* $k \geq 1$*, the convergence time monotonically increases with the binding timeout of middle-box. The best case convergence time is* $O(h)$*. In worst case, as well as on average, it is* $O(h \lg h)$*.* □

3.2 KA Interval Detection Using Exponential Search

In exponential search, the lower bound of the search space is set to 1 minute. The algorithm does not require a higher bound initially. In other words, the higher bound can be set to infinity. The difference between successive tested intervals follows geometric progression. 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 the search range higher bound, then there is no need to test that interval. Instead, the lower bound is increased to the last successfully tested interval; and the geometric 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 higher bound of the search space 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 $ProbeSequence(24) = \{2, 4, 8, 16, 32, 17, 19, 23, 31, 24, 26, 25\}$. The steps of exponential search based approach is shown in Algorithm 2.

In next 2 sections, we analyze the performance of exponential search technique. We

Algorithm 2 KA Interval Detection by Exponential Search

```
procedure KAExponentialSearch(low)
    optimal ← low                    ▷ optimal is a global parameter of the system
    high ← ∞
    increment ← 1
    Establish TCP connection with server
    while high − low ≥ 1 do
        if optimal + increment > high then
            low ← optimal
            increment ← 1
            continue
        end if
        test ← optimal + increment
        increment ← increment × 2
        Wait for test units of time
        Send a keep-alive message to server
        if message was successfully sent then
            optimal ← test
        else
            high ← test − 1
            low ← optimal
            increment ← 1
            Re-establish TCP connection with server
        end if
    end while
end procedure
```

find the number of tests performed and the time it takes to find the optimal keep-alive interval in best, worst and average cases.

3.2.1 Number of Test Probes

As before, let $N(\alpha)$ denote the number of test probes needed to detect the optimal keep-alive interval α. The keep-alive interval is increased using geometric progression. Therefore, the intervals that are tested are $2, 4, 8, 16$ and so on. When a test fails, we take the range between the last successful interval and the failed interval; we repeat the testing in this range. Let p probes are performed until the first failure, where $p \geq 1$. Then, we can write:

$$2^{p-1} \leq \alpha < 2^p$$
$$\Rightarrow p - 1 \leq \lg(\alpha) < p$$
$$\Rightarrow p = 1 + \lfloor \lg(\alpha) \rfloor$$

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:

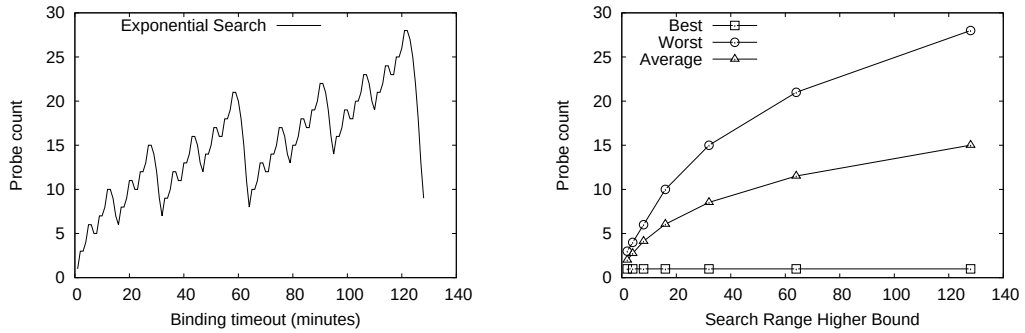
$$N'(\alpha, k) = \begin{cases} k & \text{when } \alpha = 2^k \\ p + N'(\alpha + 1 - 2^{p-1}, p - 1) & \text{otherwise.} \end{cases}$$

Then, we can write:

$$N(\alpha) = p + N'(\alpha + 1 - 2^{p-1}, p - 1) \tag{3.8}$$

Note that, using this expression of $N(\alpha)$, we can rewrite the expression of $N'(\alpha, k)$ as follows:

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



(a) Probe count as a function of binding timeout.

(b) Best, worst and average case probe count as a function of search range higher bound.

Figure 3.4: Probe count of exponential search.

Figure 3.4(a) plots the probe count for each value of α in the range $[1, 128]$. It is obvious from this plot that $N(\alpha)$ is not a monotonic function of α. We can also demonstrate this by evaluating the function for different values of α. For example, we can check $N(30)$ and $N(31)$ to show that $N(\alpha)$ is not monotonic.

$$\begin{aligned} N(30) &= 1 + 4 + N'(15, 4) \\ &= 5 + 1 + 3 + N'(8, 3) \\ &= 9 + N'(2^3, 3) \\ &= 9 + 3 \\ &= 12 \\ N(31) &= 1 + 4 + N'(16, 4) \\ &= 5 + N'(2^4, 4) \\ &= 5 + 4 \\ &= 9 \end{aligned}$$

From Equation 3.8, we can say that the minimum value of $N(\alpha)$ is 1, which is due to $\alpha = 1$.

$$N_{best} = N(1) = 1 \tag{3.9}$$

We will now try to find out the worst case probe count, when α is in the range $[1, h]$

for some $h = 2^k, k \geq 1$. For $\alpha = 2^k$, the number of probes performed in the first phase (until the first failure) is $k+1$. Then only 1 probe is performed in the next phase. So, the total probe count is $k + 2$. For all other possible values of α, the maximum probe count that can happen in phase one is k. The next range that is searched in phase 2 is no more than half the size of the original range. As such, the maximum probe count in this phase is no more than $k - 1$. Continuing along this line of argument, the maximum probe count can be no more than $k + (k-1) + \cdots + 1 = \frac{k(k+1)}{2}$. However, we observe that,

$$\frac{k(k+1)}{2} < k + 2 \text{ when } k < 3$$

Therefore, we can write:

$$N_{worst} \begin{cases} = k + 2 & \text{when } k < 3 \\ \leq \frac{k(k+1)}{2} & \text{otherwise.} \end{cases}$$

We have established an upper bound on N_{worst} for $k \geq 3$. Can we actually find an α for which the upper bound is achieved? In fact, this is achieved for $\alpha = 2^k - k$. In the first phase, there will be k probes. As pointed out earlier, the next phase would be similar to searching for $\alpha + 1 - 2^{p-1}$ in the range $[1, 2^{p-1}]$. Here $p = k$, so this is like searching $2^{k-1} - (k-1)$ in the range $[1, 2^{k-1}]$. Therefore, $(k-1)$ probes will be needed in this phase; and so on. As such, the worst case probe count is:

$$\begin{aligned} N_{worst} &= \begin{cases} k + 2 & \text{when } k < 3 \\ \frac{k(k+1)}{2} & \text{otherwise.} \end{cases} \\ &= \begin{cases} \lg h + 2 & \text{when } h \leq 4 \\ \frac{\lg h(\lg h+1)}{2} & \text{otherwise.} \end{cases} \\ &\approx O(\lg^2 h) \end{aligned} \tag{3.10}$$

Now, we compute the average probe count. Let F_k denote the total number of probes for all possible values of α in the bounded search space $[1, 2^k]$. For $\alpha = 2^k$, the k-th probe succeeds, and since the next interval to test is beyond the search range, we are done. As such, the number of probes needed is k. On the other hand, the total number of probes for

all possible values of α in the sub-range $[2^{p-1}, 2^p - 1]$ would be $2^{p-1}p + F_{p-1}$ for $1 \leq p \leq k$. Therefore, we can write:

$$\begin{aligned} F_k &= \sum_{p=1}^{k} 2^{p-1}p + \sum_{p=1}^{k} F_{p-1} + k \\ &= \frac{1}{2}\{2^{k+1}(k-1) + 2\} + \sum_{p=1}^{k} F_{p-1} + k \\ &= 2^k(k-1) + (k+1) + \sum_{p=1}^{k} F_{p-1} \end{aligned}$$

Therefore,

$$\begin{aligned} F_k - F_{k-1} &= 2^k(k-1) + (k+1) + \sum_{p=1}^{k} F_{p-1} \\ &\quad - 2^{k-1}(k-2) + k + \sum_{p=1}^{k-1} F_{p-1} \\ &= 2^{k-1}k + 1 + F_{k-1} \\ \Rightarrow F_k &= 2^{k-1}k + 1 + 2F_{k-1} \\ &= 2^{k-1}k + 1 + 2\{2^{k-2}(k-1) + 1 + 2F_{k-2}\} \\ &= 2^{k-1}\{k + (k-1)\} + 1 + 2 + 2^2F_{k-2} \\ &= \cdots \\ &= 2^{k-1}\{k + (k-1) + \cdots + 2\} + \{1 + 2 + 2^2 + \cdots + 2^{k-2}\} + 2^{k-1}F_1 \\ &= 2^{k-2}k(k+1) - 2^{k-1} + 2^{k-1} - 1 + 2^k \\ &= 2^{k-2}\{k(k+1) + 4\} - 1 \end{aligned}$$

But, when the initial search space is unbounded, the probe count for $\alpha = 2^k$ is $k + 2$ instead of k. Therefore, we can write:

$$\begin{aligned} N_{avg} &= \frac{1}{2^k}(F_k + 2) \\ &= \frac{1}{2^k}[2^{k-2}\{k(k+1) + 4\} + 1] \\ &= \frac{k(k+1)}{4} + 1 + \frac{1}{2^k} \end{aligned}$$

Replacing 2^k with h and k with $\lg h$, we can then write:

$$\begin{aligned} N_{avg} &= \frac{\lg h(1+\lg h)}{4} + 1 + \frac{1}{h} \\ &\approx O(\lg^2 h) \end{aligned} \tag{3.11}$$

Figure 3.4(b) plots the best, worst and average case probe counts in exponential search for different values of search range higher bound.

3.2.2 Convergence Time

As before, Let $T(\alpha)$ denote the convergence time when the binding timeout is α. As mentioned earlier, the first failure occurs after $p = 1 + \lfloor \lg(\alpha) \rfloor$ probes. Afterwards, 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:

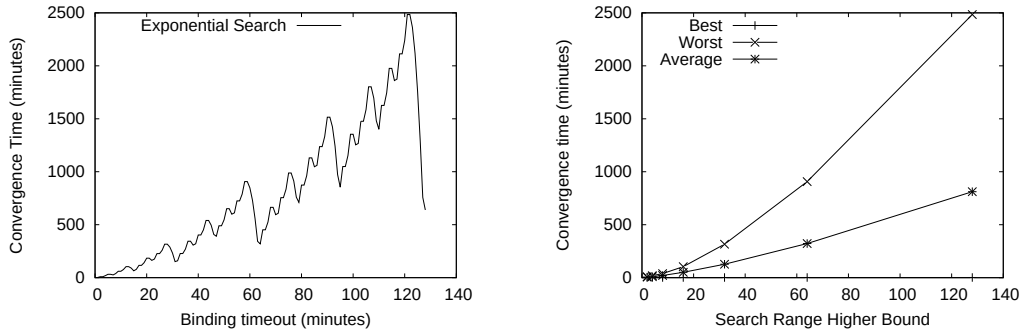
$$T'(\alpha,k) = \begin{cases} 2^{k+1} - 2 & \text{when } \alpha = 2^k \\ 2^{p+1} - 2 & \\ +T'(\alpha + 1 - 2^{p-1}, p-1) & \\ +(2^{p-1} - 1)N'(\alpha + 1 - 2^{p-1}, p-1) & \text{otherwise.} \end{cases}$$

Then, we can write:

$$\begin{aligned} T(\alpha) = 2^{p+1} - 2 &+ T'(\alpha + 1 - 2^{p-1}, p-1) \\ &+ (2^{p-1} - 1)N'(\alpha + 1 - 2^{p-1}, p-1) \end{aligned} \tag{3.12}$$

Using this expression of $T(\alpha)$, we can rewrite the expression of $T'(\alpha, k)$ as follows:

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



(a) Convergence time as a function of binding timeout.

(b) Best, worst and average case convergence time as a function of search range.

Figure 3.5: Convergence time of exponential search.

Figure 3.5(a) plots the convergence time for each value of α in the range $[1, 128]$. As can be seen from this plot, $T(\alpha)$ is also not a monotonic function of α.

We can also demonstrated this by evaluating the function for different values of α. For example, we can check $T(30)$ and $T(31)$ to show that $T(\alpha)$ is not monotonic.

$$\begin{aligned} T(30) &= 2^6 - 2 + T'(15, 4) + 15N'(15, 4) \\ &= 62 + \{2^5 - 2 + T'(8, 3) + 7N'(8, 3)\} + 15 \times 7 \\ &= 62 + 30 + (2^4 - 2) + 7 \times 3 + 105 \\ &= 232 \\ T(31) &= 2^6 - 2 + T'(16, 4) + 15N'(16, 4) \\ &= 62 + (2^5 - 2) + 15 \times 4 \\ &= 152 \end{aligned}$$

From equation 3.12, we can say that the minimum value of $T(\alpha)$ is 2, which is due to $\alpha = 1$.

$$T_{best} = T(1) = 2 \tag{3.13}$$

We will now try to find out the worst case convergence time, across the values of α in the range $[1, h]$ for some $h = 2^k, k \geq 1$. We will first find an upper bound on the convergence time, and then show that the upper bound is achieved for some value of α.

For $\alpha = 2^k$, the first failure happens when a probe is made with a timeout of 2^{k+1}. Subsequently, we will check with a timeout of $2^k + 1$, which will also fail. Thus we are able to identify that the binding timeout is 2^k.

$$\begin{aligned} T(2^k) &= (2 + 4 + \cdots + 2^{k+1}) + 2^k + 1 \\ &= 2(2^{k+1} - 1) + 2^k + 1 \\ &= 5.2^k - 1 \end{aligned}$$

For $\alpha < 2^k$, it will take maximum time to reach first failure, if $2^{k-1} \leq \alpha < 2^k$; and that time will be $(2 + 4 + \cdots + 2^k) = 2^{k+1} - 2$. Subsequent search is similar to searching for $\alpha + 1 - 2^{k-1}$ in the bounded search space $[1, 2^{k-1}]$, except that each probe takes an extra $2^{k-1} - 1$ unit time. Therefore, after first failure, the maximum time we might need to reach the second failure is: $(k-1)(2^{k-1} - 1) + (2^k - 2)$. As such, the maximum time possibly needed to identify the binding timeout would be:

$$\begin{aligned} &= (2^{k+1} - 2) \\ &+ (2^k - 2) + (k-1)(2^{k-1} - 1) \\ &+ (2^{k-1} - 2) + (k-2)(2^{k-1} - 1 + 2^{k-2} - 1) \\ &+ \cdots \\ &+ (2^2 - 2) + 1 \times \{(2^{k-1} - 1) + (2^{k-2} - 1) + \cdots + (2^1 - 1)\} \\ &= 2^{k+2} - 2(k+2) + A \end{aligned}$$

Where,

$$\begin{aligned} A &= (k-1)(2^{k-1} - 1) + (k-2)(2^{k-1} - 1 + 2^{k-2} - 1) \\ &+ \cdots + 1.\{(2^{k-1} - 1) + (2^{k-2} - 1) + \cdots + (2^1 - 1)\} \\ &= (2^{k-1} - 1)\{1 + 2 + \cdots + (k-1)\} + (2^{k-2} - 1)\{1 + 2 + \cdots + (k-2)\} \\ &+ \cdots + (2^1 - 1).1 \\ &= 1.(2^1 + 2^2 + \cdots + 2^{k-1}) + 2.(2^2 + 2^3 + \cdots + 2^{k-1}) + \cdots + (k-1)2^{k-1} \\ &- [1.(k-1) + 2.(k-2) + \cdots + (k-1).\{k - (k-1)\}] \\ &= B - C \end{aligned}$$

Where,

$$\begin{aligned} B &= 1.(2^1 + 2^2 + \cdots + 2^{k-1}) + 2.(2^2 + 2^3 + \cdots + 2^{k-1}) + \cdots + (k-1)2^{k-1} \\ &= 1.(2^k - 2) + 2.(2^k - 2^2) + \cdots + (k-1)(2^k - 2^{k-1}) \\ &= 2^{k-1}k(k-1) - \{2^k(k-2) + 2\} \\ &= 2^{k-1}(k^2 - 3k + 4) - 2 \end{aligned}$$

And,

$$\begin{aligned} C &= [1.(k-1) + 2.(k-2) + \cdots + (k-1).\{k - (k-1)\}] \\ &= \frac{k^2(k-1)}{2} - \frac{(k-1)k(2k-1)}{6} \\ &= \frac{k(k+1)(k-1)}{6} \end{aligned}$$

Therefore, the upper bound on the convergence time, for $\alpha < 2^k$, would be:

$$\begin{aligned} &= 2^{k+2} - 2(k+2) + 2^{k-1}(k^2 - 3k + 4) - 2 - \frac{k(k+1)(k-1)}{6} \\ &= \frac{k^2 - 3k + 12}{2}2^k - \frac{k^3 + 11k + 36}{6} \end{aligned}$$

Recall, on the other hand, that $T(2^k) = 5.2^k - 1$. Now, which one is bigger? To find this out, we can write:

$$\begin{aligned} &\frac{k^2 - 3k + 12}{2}2^k - \frac{k^3 + 11k + 36}{6} \\ &= 5.2^k - 1 + 2^k + 1 + \frac{k^2 - 3k}{2}2^k - \frac{k^3 + 11k + 36}{6} \\ &= T(2^k) + 2^{k-1}(k-1)(k-2) - \frac{1}{6}(k^3 + 11k + 30) \end{aligned}$$

This will be greater than $T(2^k)$, if $2^{k-1}(k-1)(k-2) - \frac{1}{6}(k^3 + 11k + 30) > 0$. This holds when $k \geq 4$. As such, we have established the upper bound on convergence time as follows:

$$T_{worst} \begin{cases} = 5.2^k - 1 & \text{when } k < 4 \\ \leq \frac{k^2-3k+12}{2}2^k - \frac{k^3+11k+36}{6} & \text{otherwise.} \end{cases}$$

Now, when $k \geq 4$, can we actually find an α for which the upper bound is achieved? When calculating this upper bound, we envisioned an α for which all the following conditions hold:

$$
\begin{aligned}
&2^{k-1} \leq \alpha < 2^k \\
&2^{k-2} \leq \alpha + 1 - 2^{k-1} < 2^{k-1} \\
&2^{k-3} \leq \alpha + 2 - 2^{k-1} - 2^{k-2} < 2^{k-2} \\
&\cdots \\
&1 \leq \alpha + k + 1 - 2^k < 2
\end{aligned}
$$

In fact, all these conditions are satisfied for $\alpha = 2^k - k$. Therefore, the upper bound is tight.

$$
\begin{aligned}
T_{worst} &= \begin{cases} 5.2^k - 1 & \text{when } k < 4 \\ \frac{k^2-3k+12}{2}2^k - \frac{k^3+11k+36}{6} & \text{otherwise.} \end{cases} \\
&= \begin{cases} 5h - 1 & \text{when } h \leq 8 \\ \frac{\lg^2 h - 3lgh + 12}{2}h - \frac{\lg^3 h + 11\lg h + 36}{6} & \text{otherwise.} \end{cases} \\
&\approx O(h \lg^2 h)
\end{aligned} \tag{3.14}
$$

Now, let us compute the convergence time on average. Let X_k denote the sum of convergence times for all possible values of α in the bounded search space $[1, 2^k]$. Let, $Y_{p,k}$ denote the sum of convergence times for all $\alpha \in [2^{p-1}, 2^p - 1]$, where $1 \leq p \leq k$. Also, For $\alpha = 2^k$, the convergence time would be $2 + 4 + \cdots + 2^k = 2^{k+1} - 2$. Therefore,

$$X_k = 2^{k+1} - 2 + \sum_{p=1}^{k} Y_{p,k}$$

Let us first try to obtain an expression for $Y_{p,k}$. To do so, we recall that, total number of probes for all possible values of α in the bounded search space $[1, 2^k]$ is $F_k = 2^{k-2}\{k(k+1) + 4\} - 1$. Therefore, we can write:

$$
\begin{aligned}
Y_{p,k} &= 2^{p-1}(2+4+\cdots+2^p) + X_{p-1} + (2^{p-1}-1)F_{p-1} \\
&= 2^{p-1}\{2^{p+1}-2\} + X_{p-1} + (2^{p-1}-1)[2^{p-3}\{p(p-1)+4\}-1] \\
&= 2^{2p} - 2^p + X_{p-1} + (2^{p-1}-1)\{2^{p-3}p(p-1) + 2^{p-1} - 1\} \\
&= 2^{2p} - 2^p + X_{p-1} + 2^{2p-4}p(p-1) + 2^{2p-2} - 2^p - 2^{p-3}p(p-1) + 1 \\
&= \frac{p^2-p+20}{16}2^{2p} - \frac{p^2-p+16}{8}2^p + 1 + X_{p-1}
\end{aligned}
$$

Using the above expression, we can rewrite the expression for X_k as

$$
X_k = 2^{k+1} - 2 + \sum_{p=1}^{k}\left[\frac{p^2-p+20}{16}2^{2p} - \frac{p^2-p+16}{8}2^p + 1 + X_{p-1}\right]
$$

Using the above to obtain X_{k-1} and then deducting it from X_k, we get,

$$
\begin{aligned}
X_k - X_{k-1} &= 2^k + \frac{k^2-k+20}{16}2^{2k} - \frac{k^2-k+16}{8}2^k + 1 + X_{k-1} \\
\Rightarrow X_k &= 2X_{k-1} + 2^k + \frac{k^2-k+20}{16}2^{2k} - \frac{k^2-k+16}{8}2^k + 1 \\
&= 2X_{k-1} + \frac{2^k}{16}(k^2 2^k - k2^k) - \frac{1}{8}(k^2 2^k - k2^k) + \frac{5}{4}2^{2k} - 2^k + 1 \\
&= 2^2 X_{k-2} + (1+2) + \frac{2^k}{16}\{k^2 2^k + (k-1)^2 2^{k-1}\} - \frac{2^k}{16}\{k2^k + (k-1)2^{k-1}\} \\
&\quad - \frac{2^k}{8}\{k^2 + (k-1)^2\} + \frac{2^k}{8}\{k + (k-1)\} + \frac{5}{4}2^k(2^k + 2^{k-1}) - (2^k + 2^k) \\
&= \cdots \\
&= 2^k X_0 + 2^k - 1 + \frac{2^k}{16}\sum_{i=1}^{k} i^2 2^i - \frac{2^k}{16}\sum_{i=1}^{k} i2^i \\
&\quad - \frac{2^k}{48}k(k+1)(2k+1) + \frac{2^k}{16}k(k+1) + \frac{5}{4}2^{k+1}(2^k - 1) - k2^k
\end{aligned}
$$

We now need to find a closed form of $\sum_{i=1}^{k} i^2 2^i$.

$$\begin{aligned}
\sum_{i=1}^{k} i^2 2^i &= \sum_{i=1}^{k} (i-1+1)^2 2^i \\
&= \sum_{i=1}^{k} (i-1)^2 2^i + 2\sum_{i=1}^{k} (i-1)2^i + \sum_{i=1}^{k} 2^i \\
&= 2\sum_{i=0}^{k-1} i^2 2^i + 4\sum_{i=0}^{k-1} i2^i + 2(2^k - 1) \\
&= 2\sum_{i=1}^{k} i^2 2^i - 2k^2 2^k + 4\{2^k(k-2)+2\} + 2^{k+1} - 2 \\
\Rightarrow \sum_{i=1}^{k} i^2 2^i &= 2^{k+1}k^2 - 2^{k+2}k + 2^{k+3} - 2^{k+1} - 6 \\
&= 2^{k+1}k(k-2) + 6(2^k - 1)
\end{aligned}$$

Putting this into the expression for X_k, we get:

$$\begin{aligned}
X_k &= 2^k.0 + 2^k - 1 + \frac{2^k}{16}\{2^{k+1}k(k-2) + 6(2^k-1)\} - \frac{2^k}{16}\{2^{k+1}(k-1)+2\} \\
&\quad - \frac{2^k}{48}k(k+1)(2k-2) + \frac{5}{4}2^{k+1}(2^k-1) - k2^k \\
&= \frac{k^2 - 3k + 24}{8}2^{2k} - \frac{k^3 + 23k + 48}{24}2^k - 1
\end{aligned}$$

But, when the initial search space is unbounded, two additional probes are needed for $\alpha = 2^k$. After the probe with timeout 2^k succeeds, we cannot assume that we have reached the optimal timeout. We need to subsequently test the timeout of 2^{k+1}. When that fails, we check whether $2^k + 1$ works or not. When that also fails, we are able to conclude that the network time out is 2^k. Therefore, the additional time spent is $2^{k+1} + 2^k + 1 = 3.2^k + 1$. Now, we can find the average convergence time:

$$\begin{aligned}
T_{avg} &= \frac{1}{2^k}(X_k + 3.2^k + 1) \\
&= \frac{1}{2^k}\left[\frac{k^2 - 3k + 24}{8}2^{2k} - \frac{k^3 + 23k - 24}{24}2^k\right] \\
&= \frac{k^2 - 3k + 24}{8}2^k - \frac{k^3 + 23k - 24}{24} \\
&= \frac{\lg^2 h - 3\lg h + 24}{8}h - \frac{\lg^3 h + 23\lg h - 24}{24} \\
&\approx O(h \lg^2 h)
\end{aligned} \tag{3.15}$$

Based on the above calculations for convergence time and probe counts in exponential search, we can now write the following theorems.

Theorem 3.2.1. *When detecting the optimal keep-alive interval using exponential search technique in the range* $[1, h]$ *for some* $h = 2^k$, $k \geq 1$, *the number of tests performed is not a monotonic function of the binding timeout of middle-box. The best case probe count is 1, while the worst case probe count* $O(\lg^2 h)$. *On average, the number of tests performed is also* $O(\lg^2 h)$. □

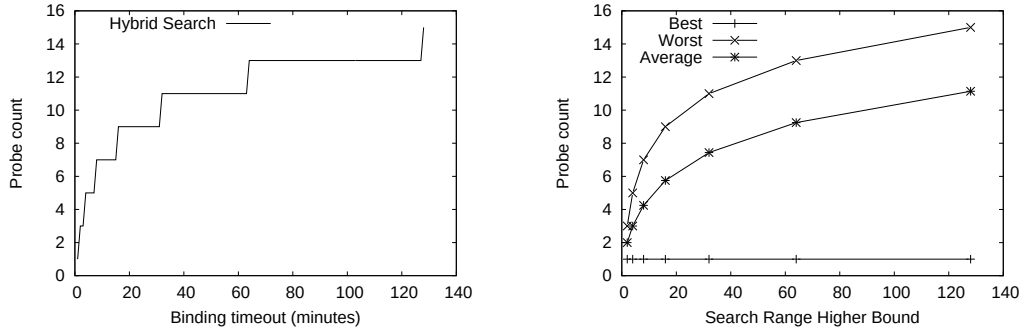
Theorem 3.2.2. *When detecting the optimal keep-alive interval using exponential search technique in the range* $[1, h]$ *for some* $h = 2^k$, $k \geq 1$, *the convergence time is not a monotonic function of the binding timeout of middle-box. The best case convergence time is 2. In worst case, as well as on average, it is* $O(h \lg^2 h)$. □

3.3 KA Interval Detection Using Hybrid Search

Hybrid search is a combination of binary and exponential searches. Initially, it acts like the exponential search. The lower bound of the search space is set to 1 minute; no higher bound is needed initially. Like exponential search, the difference between successive tested intervals follows geometric progression. When a tested interval overshoots the actual binding timeout, the test connection will get terminated. At that point, the lower bound is increased to the last successfully tested interval; the higher bound of the search space 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 $ProbeSequence(24) = \{2, 4, 8, 16, 32, 24, 28, 26, 25\}$. The steps of hybrid search based approach is shown in Algorithm 3.

Algorithm 3 KA Interval Detection by Hybrid Search

```
procedure KAHybridSearch(low)
    high ← ∞
    test ← low
    increment ← 1
    Establish TCP connection with server
    repeat                                          ▷ Exponential search phase
        optimal ← test                  ▷ optimal is a global parameter of the system
        test ← optimal + increment
        increment ← increment × 2
        Wait for test units of time
        Send a keep-alive message to server
    until Sending the keep-alive message failed
    high ← test − 1
    low ← optimal
    Re-establish TCP connection with server
    while high − low ≥ 1 do                         ▷ Binary search phase
        test ← (low + high + 1)/2
        Wait for test units of time
        Send a keep-alive message to server
        if message was successfully sent then
            low ← optimal ← test
        else
            high ← test
            Re-establish TCP connection with server
        end if
    end while
end procedure
```



(a) Probe count as a function of network timeout.

(b) Best, worst and average case probe count as a function of search range.

Figure 3.6: Probe count of hybrid search.

3.3.1 Number of Test Probes

Let $N(\alpha)$ denote the number of test probes needed to detect the optimal keep-alive interval α. In the first phase of hybrid technique, the keep-alive interval is increased using geometric progression. Therefore, the intervals that are tested are $2, 4, 8, 16, 32$ and so on. When a test fails, we take the range between the last successful interval and the failed interval; we perform binary search in this range there after.

If p probes are performed in the exponential phase ($p \geq 1$), all but the p-th probe succeed. As shown earlier, $p = 1 + \lfloor \lg(\alpha) \rfloor$. 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,

$$\begin{aligned} N(\alpha) &= p + p - 1 \\ &= 2 \times (1 + \lfloor \lg(\alpha) \rfloor) - 1 \\ &= 1 + 2\lfloor \lg(\alpha) \rfloor \end{aligned} \tag{3.16}$$

Figure 3.6(a) plots the probe count for each value of α in the range $[1, 128]$. As can be seen from the graph, $N(\alpha)$ is monotonic. The minima is reached for minimum value of α.

$$N_{best} = N(1) = 1 \tag{3.17}$$

We would also like to identify the worst and average case probe count. To do so, we

consider the possible values of α in the range $[1, h]$ for some $h = 2^k, k \geq 1$. $N(\alpha)$ will be maximum when α is maximum, since the function is monotonic. Therefore,

$$\begin{aligned} N_{worst} &= N(2^k) \\ &= 1 + 2\lfloor \lg 2^k \rfloor \\ &= 2k + 1 \\ &= 1 + 2 \lg h \\ &\approx O(\lg h) \end{aligned} \tag{3.18}$$

Now, we compute the average probe count. We assume that the binding timeout is equally likely to be any discrete value in the search range. We observed earlier that, when α is in $[2^{p-1}, 2^p - 1]$, total probe count is $2p - 1$. The size of this range is 2^{p-1}. The entire range $[1, 2^k]$ can be thought of the collection of all such sub-ranges for $1 \leq p \leq k$. Additionally, $\alpha = 2^k$ is also in this range, which is not covered by the sub-ranges. Therefore,

$$\begin{aligned} N_{avg} &= \frac{1}{2^k}\{N(2^k) + \sum_{p=1}^{k} 2^{p-1}(2p-1)\} \\ &= \frac{1}{2^k}\{2k + 1 + \sum_{p=1}^{k} 2^p p - \sum_{p=0}^{k-1} 2^p)\} \\ &= \frac{1}{2^k}\{2k + 1 + 2^{k+1}(k-1) + 2 - 2^k + 1\} \\ &= \frac{1}{2^k}\{2^{k+1}k - 3 \times 2^k + 2(k+2)\} \\ &= 2k - 3 + \frac{k+2}{2^{k-1}} \\ &= 2 \lg h + \frac{2}{h}(2 + \lg h) - 3 \\ &\approx O(\lg h) \end{aligned} \tag{3.19}$$

3.3.2 Convergence Time

Let $T(\alpha)$ denote the convergence time for detecting the optimal keep-alive interval α. Let $t_e(\alpha)$ and $t_b(\alpha)$ be the time spent during the exponential and binary phases, respectively.

Then,

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

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. Therefore, we can write:

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

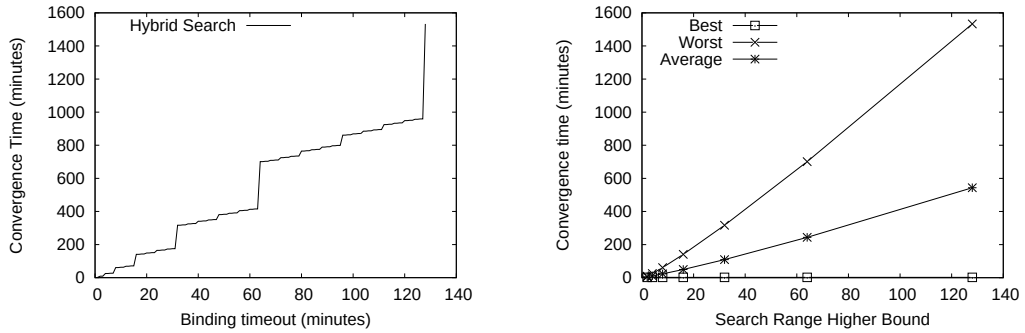
Where, $a_{p-1}a_{p-2}\cdots a_1 a_0$ is the binary representation of α. Combining the above expressions, we can write:

$$\begin{aligned} T(\alpha) &= t_e(\alpha) + t_b(\alpha) \\ &= 2^{p+1} - 2 + (2^{p-1} - 1)p + (p - 1) + \sum_{i=1}^{p-2} a_i i 2^i \\ &= 2^{p-1}(p + 4) - 3 + \sum_{i=1}^{p-2} a_i i 2^i \\ &= 2^{\lfloor \lg(\alpha) \rfloor}(5 + \lfloor \lg(\alpha) \rfloor) - 3 + \sum_{i=1}^{p-2} a_i i 2^i \\ &= 2^{\lfloor \lg(\alpha) \rfloor}(5 + \lfloor \lg(\alpha) \rfloor) - 3 + \sum_{i=1}^{\lfloor \lg(\alpha) \rfloor - 1} a_i i 2^i \end{aligned} \tag{3.20}$$

Figure 3.7 plots the convergence time for all values of α in the range $[1, 128]$. Like probe count, convergence time is also a monotonic function of α. The minima is reached for minimum value of α.

$$T_{best} = T(1) = 2 \tag{3.21}$$

$T(\alpha)$ will be maximum when α is maximum. As before, we will consider the range $[1, h]$ for some $h = 2^k, k \geq 1$.



(a) Convergence time as a function of network timeout.

(b) Best, worst and average case convergence time as a function of search range.

Figure 3.7: Convergence time of of hybrid search.

Therefore,

$$\begin{aligned} T_{worst} &= T(2^k) \\ &= 2^k(k+5) - 3 \\ &= h(5 + \lg h) - 3 \\ &= h \lg h + 5h - 3 \\ &\approx O(h \lg h) \end{aligned} \tag{3.22}$$

Now, let us compute the convergence time on average. As before, we assume that the binding timeout is equally likely to be any discrete value in the search range.

$$\begin{aligned} T(\alpha) &= t_e(\alpha) + t_b(\alpha) \\ &= 2^{p+1} - 2 + (2^{p-2} + 1 + 2^{p-1} - 1)(p-1) \\ &= 2^{p+1} - 2 + 3(p-1)2^{p-2} \end{aligned}$$

Where $t_b(\alpha)$ has been substituted using the expression for average convergence time of binary search in the range $[2^{p-1}, 2^p - 1]$. Therefore, sum of convergence times for all α in this range is:

$$\begin{aligned} & 2^{p-1}\{2^{p+1} - 2 + 3(p-1)2^{p-2}\} \\ &= 2^{2p} - 2^p + 3(p-1)2^{2p-3} \\ &= \frac{3p+5}{8} 2^{2p} - 2^p \end{aligned}$$

As such, sum of convergence times for all α across the entire range is:

$$\begin{aligned} F_k &= 2^k(k+5) - 3 + \sum_{p=1}^{k}(\frac{3p+5}{8}2^{2p} - 2^p) \\ &= 2^k(k+5) - 3 + \frac{5}{6}(4^k - 1) - 2(2^k - 1) + \frac{3}{8}G_k \end{aligned}$$

Where,

$$\begin{aligned} G_k &= \sum_{p=1}^{k} 2^{2p}p \\ &= 1 \times 4^1 + 2 \times 4^2 + \cdots + k \times 4^k \\ &= 4 + 4(2 \times 4^1 + 3 \times 4^2 + \cdots + k \times 4^{k-1}) \\ &= 4 + 4(4^1 + 4^2 + \cdots + 4^{k-1}) + 4(1 \times 4^1 + 2 \times 4^2 + \cdots + (k-1) \times 4^{k-1}) \\ &= 4 + \frac{16}{3}(4^{k-1} - 1) + 4G_k - k4^{k+1} \\ \Rightarrow 3G_k &= k4^{k+1} - \frac{4^{k+1}}{3} + \frac{4}{3} \\ &= \frac{4^{k+1}}{3}(3k-1) + \frac{4}{3} \\ \Rightarrow\ G_k &= \frac{1}{9}\{4^{k+1}(3k-1) + 4\} \end{aligned}$$

Putting the value of G_k into the expression for F_k, we get:

$$\begin{aligned} F_k &= 2^k(k+5) - 3 + \frac{5}{6}(4^k - 1) - 2(2^k - 1) + \frac{1}{6}\{4^k(3k-1) + 1\} \\ &= 2^{2k}\frac{3k+4}{6} + 2^k(k+3) - \frac{5}{3} \end{aligned}$$

Therefore, we can write:

$$\begin{aligned} T_{avg} &= \frac{1}{2^k}\left[2^{2k}\frac{3k+4}{6} + 2^k(k+3) - \frac{5}{3}\right] \\ &= 2^k\frac{3k+4}{6} + (k+3) - \frac{5}{3 \times 2^k} \\ &= h\frac{4+3\lg h}{6} + (3 + \lg h) - \frac{5}{3h} \\ &= (\frac{h}{2} + 1)\lg h + \frac{2}{3}h + 3 - \frac{5}{3h} \\ &\approx O(h \lg h) \end{aligned} \tag{3.23}$$

Based on the above calculations for convergence time and probe counts in hybrid search, we can now write the following theorems.

Theorem 3.3.1. *When detecting the optimal keep-alive interval using hybrid search technique in the range* $[1, h]$ *for some* $h = 2^k$, $k \geq 1$, *the number of tests performed monotonically increases with the binding timeout of middle-box. Probe count in best case is 1. In worst case, as well as on average, it is* $O(\lg h)$. □

Theorem 3.3.2. *When detecting the optimal keep-alive interval using hybrid search technique in the range* $[1, h]$ *for some* $h = 2^k$, $k \geq 1$, *the convergence time monotonically increases with the binding timeout of middle-box. The best case convergence time is 2. In worst case, as well as on average, it is* $O(h \lg h)$. □

3.4 Comparative Analysis of Search Techniques

With the analysis of previous sections, we can now compare the behavior of these search techniques. We will first compare the number of tests performed in each approach. Then we will compare the convergence time. Finally, we will comment on the significance of these features.

3.4.1 Probe Count Comparison

To begin with, we look at the number of tests performed by the search approaches side by side in Table 3.1. From the order statistics, it is clear that binary search and hybrid search are better than exponential search in terms of probe count on average. Both of the latter have an average probe count of $O(\lg h)$. In order to differentiate between the two, we can further look into the exact functions for the best, average and worst case convergence time. This is listed in Table 3.2. Clearly, binary search outperforms hybrid search on average and in worst case. In the best case scenario, however, hybrid search will need only 1 probe, which is less than the $\lg h$ probes taken by binary search.

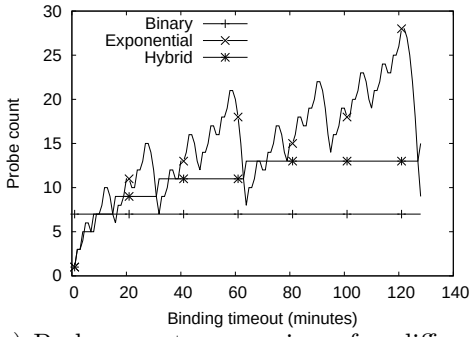
The probe count of these techniques has been compared graphically in Figure 3.8. Figure 3.8(a) shows the number of tests performed by different KA interval detection

Table 3.1: Probe count comparison among optimal KA detection techniques.

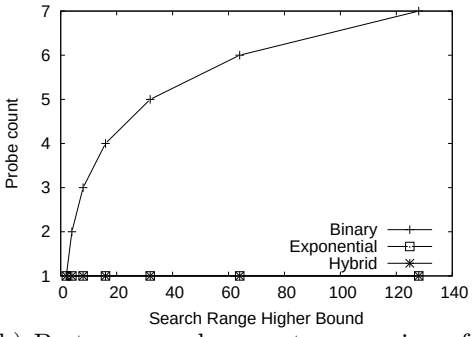
Search technique	**Best case** N	**Average case** N	**Worst case** N
Binary search	$O(\lg h)$	$O(\lg h)$	$O(\lg h)$
Exponential search	1	$O(\lg^2 h)$	$O(\lg^2 h)$
Hybrid search	1	$O(\lg h)$	$O(\lg h)$

Table 3.2: Probe count of binary search vs. hybrid search.

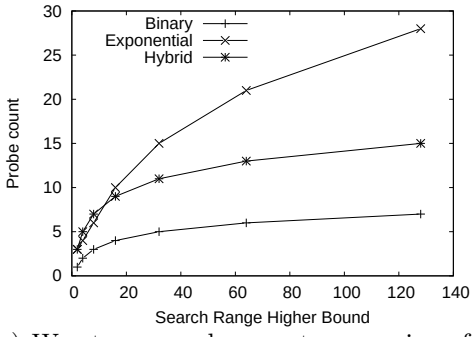
Search technique	**Best case** N	**Average case** N	**Worst case** N
Binary search	$\lg h$	$\lg h$	$\lg h$
Hybrid search	1	$2\lg h + \frac{2}{h}(2 + \lg h) - 3$	$1 + 2\lg h$



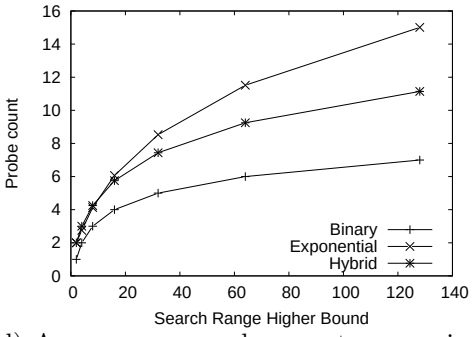
(a) Probe count comparison for different binding timeouts.



(b) Best case probe count comparison for different search ranges.



(c) Worst case probe count comparison for different search ranges.



(d) Average case probe count comparison for different search ranges.

Figure 3.8: Probe count comparison.

techniques for different binding timeouts in between 1 to 128 minutes. Both hybrid and exponential search needs fewer tests than binary search for smaller values of binding timeout. But, for larger binding timeouts, the number of probe count grows in these approaches, while that of binary search remains constant. For example, for hybrid search this cross over happens for $\alpha = 16$.

Figure 3.8(b) and 3.8(c) show respectively the probe count comparison in best and worst cases. The average case probe count is plotted in Figure 3.8(d).

3.4.2 Convergence Time Comparison

The convergence time of the different search approaches is shown side by side in Table 3.3. From the order statistics, it is clear that binary search and hybrid search are better than exponential search in terms of convergence time on average as well. Both of them have an average convergence time of $O(h \lg h)$. In order to differentiate between the two, we can further look into the exact functions for the best, average and worst case convergence time. This is listed in Table 3.4. Clearly, binary search outperforms hybrid search on average and in worst case. In the best case scenario, however, hybrid search will take less time.

Figure 3.9 shows graphically the convergence time comparison of the search approaches. For the binding timeouts in the range between 1 to 128 minutes, the time it takes for each search technique to detect the optimal keep-alive interval is shown in Figure 3.9(a). Much like the probe count, we observe that, for smaller binding timeouts, both hybrid and exponential search performs better than binary search. But, for larger values of α, convergence time of binary search is much less than the two. For hybrid search, this cross over happens for $\alpha = 32$. The curve for exponential search is not monotonic; it crosses the curve for binary search several times. For the most part, it remains at much higher than the curve for binary search along the Y-axis.

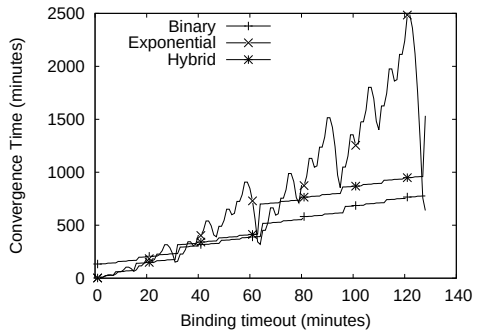
Figure 3.9(b) compares the best case convergence time of the search approaches, while

Table 3.3: Convergence time comparison among optimal KA detection techniques.

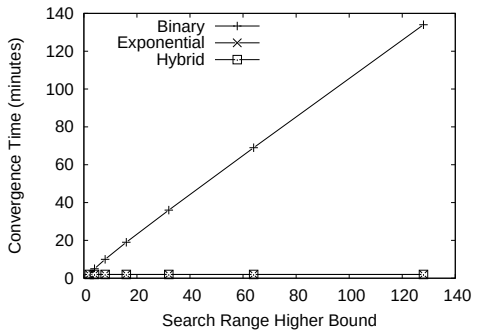
Search technique	**Best case** T	**Average case** T	**Worst case** T
Binary search	$O(h)$	$O(h \lg h)$	$O(h \lg h)$
Exponential search	2	$O(h \lg^2 h)$	$O(h \lg^2 h)$
Hybrid search	2	$O(h \lg h)$	$O(h \lg h)$

Table 3.4: Convergence time of binary search vs. hybrid search.

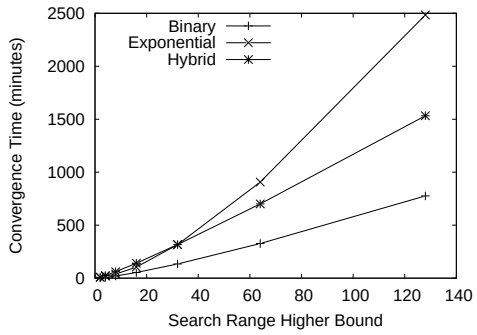
Search technique	**Best case** T	**Average case** T	**Worst case** T
Binary search	$h + \lg h - 1$	$(\frac{h}{2} + 1) \lg h$	$h \lg h + \lg h - h + 1$
Hybrid search	2	$(\frac{h}{2} + 1) \lg h + \frac{2}{3}h + 3 - \frac{5}{3h}$	$h \lg h + 5h - 3$



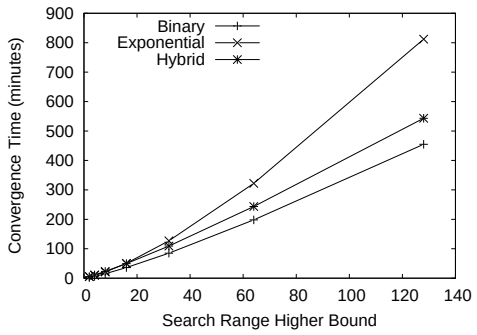
(a) Convergence time comparison for different binding timeouts.



(b) Best case convergence time comparison for different search ranges.



(c) Worst case convergence time comparison for different search ranges.



(d) Average case convergence time comparison for different search ranges.

Figure 3.9: Convergence time comparison.

Figure 3.9(c) compares the worst cases. The average convergence time of each technique is plotted together in Figure 3.9(d). As discussed above, it is also visible in these plots that binary search has the smallest convergence time, on average.

3.4.3 Discussion

As pointed out in the earlier section, binary search clearly has the better convergence time. Long convergence time of exponential and hybrid search make them susceptible to transient network failures and as such not being able to complete the testing. This is especially important when the user is moving from one network to another frequently. However, the search algorithm should cache the improved keep-alive interval in the persistent store. Thus, the algorithm can resume with a reduced space to search on by retrieving state from the persistent store. For all the search techniques, following data need to be persisted: the *largest keep-alive interval that is already known to work* and the *smallest keep-alive interval that is already known not to work*. For exponential search, the *keep-alive interval currently being tested* need to be stored as well. For hybrid search, another piece of information that should be stored is the *current phase (binary/exponential) of search*.

Another point on convergence time is that it does not have any impact of the power consumption on the end user's device. There is no expensive computation happening on the device due to longer convergence time. It however has some impact on the service. The service needs to ensure that it is capable of handling all the test connections from millions of devices. A port on the server machine is occupied to maintain the test connection until testing has converged to the optimal interval. However, once again, there is no expensive bandwidth or CPU consumptions in the test connections. So, we expect the impact on the service is also minimal.

We have compared different techniques to detect optimal keep-alive interval of a TCP connection. We have used convergence time and the probe count as two metrics in com-

paring these techniques. No doubt, these are important metrics, but are these the only or the most important metrics? As noted in Chapter 1, the goal of detecting optimal keep-alive interval is to reduce the number of keep-alive packets we need to send to keep the data connection active. Once we detect it, we can send keep-alive packets using that period. But, when we haven't reached the optimal interval in testing yet, at what frequency would we send the KA packets? When we select an interval to test, if that test is successful, we now have knowledge about a keep-alive interval that works and is longer than what we knew before. As such, this updated interval can be used as the keep-alive period in the data connection. This process is repeated every time a better KA interval is detected through testing. Therefore, a detection technique that can quickly improve the KA interval of the data connection is very preferable. We think the most important criteria in comparing these techniques is the answer to the following question: over a span of time, which detection technique results in the minimum number of keep-alive packets sent over the data and the test connection combined? We have not obtained a theoretical function to represent this value. However, in the next chapter, we will obtain this through simulation and compare the search techniques accordingly.

Chapter 4

Simulation Results

We have implemented the different techniques to detect the optimal keep-alive interval on a simulation platform. In this chapter, we present and analyze the data from that simulation. Through simulation, we study the behavior of these techniques based on some metrics.

4.1 Simulation Setting

We used Omnet++ as our simulation platform [8], which is a C++ based discrete event simulator. In all our simulation, the delay from a node to the middle-box (NAT, Firewall, Proxy server etc.) is set to 10ms. When connected to the network of the department of CSE, BUET through different access points, we executed ping command to the default gateway. The round trip time (RTT) to the gateway was always less than 20ms. Based on this simple experiment, we chose the 10ms delay in one direction between a node and a middle-box. Similarly, the RTT of pings to different prominent services like google.com, yahoo.com etc. from CSE, BUET network has been used in setting the delay in one direction between the middle-box and the server. The delay was set to 100ms. Results of our experiment do not depend on these exact values of delays. As long as these delays

are much less than 1 minute, our results will continue to hold.

For each experiment, a separate test connection was used for searching, while a data connection was maintained using keep-alive packets. The keep-alive packets in both test and data connections were sent on minute boundaries. The lower bound of the search space was set to 1 minute. This also means that data connection sends a keep-alive packet every minute when no testing is performed, or when no improved interval has yet been reported from testing.

Binary search technique also requires a higher bound of the search space to be specified. In our experiments, this was set to 128 minutes. This is slightly higher than the IETF recommended timeout of 124 minutes [21] for any TCP connection. Therefore, the best possible keep-alive interval we could achieve is 128 minutes. The binding timeout of the middle-box was varied in the range $[2, 129]$.

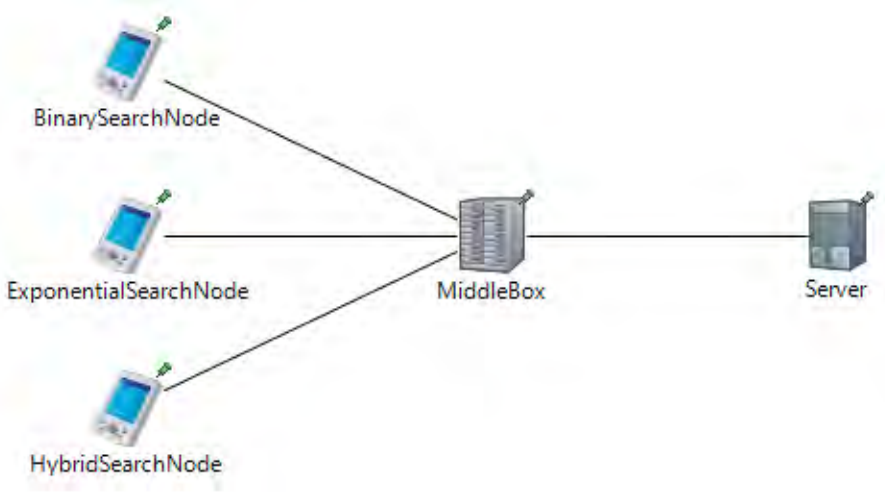


Figure 4.1: Simulation topology.

The topology of the simulation setup is shown in Figure 4.1. Here 3 end user devices or nodes are connected to a server via a middle-box. The devices are named after the search technique they employ in detecting the optimal KA interval. The devices do not communicate with each other or interfere with each other in any ways.

Now that the basic setup is understood, we can jump into the different experiments we

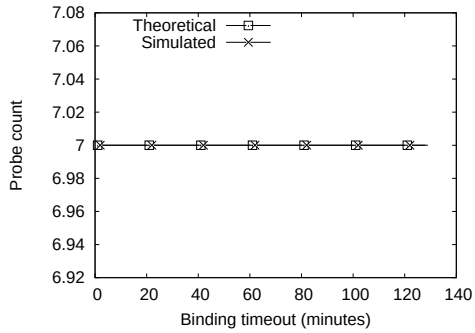
conducted and analyze their results. Firstly, we introduced the link delay and observed how it impacted the probe count and convergence time of each technique. Then we experimented on relaxing the optimality requirement on the keep-alive interval to shave off some portion of convergence time. To reduce convergence time, we additionally experimented with some variations of hybrid search. We also ran the experiments over specific amount of times to compare the number of keep-alives sent through the data connection, while testing was in progress using the different approaches.

Afterwards, we introduced failures in the application level packets. This was to observe the behavior of the search techniques in presence of transient network failures. We proposed some modifications to cope with packet failures and evaluated their performance. Finally, with the modified techniques, we again ran the experiments over specific amount of times to compare the data connection keep-alive counts over specific time spans. In the sections that follow, we provide details of the different experiments.

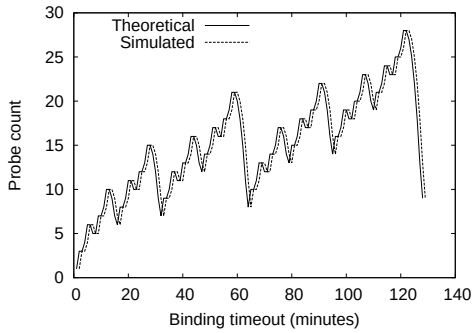
4.2 Impact of Delay

Figure 4.2(a) 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.

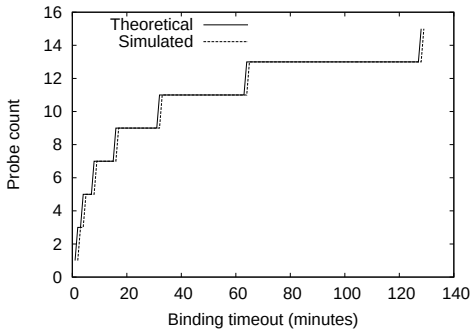
Figure 4.2(b) plots the probe count of exponential search as a function of binding timeout. Observe that the experimental curve is off from the theoretical curve by 1 minute. That is, the observed probe count for any α is equal to the theoretical probe count for $\alpha - 1$. This is due to network delay 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, a keep-alive interval of α does not work. From its perspective the binding timeout is $\alpha - 1$. As such, the observed



(a) Binary search.



(b) Exponential search.



(c) Hybrid search.

Figure 4.2: Probe count of different search techniques in presence of network delay.

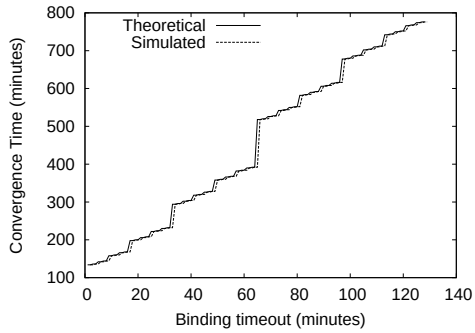
probe count matches with the theoretical probe count for $\alpha - 1$.

Figure 4.2(c) plots the probe count of hybrid search as a function of binding timeout. Like we have observed earlier, the hybrid curve too is off from the theoretical curve by 1 minute.

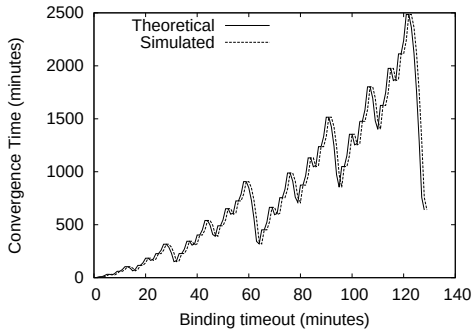
Figure 4.3(a) plots the experimental as well as theoretical convergence time curve for binary search. Here too, the experimental curve is off from the theoretical curve by 1 unit along the Y-axis. The same behavior is observed for exponential and hybrid search, as can be seen from Figure 4.3(b) and Figure 4.3(c) respectively.

4.3 Reducing Convergence Time

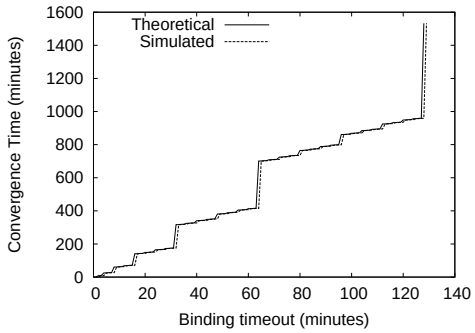
As can be seen from theoretical analysis, as well as the previous experiment, it takes quite a bit of time to detect the optimal KA interval, if the binding timeout of the middle-box



(a) Binary search.



(b) Exponential search.



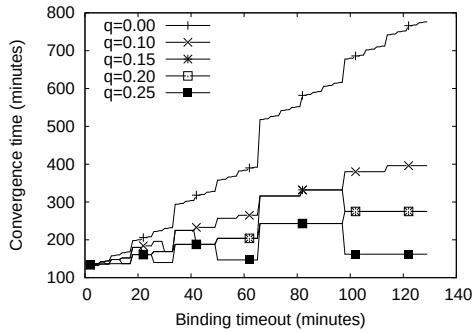
(c) Hybrid search.

Figure 4.3: Convergence time of different search techniques in presence of network delay.

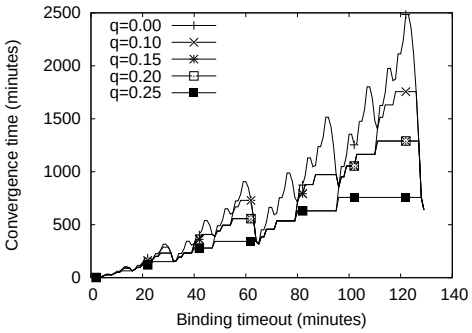
is large. As we discussed earlier, the long convergence time should not have any impact on power consumption of the end user device. However, it will have some impact on the server. If the convergence time could be reduced, then the test connection could be released sooner, which would release a port on the server. It is always possible to design the service with appropriate load balancing techniques in place to cope with the high convergence times. We will not be focusing on the service design in this scope. Instead, we will show that if there is a need to reduce load on the service by shaving off some portion of the convergence time, this can be achieved by tuning several parameters in the detection algorithm.

4.3.1 Settling to Sub-optimal Keep-alive Interval

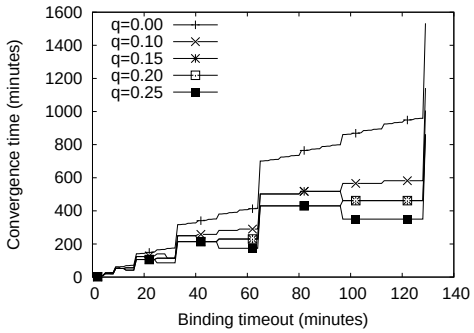
So far, we have always focused on reaching the optimal keep-alive interval. In this section, we show that we can reduce the convergence time by sacrificing accuracy or optimality of



(a) Binary search.



(b) Exponential search.



(c) Hybrid search.

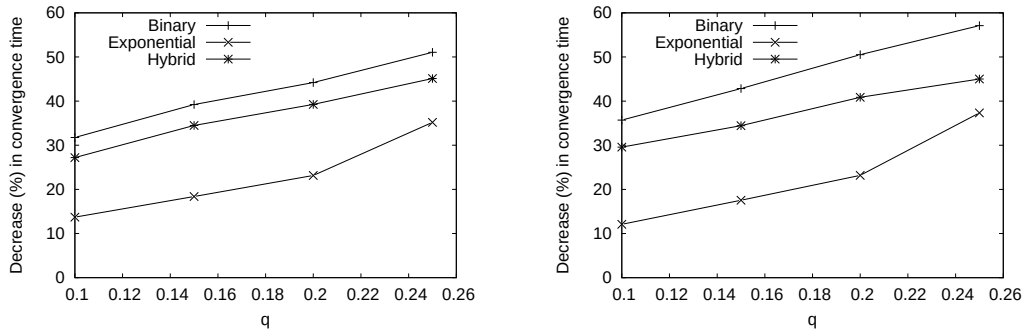
Figure 4.4: Impact of relaxing optimality of α on convergence time of different search techniques.

the converged interval.

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: $\alpha' \geq (1-q)\alpha$. In each of Algorithms 1, 2, 3, the testing continues as long as $high - low \geq 1$. Let us modify this constraint so that the testing continues 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



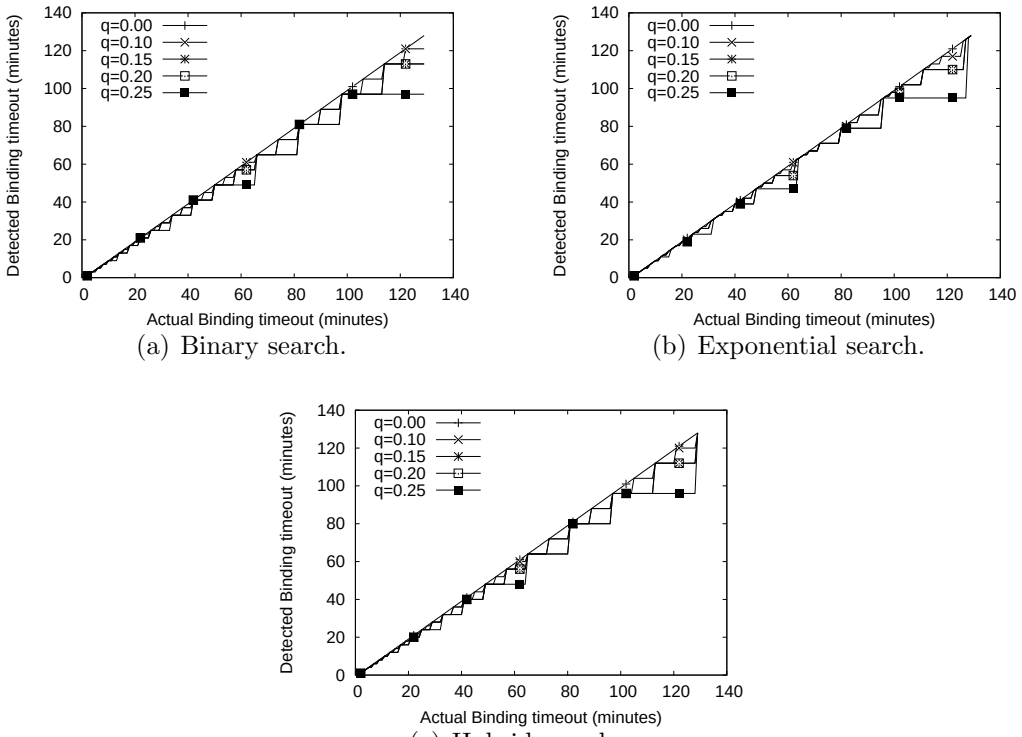
(a) Average decrease (%) in convergence time.

(b) Median decrease (%) in convergence time.

Figure 4.5: Average and median decrease in convergence time due to relaxing optimality of α.

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. The results are shown in Figure 4.4 and Figure 4.5. Binary search has the most reduction in convergence time with admittance of error in the detected keep-alive interval. The average decrease in convergence time in both binary and hybrid search approach shows a linear relationship with q. Exponential search shows less reduction in convergence time; about one-third of reduction observed in binary search. However, for $q \geq 0.2$, it exhibits increased amount of reduction in convergence time.

Figure 4.6 shows the keep-alive interval detected by different search techniques for different values of q. Observe the step pattern in each of the plots. This is due to the fact that for several contiguous values of α, the algorithm settles to the same keep-alive interval. For each α, we computed the percentage decrease in the detected binding timeout, compared to α. Then we took the average and median in the range $[1, 128]$. We performed this computation for different values of q. The resulting plots are shown in Figure 4.7. The reduction in the keep-alive interval is very close in binary and hybrid searches and shows a linear relationship with respect to q. Exponential search has almost 50% less reduction in the keep-alive interval, when compared with hybrid and binary



(a) Binary search.

(b) Exponential search.

(c) Hybrid search.

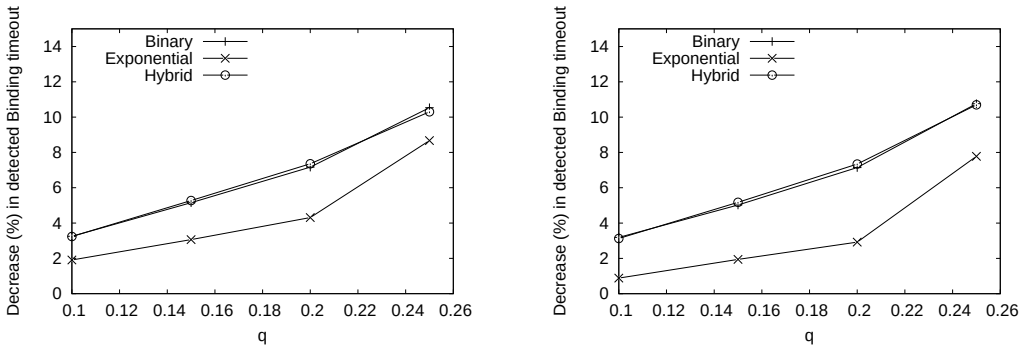
Figure 4.6: Impact of relaxing optimality of α on detected binding timeout in different search techniques.

search techniques. Therefore, we could potentially use higher values of q with exponential search to further improve on its convergence time.

It is not necessary to hard code the value of q into the algorithm. Rather, it could potentially 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.

4.3.2 Variations of Hybrid Search

In this section, we perform experiments to demonstrate another approach to reduce convergence time. This approach is applicable to hybrid search only. Recall that hybrid search acts like exponential search initially. When a tested interval overshoots the actual binding timeout, the test connection gets terminated. Subsequently, binary search is per-



(a) Average decrease (%) in detected binding timeout.

(b) Median decrease (%) in detected binding timeout.

Figure 4.7: Average and median decrease in detected binding timeout due to relaxing optimality of α.

formed in the new search range. So, the transition to binary search happens upon the first failure. We experiemented with a variation of hybrid search, where we introduced another case for transitioning to the binary search phase. That is, 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 of hybrid search for different values of r is shown in Table 4.1. Here, the search technique name has been annotated with the value of r used. In addtion, the convergence time of binary search and standard hybrid search is also shown in the table for reference. Clearly, by tuning r, the convergence time can

Table 4.1: Average convergence time in variations of hybrid search. The average is taken over the search range of $[1, 128]$ minutes.

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

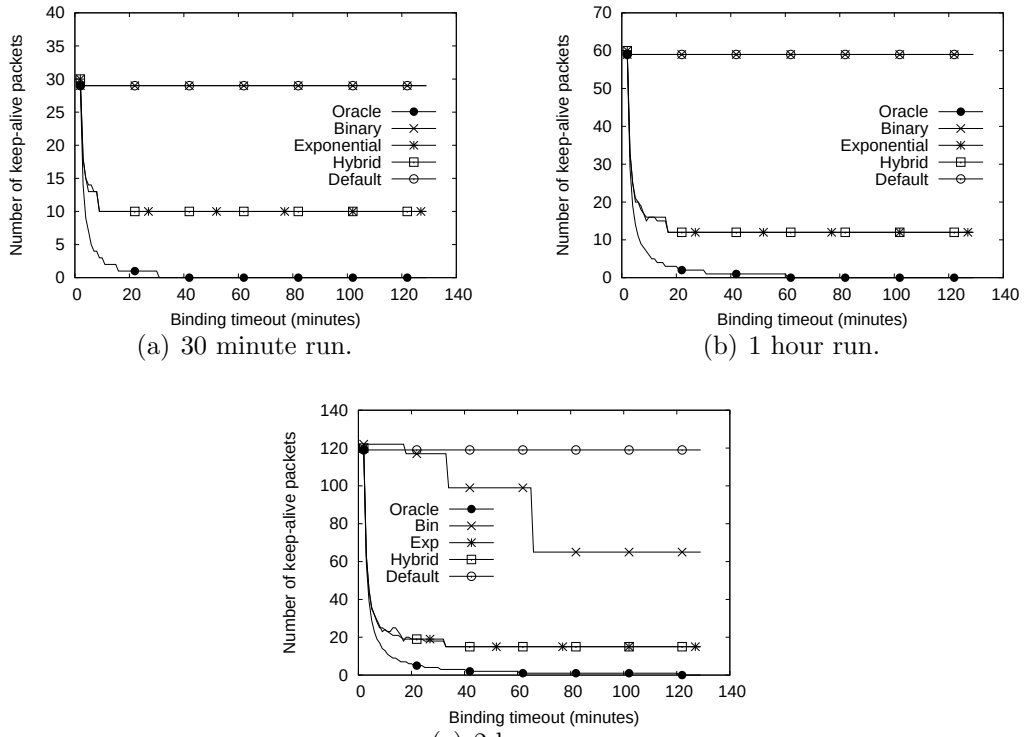
be reduced considerably. Much like the parameter q, parameter r can also be tuned dynamically from the server.

4.4 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. The time durations are chosen to reflect some real world scenarios.

Firstly, we performed a 30 minute and a 1 hour run. These are typical durations of meetings in any organization. One may go to a different building for a meeting, that belongs to a different APN than his own office building. It could also be the case that the meeting room is in the same building but connects to internet through a different gateway than his own office room. As such, even if the device has already detected the optimal keep-alive interval from his office room, it will need to calculate it for this new environment.

Another example could be a student attending an hour long class in different buildings in the campus. Since graduate classes can be longer than a hour, we also simulated a 2 hour run. This duration also matches duration of 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 4.8. 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 4.8(a) and Figure 4.8(b). This is because the wait time for sending the first probe over the test connection is longer (65



(a) 30 minute run.

(b) 1 hour run.

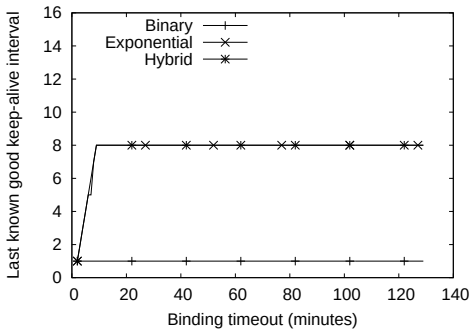
(c) 2 hour run.

Figure 4.8: Number of keep-alive packets sent during meeting or seminar scenario.

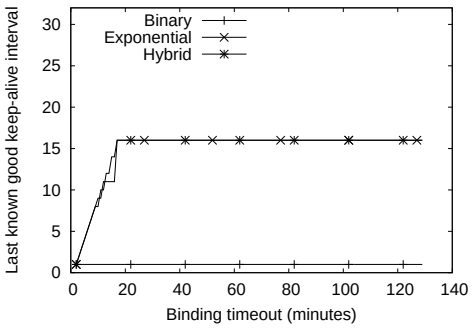
minutes) than the duration of the scenario. As such, no improvements could be offered by the binary search technique. Figure 4.8(c) shows reduction in number of keep-alive sent using binary search technique. The duration of this run permitted one or more probes, which results in improved keep-alive interval. The updated keep-alive interval, at the end of each time span is shown in Figure 4.9.

Let us take a look at the behavior of hybrid and exponential search, on the contrary, in Figure 4.8. 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 4.8(c).

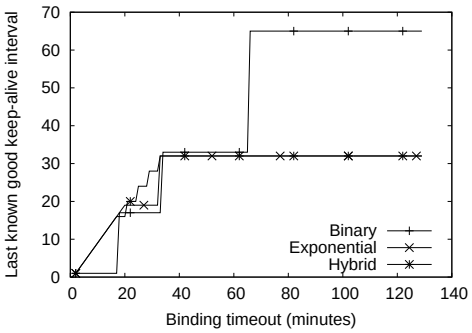
The updated keep-alive interval at the end of each time span, as seen in Figure 4.9 is also identical for these approaches. In Figure 4.9(a) and Figure 4.9(b), hybrid and



(a) 30 minute run.



(b) 1 hour run.

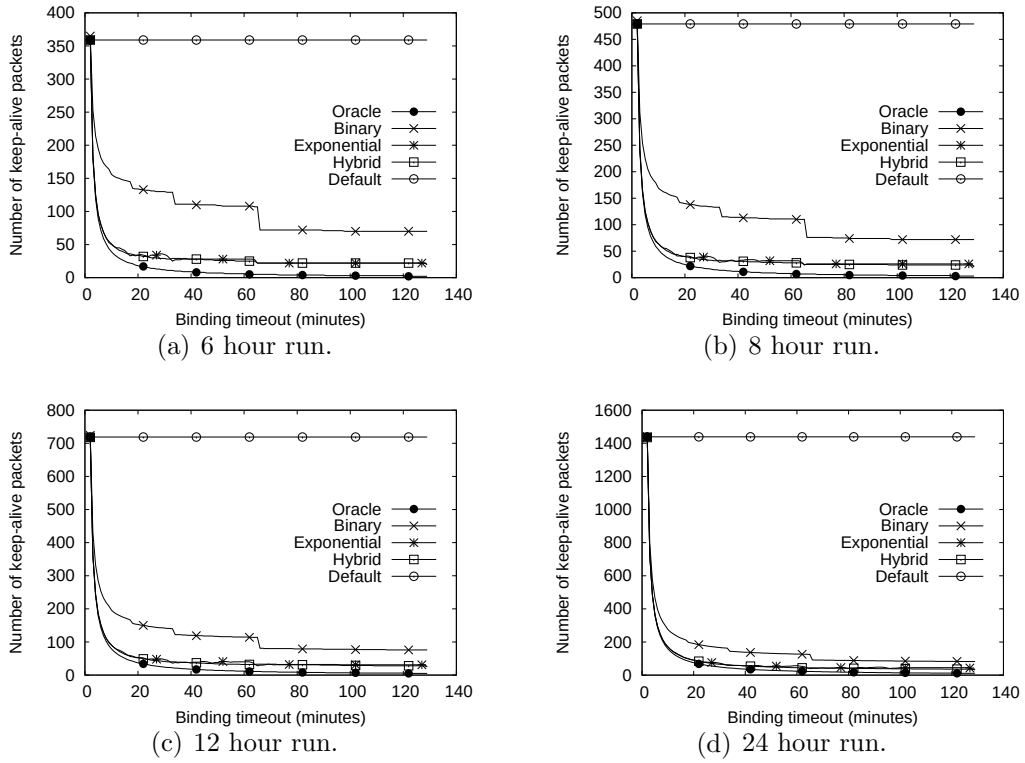


(c) 2 hour run.

Figure 4.9: Last known good keep-alive interval during meeting or seminar scenario.

exponential techniques improved the KA interval while binary search failed to report any improvements. In Figure 4.9(c), however, binary search reported a better KA interval compared to the other techniques, when binding timeout was greater than 33 minutes. Despite the better KA interval, binary search could not reduce the overall number of KA packets sent. This is because this approach reports large improvements, but long time apart. Hybrid and Exponential searches, on the other hand, report improvements in small amount initially. This immediate feedback starts reducing the number of KA packets sent over the data connection quickly.

Next, we experimented with 6 and 8 hour runs. The motivation for these durations is as follows: On a Friday night (last working day), you go to a resort to spend the weekend. You arrive at the resort at night. After sleeping through the night, you will engage in fun activities from the next morning. Since this is a new place, your mobile phone (or other hand held device capable of connecting to internet) will need to start testing for the optimal keep-alive interval to the notification service. We expect the duration of your



(a) 6 hour run. (b) 8 hour run.

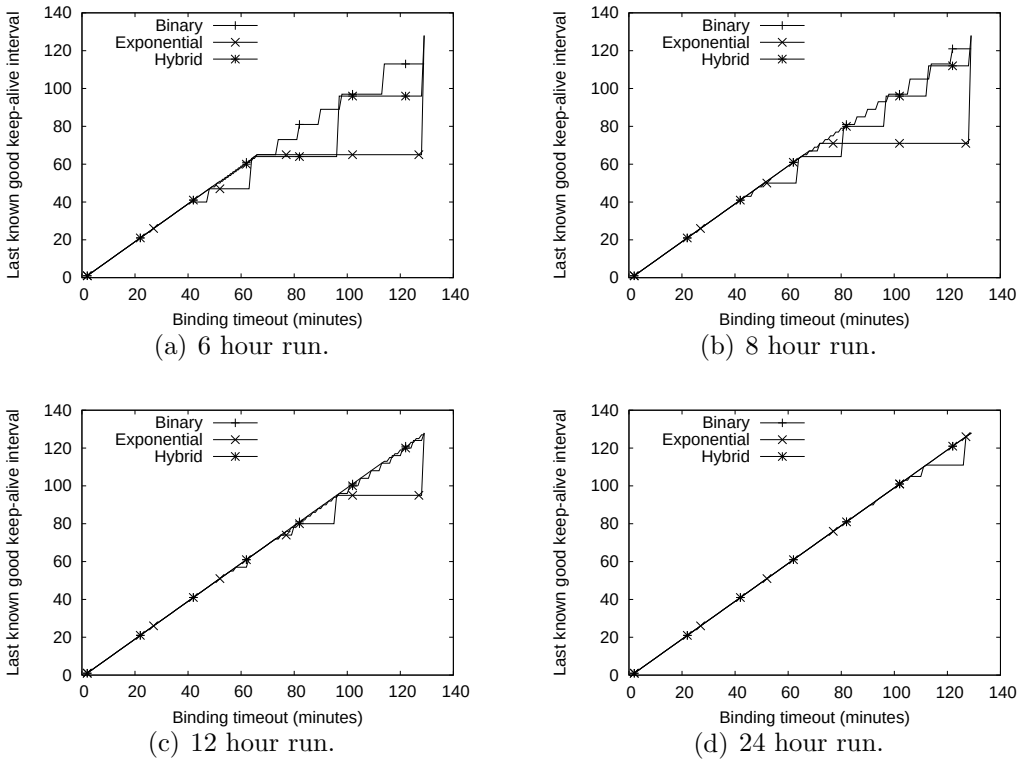
(c) 12 hour run. (d) 24 hour run.

Figure 4.10: Number of keep-alive packets sent over 1 data and 1 test connection over longer durations.

sleep to be 6 to 8 hours. What is the number of keep-alives sent during this time?

In addition, we experimented with longer durations: 12 hours and 24 hours. The number of keep-alives sent in all these cases, for the different search techniques are shown in Figure 4.10. In all these runs, hybrid search and exponential search curves are almost identical and they approach the Oracle curve with increasing study durations. The number of keep-alives sent is highest in the binary search approach. However, in the 24 hour run, the curve for binary search was adjacent to the other curves.

Figure 4.11 shows what the keep-alive interval was at the end of the study durations. Observe the step pattern in each of these curves. Based on the test sequence, we may reach the optimal keep-alive interval soon, but we may not realize that we reached it. Subsequently, we spend some time to identify that no other interval greater than the current one would work. For hybrid and exponential search, this happens for observed



(a) 6 hour run. (b) 8 hour run.

(c) 12 hour run. (d) 24 hour run.

Figure 4.11: Last known good keep-alive interval over longer durations.

binding timeouts that are power of 2. For example, in the 6 hour run when binding timeout of middle-box is 129 minutes, the observed binding timeout is 128 minutes (recall the impact of delay). Both hybrid and exponential search will report this KA interval, after testing for around 4 hour 15 minutes. But, in exponential search we would not know that this is the optimal value until about 6 hours later! In case of hybrid search, we would have to wait for 21 more hours, after detecting this KA interval! But, much much before these waits, we realize that this interval does work and are able to apply it to reduce the KA count of data connection. On the other hand, if the observed binding timeout was in the range [65, 127] minutes instead, the best keep-alive interval exponential search can report within 6 hours of testing is 65 minutes. For hybrid search this would be 96 minutes if the observed binding timeout was in the range [96, 127], or 65 minutes if it was in the range [65, 95]. This is why we observe the step patterns. The steps in the binary search curve are much more granular compared to the others. Exponential search exhibits the largest steps.

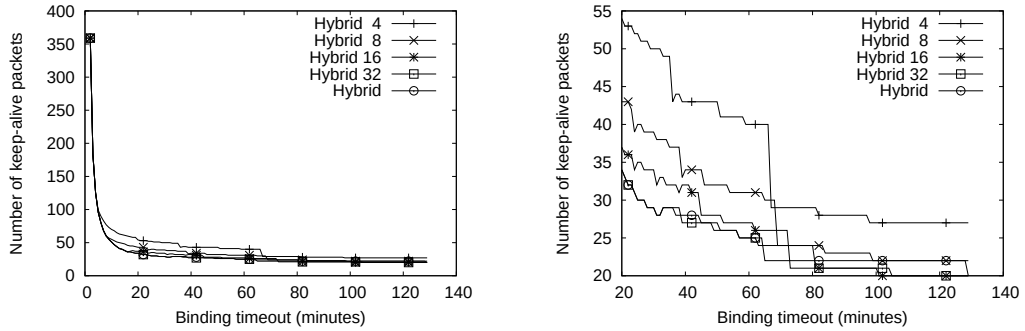
Table 4.2: Average reduction (%) of number of keep-alives sent when testing is performed to improve the keep-alive interval from the conservative default value.

Experiment duration (Hours)	**Binary search**	**Exponential search**	**Hybrid search**
0.5	0	64.30	64.25
1	0	77.79	77.83
2	26.79	84.92	85.05
6	71.62	90.88	91.06
8	77.52	91.74	92.09
12	83.47	92.84	93.13
24	89.61	94.12	94.39

In the 12 hour run, binary search reached the optimal interval for all binding timeouts. Hybrid search reached the optimal interval for most of the binding timeouts. In case of 24 hour run, both binary and hybrid searches reached the optimal interval. In this run, exponential search too reached the optimal interval for most values of binding timeouts. For some large binding timeouts, however, it did not reach the optimal interval even in 24 hours.

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 4.2. In each case, hybrid and exponential results almost exactly the same amount of improvement; and they perform better than binary search. For larger durations (24 hours), binary search performs very close to the other two. For durations around and less than 12 hours, results of hybrid and exponential search are much better than that of binary search. For durations less than an hour, binary search is unable to reduce the number of KA packets sent at all. The smaller duration runs will be more common in real usage patterns. It is unlikely that a mobile device remains in the same APN or behind the same gateway for long periods. Rather it moves from one APN to another, staying

in each for short or moderate amounts of times. As such, exponential and hybrid search techniques should be preferred to binary search. Since hybrid search has much better convergence time than exponential search, it is fair to say that hybrid search is the best choice.



(a) Here, the keep-alive interval search range is 1-128 minutes.

(b) For the same search range, here the curves are zoomed in the range 20-128 minutes.

Figure 4.12: 6 hour run of variations of hybrid search.

Figure 4.12 shows the performance of variations of hybrid search during a 6 hour run. From the zoomed graph in Figure 4.12(b), we can see that for higher values of r, performance is better. When $r = 32$, the average reduction (%) of number if keep-alives sent is, in fact, slightly better than the standard hybrid search. The results for other run durations are listed in Table 4.3. From these data, we see that by tuning r, we can reduce convergence time significantly, with very little or no adverse impact on performance in terms of number of KA packets sent.

4.5 Impact of Packet Failure

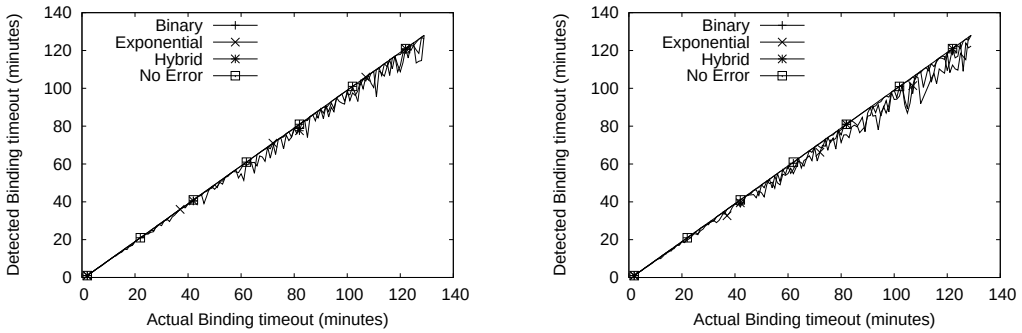
So far, when a keep-alive message fails over the test connection, we have assumed that the middle-box has dropped the connection due to too long an idleness. However, a packet could also be dropped for some transient issues in the network. But, when the keep-alive message fails, the sender has no way to differentiate between the different causes

Table 4.3: Average reduction (%) of number of keep-alives sent when applying variations of hybrid search to improve the keep-alive interval from the conservative default value.

Experiment duration	**Hybrid 4**	**Hybrid 8**	**Hybrid 16**	**Hybrid 32**	**Hybrid**
0.5 hours	61.26	64.25	64.25	64.25	64.25
1 hour	68.64	76.60	77.83	77.83	77.83
2 hours	75.77	82.23	84.31	85.05	85.05
6 hours	87.87	89.99	90.89	91.16	91.06
8 hours	89.69	91.29	91.87	92.07	92.09
12 hours	91.58	92.66	93.05	93.16	93.13
24 hours	93.66	94.20	94.39	94.44	94.39

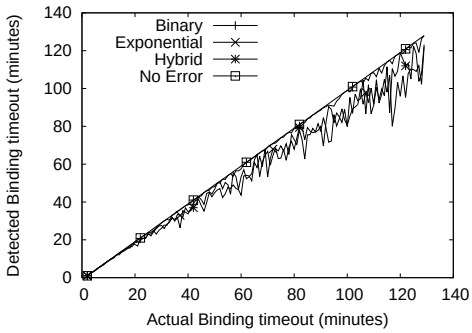
of failure. As such, the algorithm we developed thus far will settle for a sub-optimal keep-alive interval if a keep-alive message fails due to transient network failure.

In this experiment, we try to simulate transient network failures and observe the behavior of the different search techniques. 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 4.13 and Figure 4.14 plots the detected binding timeout against the actual binding timeouts. The curve marked as 'No Error'

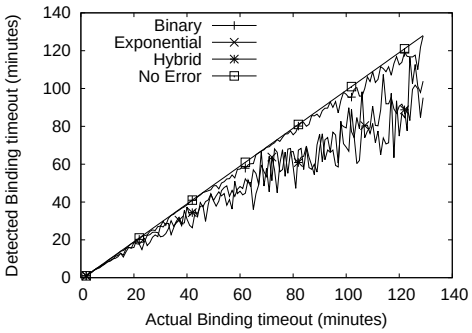


(a) $p = 0.005$. (b) $p = 0.01$.

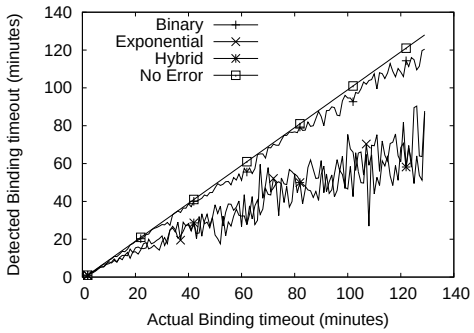
Figure 4.13: Comparison of detected binding timeout among different search techniques, when no more than 1% of keep-alive messages may fail due to transient network failure.



(a) $p = 0.02$.



(b) $p = 0.05$.



(c) $p = 0.10$.

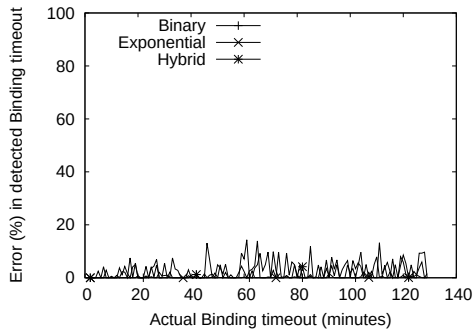
Figure 4.14: Comparison of detected binding timeout among different search techniques, when $0.01 < p \leq 0.10$.

represents the detected binding timeout when there is no error involved. Recall that, this is 1 minute less than the middle-box's binding timeout, due to propagation delay.

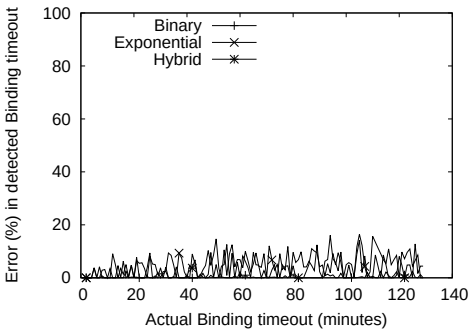
From these curves, it is clear that, when $p > 0.01$, the impact cannot be neglected. The percentage reduction in the detected binding timeout is plotted in Figure 4.15. This further strengthens our observation.

To better understand the impact of packet failure on each KA detection technique separately, we present the results for each technique independently as well. Figure 4.16 shows the impact of packet failure on binary search. The average percentage error in detected binding timeout is less than 5%, even when up to 10% keep-alive packets may be lost.

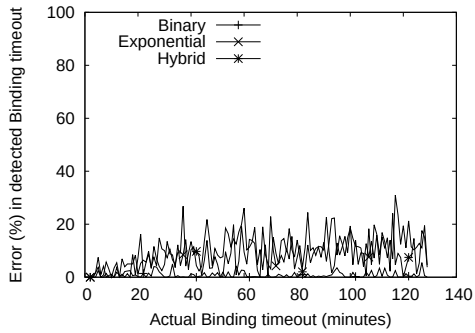
On the other hand, in Figure 4.17, we can see that when exponential search is used, almost 40% error can be introduced in the detected binding timeout, on average, if $p =$



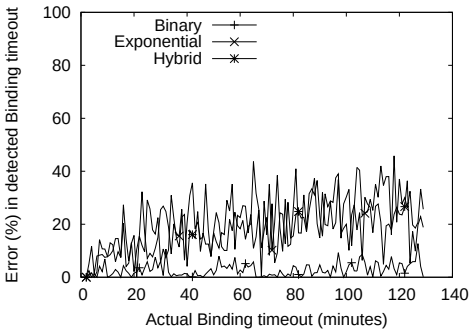
(a) $p = 0.005$.



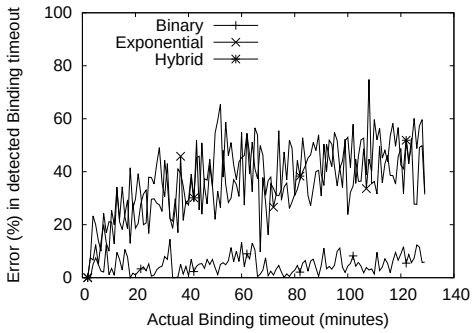
(b) $p = 0.01$.



(c) $p = 0.02$.



(d) $p = 0.05$.



(e) $p = 0.10$.

Figure 4.15: Error percentage in detected binding timeout among different search techniques.

0.10. Hybrid search suffers similarly in presence of keep-alive packet failures.

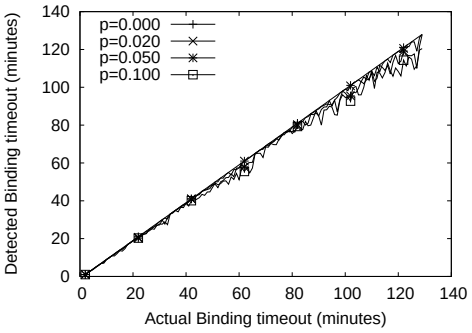
The average percentage decrease in detected binding timeouts, due to packet failures, is shown in Figure 4.19(a). In each search technique, the curve is linear. However, the slope of the exponential and hybrid curves are much stiffer, compared to that of binary curve. Similar behavior is observed in Figure 4.19(b), which plots the median, instead of average, percentage decrease.

To summarize the results of this section, we can say, packet failure impacts all the search techniques, specially when $p > 0.01$. Binary search technique, however, is the least impacted. Since the impact is considerable, we need to propose a mitigation and analyze its performance. We do that in the next section.

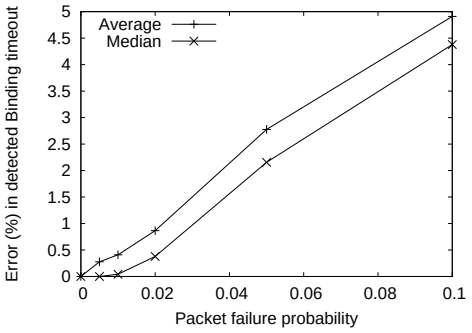
4.6 Retry on Packet Failure

We modified each algorithm as follows: When keep-alive message fails, we cannot automatically assume that it failed due to exceeding the optimal keep-alive interval. It could well have been due to some transient failure. Therefore, we re-establish the TCP connection and test the same interval again. That means, we will again wait for the same amount of time, then send a keep-alive message and check if it was successfully sent or not. 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. If, on the other hand, the second attempt succeeds, we conclude that the earlier attempt failed due to transient error and we have now found that the current test interval does work. And we proceed to test the next possible interval.

After implementing this modification, we re-ran the same experiment that we did to understand impact of packet failure earlier. The results are shown in figure 4.20 and Figure 4.21. For $p \leq 0.01$, the curve for detected binding timeout for each of the techniques has converged back to the curve for no packet failure. The effect of packet failure has

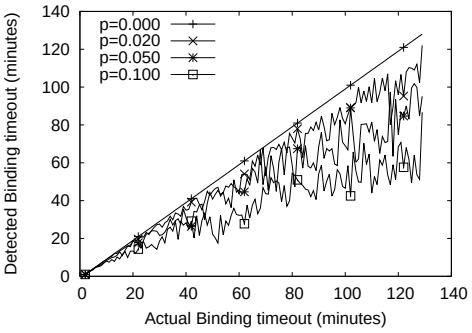


(a) Detected binding timeout.

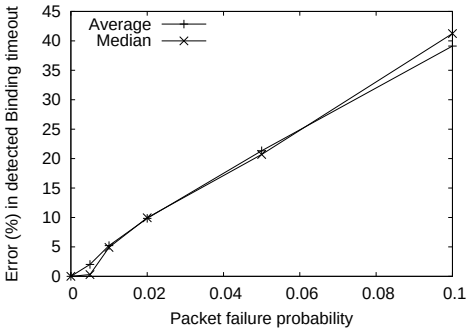


(b) Average and median error (%) in detected binding timeout.

Figure 4.16: Impact of packet failure on binary search technique.

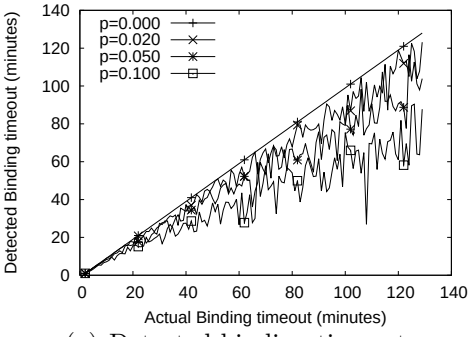


(a) Detected binding timeout.

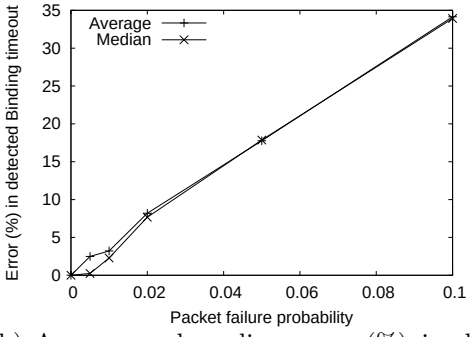


(b) Average and median error (%) in detected binding timeout.

Figure 4.17: Impact of packet failure on exponential search technique.

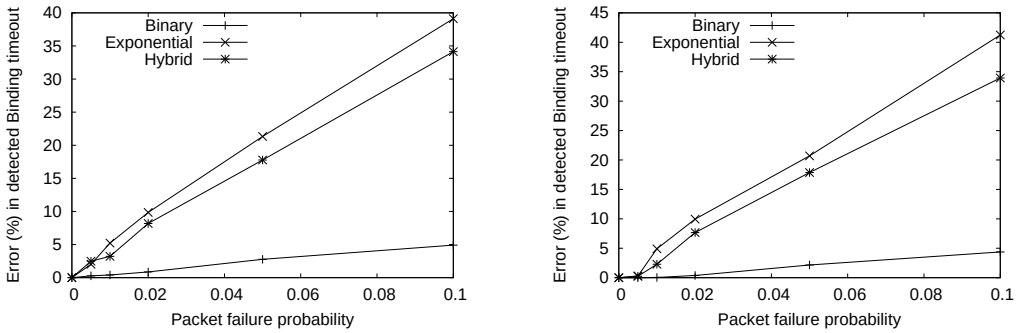


(a) Detected binding timeout.



(b) Average and median error (%) in detected binding timeout.

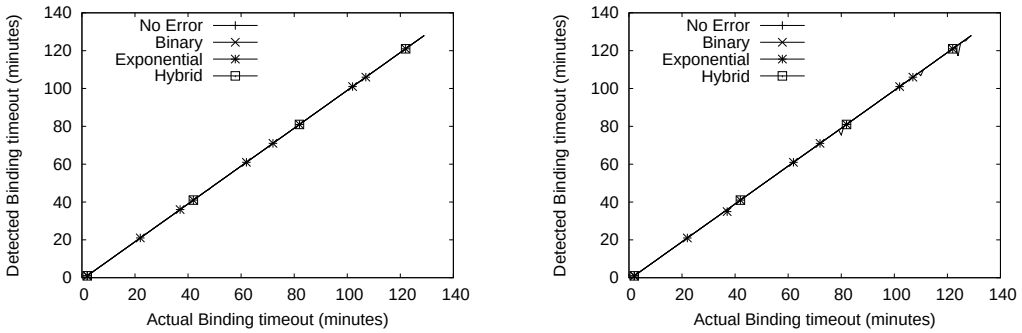
Figure 4.18: Impact of packet failure on hybrid search technique.



(a) Average error (%) in detected binding timeout.

(b) Median error (%) in detected binding timeout.

Figure 4.19: Average and median error percentage in detected binding timeout in different search techniques in presence of packet failure.



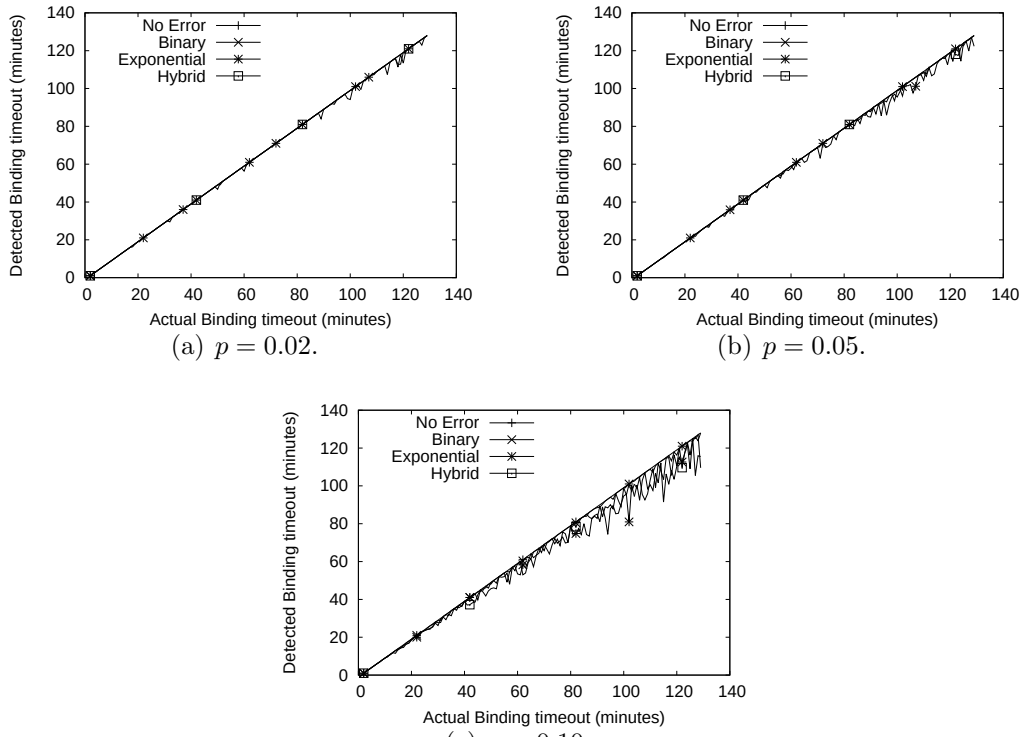
(a) $p = 0.005$.

(b) $p = 0.01$.

Figure 4.20: Comparison of detected binding timeout among different search techniques with retry, when $p \leq 0.01$.

completely been neutralized by single retry. For $p = 0.02$ and $p = 0.05$, the error in the detected binding timeout is also negligible. For $p = 0.10$, however, we observe error that is worth further investigation. We would therefore look at the behavior of each technique separately for $p = 0.10$.

Figure 4.22 demonstrates that the retry scheme in binary search approach is quite successful in coping with transient network issues. Figure 4.22(a) only plots the detected binding timeout curve for the no error case and the case when $p = 0.10$. There is very little deviation between the two. In fact, from Figure 4.22(b), we see that on average the error in detected binding timeout is less than 0.5% even when 10% of the keep-alive packets may fail. Therefore, the single retry scheme works satisfactorily in the binary



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

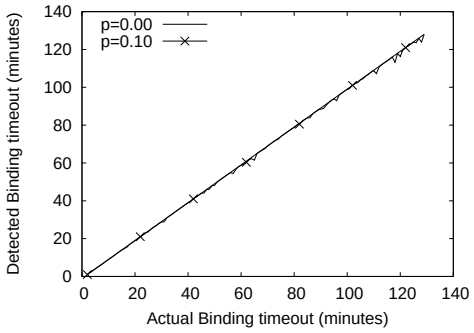
(c) $p = 0.10$.

Figure 4.21: Comparison of detected binding timeout among different search techniques with retry, when $0.01 < p \leq 0.10$.

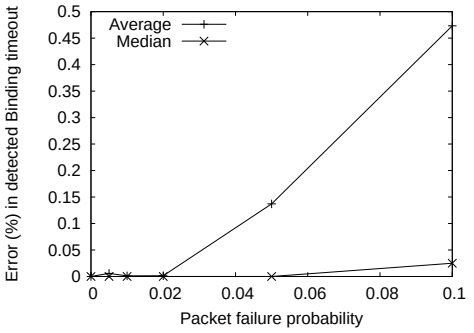
search case.

Figure 4.23 shows the impact of packet failure on exponential search technique with retry. The impact is much higher, compared to binary search. However, it is still within acceptable limits. On average, the error introduced in the detected binding timeout is around 5%, when $p = 0.10$. For hybrid search, on the other hand, it is around 3.5%, as can be seen from Figure 4.24 While the error is not negligible, we think this is within tolerable limit. If, however, there is a need to further reduce this error, multiple retries should be implemented. However, note that this can increase the convergence time significantly, which can cause adverse impact on the service.

Figure 4.25 shows the adverse impact of retry on convergence time of each search technique. In these graphs, the curve termed 'No retry' represents the convergence time when no retry scheme is implemented and there is no packet failure. The '$p = 0.00$' curve

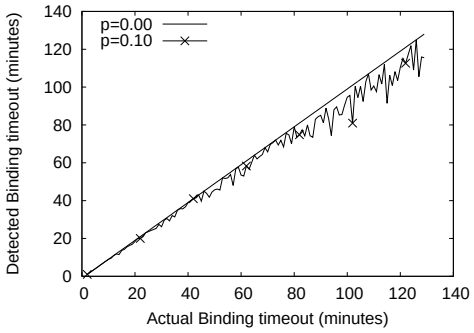


(a) Detected binding timeout.

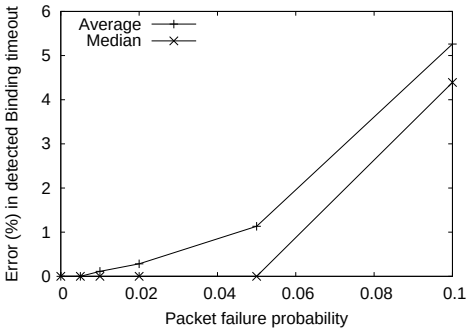


(b) Average and median error (%) in detected binding timeout.

Figure 4.22: Impact of packet failure on binary search technique with retry.

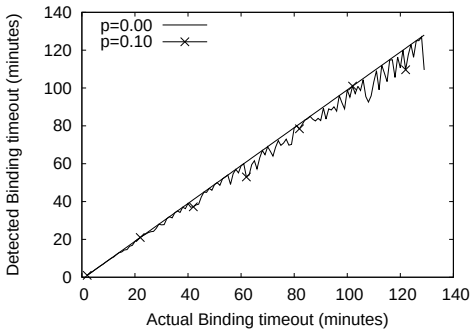


(a) Detected binding timeout.

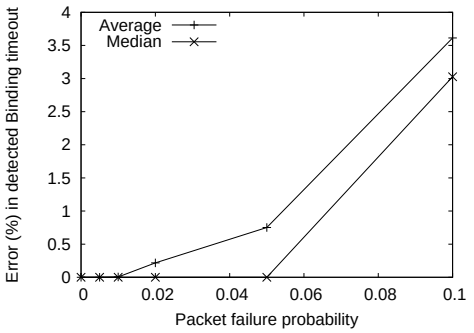


(b) Average and median error (%) in detected binding timeout.

Figure 4.23: Impact of packet failure on exponential search technique with retry.

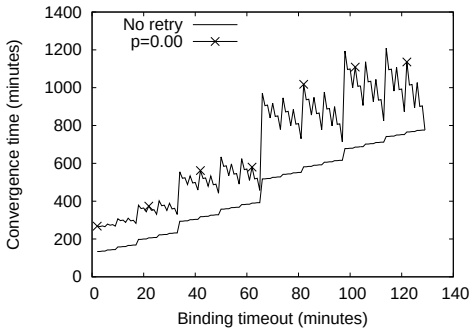


(a) Detected binding timeout.

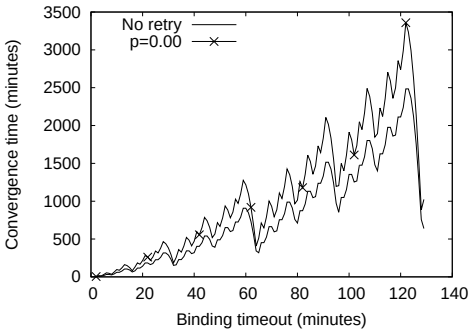


(b) Average and median error (%) in detected binding timeout.

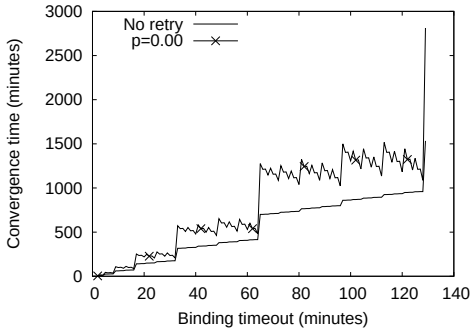
Figure 4.24: Impact of packet failure on hybrid search technique with retry.



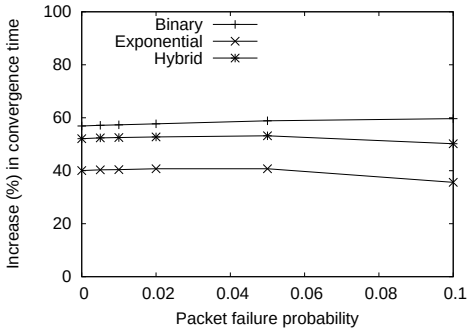
(a) Binary search.



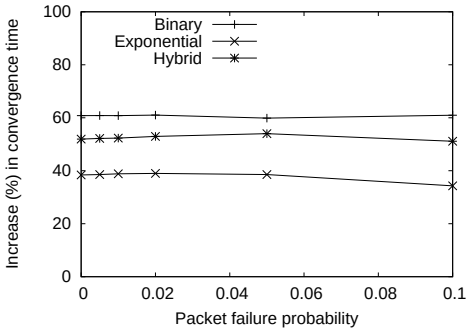
(b) Exponential search.



(c) Hybrid search.

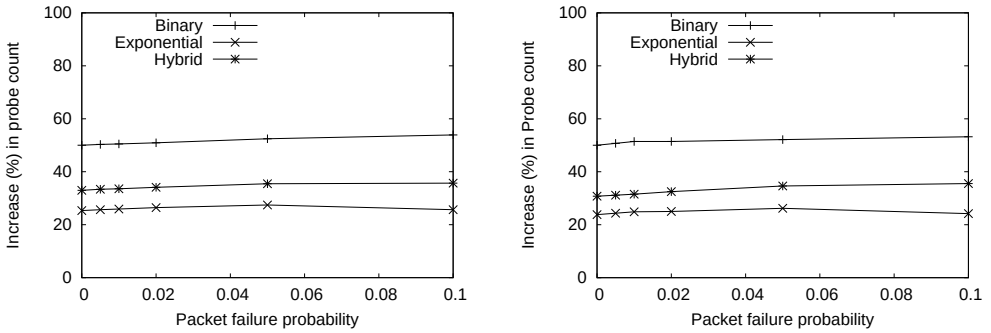


(d) Average increase (%) in convergence time due to retry.



(e) Median increase (%) in convergence time due to retry.

Figure 4.25: Impact of retry on convergence time of different search techniques.



(a) Average increase (%) in probe count due to retry.

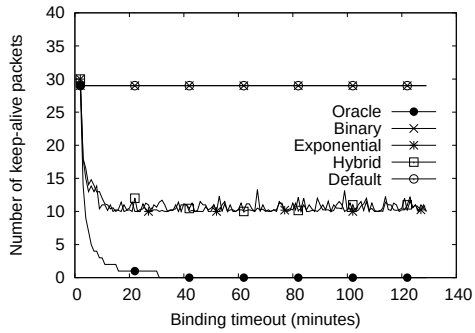
(b) Median increase (%) in probe count due to retry.

Figure 4.26: Impact of retry on probe count of different search techniques.

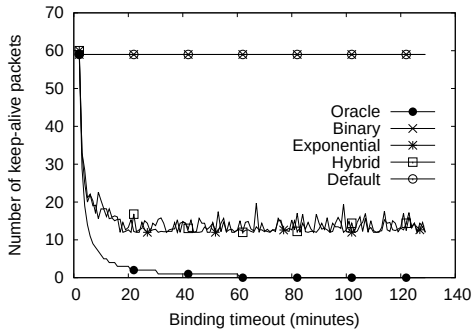
represents the convergence time with retry scheme implemented, but in this case too, there is no packet failure. Therefore, the increase in convergence time in the latter case is not caused by any transient failures. Rather it is caused by the fact that, when a keep-alive fails due to overshooting the optimal interval, we have to retry the test just to make sure that we have not encountered a transient failure. For each valid keep-alive failure, we encounter double the wait time as such. Hence the convergence time grows significantly. Binary search encounters the most increase in convergence time: around 60% on average, as can be seen in Figure 4.25(d). Exponential search has the least increase: around 40% on average.

Probe count in each technique is also increased due to the retry scheme. Figure 4.26 shows the average and median increase (%) in probe count. Like convergence time, probe count too is most increased in case of binary search; and least increased in case of exponential search. The curve for hybrid search is closer to the curve for exponential search.

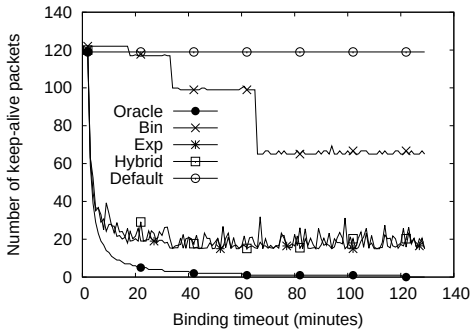
With single retry scheme, the error in detected binding timeout is in tolerable range. And there is significant impact on convergence time and probe count for each number of additional retries we introduce into our algorithms. As such, we would not implement more than single retry on packet failure. In subsequent experiment, we show that this indeed works satisfactorily.



(a) 30 minute run.



(b) 1 hour run.



(c) 2 hour run.

Figure 4.27: Number of keep-alive packets sent over 1 data and 1 test connection during meeting or seminar scenario. ($p = 0.02$, $q = 0.10$).

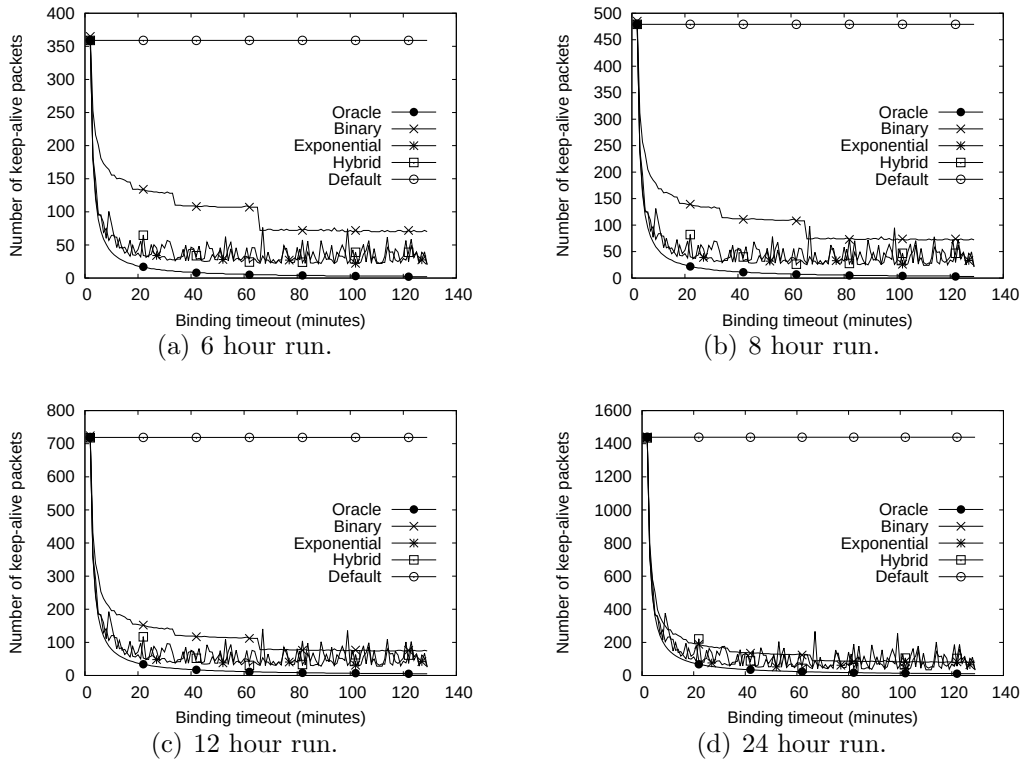
4.7 Performance of Modified Algorithm

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. That means the search terminated when the detected KA interval had error no more than 10% of the optimal keep-alive interval. The parameter r was not used in case of hybrid search – which means, binary search phase commenced after the first failure encountered during the exponential search phase.

The result for each binding timeout was averaged over 20 repetitions. The results are shown in Figure 4.27 and Figure 4.28. These curves are very similar to the corresponding curves of Figure 4.8 and Figure 4.10. The average percentage reduction in number of KA

packets sent is listed in Table 4.4. The values are comparable to the corresponding values in Table 4.2.

Therefore, we can conclude that the modified techniques successfully coped with transient network failures and were 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 hybrid search should be used to detect the optimal keep-alive interval.



(a) 6 hour run. (b) 8 hour run.

(c) 12 hour run. (d) 24 hour run.

Figure 4.28: Number of keep-alive packets sent over 1 data and 1 test connection over longer durations. ($p = 0.02$, $q = 0.10$).

Table 4.4: 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 (Hours)	**Binary search**	**Exponential search**	**Hybrid search**
0.5	0	62.37	62.37
1	0	75.44	75.58
2	26.45	82.39	82.62
6	71.59	88.19	88.56
8	77.57	89.03	89.52
12	83.59	90.10	90.56
24	89.61	91.32	91.69

Chapter 5

Conclusion

In this final chapter, we draw the conclusion of our thesis by describing the major contributions of this research work. We follow that up by providing some directions for future research.

5.1 Major Contributions

The major contributions of this thesis are as follows:

- We applied several search approaches to dynamically detect the optimal keep-alive interval. These include binary search, exponential search and hybrid search. Hybrid search combines different aspects of binary and exponential search techniques. 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 showed that both binary search and hybrid search performs $O(\lg h)$ probes over the test connection on average, when the optimal KA interval is in the range $[1, h]$. Exponential search, however, performs $O(\lg^2 h)$ probes in the same setting.

- We showed that total testing time to detect the optimal KA interval is $O(h \lg h)$ in both binary and hybrid search. $O(h \lg^2 h)$ time is taken in case of exponential search.

- The optimal keep-alive interval is always less than the binding timeout of the middlebox, due to propagation delay. We showed that this difference, however, is no more than a minute, since the propagation delay is much less than a minute and KA intervals that are tested are rounded to minute boundary.

- We defined a tunable parameter q in the search techniques. It represents the bound on error admitted in the detected KA interval, as a fraction of the optimal KA interval. By turning this parameter, the convergence time can be reduced.

- For hybrid search, we defined another tunable parameter r. It represents the maximum interval that may be tested during the exponential search phase. By turning this parameter, the convergence time can be reduced with very little or no impact on performance in terms of number of KA packets sent.

- Transient network failures may cause the search to settle on a keep-alive interval lower than the optimal one. To mitigate this issue, we proposed a retry scheme as follows: When a probe (sending of KA message) fails in the test connection, it is retried once. If both attempts fail, then we rule out the possibility of transient network issue. This scheme works very well; however, it increases the convergence time.

- As testing reveals better KA intervals, we updated the KA period in the data connection immediately. As such, reduction of number of KA packets sent is observed well before the testing is complete.

- We ran all the techniques over different durations to compare the reduction in KA count over data connection. The run durations are inspired by real world use cases and scenarios. For larger durations (24 hours), binary search performs very close

to the other two. For durations around and less than 12 hours, results of hybrid and exponential search are much better than that of binary search. For durations less than an hour, binary search is unable to reduce the number of KA packets sent at all. The smaller duration runs will be more common in real usage patterns, since a mobile device is expected to move from one APN to another, staying in each for short or moderate amounts of times. As such, exponential and hybrid search techniques should be preferred to binary search. And since exponential search has very high convergence time, we concluded that hybrid search should be used to detect the optimal keep-alive interval.

5.2 Directions of Further Research

The work in this thesis has introduced further research opportunities. These new research directions are listed in this final section.

1. In this research work, we examined the behavior of binary, exponential and hybrid searches thoroughly in detecting optimal keep-alive interval of TCP connections. We provided theoretical analysis of convergence time and probe counts of each of these techniques. But, we did not find any function to give the number of KA messages sent via the data connection over a specified time duration, while testing is in progress in parallel. We obtained this data only through simulation. Further work should be done to calculate this number theoretically.

2. Additionally, we could explore other search techniques. One option could be linear search, where we start with a small interval and upon each successful test, we increase the interval by a constant amount. According to our early analysis, however, this will be inferior to exponential search. Exponential search performs a slow start. Initially it is like linear search. But, as more tests are successful, it increases the KA intervals following geometric progression. However, some tweaks to linear search could be

researched to improve it. For example, we could introduce some randomness: after a test is successful the interval is increased by a constant amount or some multiple of the constant amount probabilistically.

3. Searching with multiple test connections is also an interesting approach. This can reduce the convergence time significantly and also improve the keep-alive intervals quickly. The more connections we can use for testing purpose, the more improvements we could potentially get. However, care should be taken to not overload the server with too many connections. In what sequence should each connection search? Should each connection employ the same technique, or is it better to mix different techniques in different connections? There seems to be a good research opportunity here.

4. Since it is not possible currently to differentiate between keep-alive failures from transient network failures, we re-send the keep-alive packet once when a failure is detected. This increases the convergence time. Could there be a way to isolate the transient errors, without any adverse effect on convergence time? This needs further investigation.

5. We have implemented the KA detection techniques on a simulation platform only. We have referenced the power model mentioned in Section 2.4 to infer that the reduction in keep-alive count has resulted in improvements in power consumption. We have not implemented the algorithms on real mobile devices. Also, we have not considered presence of traffic other than the keep-alives. In a real device, the radio could be in high power state due to transfer of some data from services like email, web browsing etc. If a keep-alive is scheduled at this time, the power overhead would be much less, compared to if it were scheduled when the device was otherwise idle. To get a more accurate measurement of improvements due to keep-alive testing, further experiments can be done in the simulation platform using usage traces from actual end user devices. Such traces can possibly be obtained from [4]. Obviously, the most accurate and reliable results can be obtained, once we implement the algorithms in real devices and perform actual power measurements in those devices in different user scenarios.

6. After the optimal keep-alive interval is detected, the data connection sends keep-alive using the optimal interval. 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. We need to develop a strategy to bring the data connection out of this unstable state (frequent disconnections). The most conservative approach would be to revert back to the default or the conservative keep-alive interval and restart testing for better intervals. Another option is to remain cautiously optimistic: reduce the keep-alive 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 cause little increase in power consumption but may fail to stabilize the connection. Several other strategies could be formulated. We could analyze these approaches in detail and run simulations to compare their behaviors.

7. In Section 1.5, we made an assumption that a when a device connects to a mobile or WiFi network, it can definitively tell whether it has connected to this network before. That way, it can use keep-alive interval cached from prior testing. It can also resume incomplete testing. If, on the other hand, the network is new to the device, it will trigger the keep-alive interval detection algorithm. However, a device can only use heuristics for this detection and may get it wrong sometimes. If a device thinks it has visited a network before, but in reality it had not, then it can potentially start using a keep-alive interval that overshoots the binding timeout of middle-box. In this case too, the data connection would become unstable. The techniques discussed above also helps in such situations to bring the data connection back to stable state and at the same time to trigger new testing.

8. Finally, work could be done to write a redistributable library that can be plugged into any device to detect the optimal keep-alive interval. The API and protocol design for such a library remains to be investigated.

Bibliography

[1] https://chromium.googlesource.com/chromium/chromium/+/trunk/google_apis/gcm/engine/heartbeat_manager.cc. [Online; Last accessed on 04-May-2014].

[2] https://productforums.google.com/forum/#!msg/nexus/fslYqYrULto/lU2D3Qe1mugJ. [Online; Last accessed on 04-May-2014].

[3] Apple Push Notification Service. https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html. [Online; Last accessed on 25-April-2014].

[4] Crawdad: A Community Resource for Archiving Wireless Data At Dartmouth. http://www.crawdad.org/. [Online; Last accessed on 26-April-2014].

[5] Google Cloud Messaging for Android. http://developer.android.com/google/gcm/index.html. [Online; Last accessed on 25-April-2014].

[6] Heartbeat Interval Adjustment. http://technet.microsoft.com/en-us/library/cc182270.aspx. [Online; Last accessed on 04-May-2014].

[7] Heartbleed. http://en.wikipedia.org/wiki/Heartbleed. [Online; Last accessed on 04-May-2014].

[8] Omnet++. http://www.omnetpp.org/. [Online; Last accessed on 25-April-2014].

[9] PowerTutor. A Power Monitor for Android-Based Mobile Platforms. http://www.powertutor.org. [Online; Last accessed on 11-May-2014].

[10] Proxy Server. `http://en.wikipedia.org/wiki/Proxy_server`. [Online; Last accessed on 25-April-2014].

[11] Push notifications for Windows Phone 8. `http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff402558(v=vs.105).aspx`. [Online; Last accessed on 25-April-2014].

[12] The Heartbleed Bug. `http://heartbleed.com/`. [Online; Last accessed on 04-May-2014].

[13] Understanding Direct Push. `http://technet.microsoft.com/en-us/library/aa997252(EXCHG.80).aspx`. [Online; Last accessed on 25-April-2014].

[14] Windows Push Notification Services (WNS) overview (Windows Runtime apps). `http://msdn.microsoft.com/en-us/library/windows/apps/hh913756.aspx`. [Online; Last accessed on 25-April-2014].

[15] 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*, pages 101–114. ACM, 2003.

[16] 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*, pages 280–293. ACM, 2009.

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

[18] A. Chakraborty, V. Navda, V. N. Padmanabhan, and R. Ramjee. Coordinating cellular background transfers using LoadSense. In *Proceedings of the 19th annual international conference on Mobile computing & networking*, pages 63–74. ACM, 2013.

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

[20] 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, September 2011. US Patent App. 13/229,325.

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

[22] 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*, pages 260–266. ACM, 2010.

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

[24] 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 App. 13/764,663.

[25] N. Hu and P. Steenkiste. Evaluation and characterization of available bandwidth probing techniques. *IEEE Journal on Selected Areas in Communications*, 21(6):879–894, 2003.

[26] M. Jain and C. Dovrolis. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput. In *ACM SIGCOMM Computer Communication Review*, volume 32, pages 295–308. ACM, 2002.

[27] M. Jain and C. Dovrolis. End-to-end estimation of the available bandwidth variation range. In *ACM SIGMETRICS Performance Evaluation Review*, volume 33, pages 265–276. ACM, 2005.

[28] V. Kononen and P. Paakkonen. Optimizing power consumption of always-on applications based on timer alignment. In *Proceedings of the Third International Conference on Communication Systems and Networks (COMSNETS)*, pages 1–8. IEEE, 2011.

[29] 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*, volume 1, pages 415–420. IEEE, 2000.

[30] L.-S. Meng, D.-s. Shiu, P.-C. Yeh, K.-C. Chen, and H.-Y. Lo. Low power consumption solutions for mobile instant messaging. *IEEE Transactions on Mobile Computing*, 11(6):896–904, 2012.

[31] J. Postel. Transmission Control Protocol. September 1981. RFC 793 (Standard).

[32] 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.

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

[34] A. Shye, B. Scholbrock, and G. Memik. Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 168–178. ACM, 2009.

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

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

[37] A. Ukhanova, E. Belyaev, L. Wang, and S. Forchhammer. Power consumption analysis of constant bit rate video transmission over 3G networks. *Computer Communications*, 35(14):1695–1706, 2012.

[38] N. Vallina-Rodriguez and J. Crowcroft. Energy management techniques in modern mobile handsets. *Communications Surveys & Tutorials, IEEE*, 15(1):179–198, 2013.

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

[40] Y. Xiao, R. Bhaumik, Z. Yang, M. Siekkinen, P. Savolainen, and A. Yla-Jaaski. A system-level model for runtime power estimation on mobile devices. In *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications (GreenCom) & Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, pages 27–34. IEEE, 2010.

[41] Y. Xiao, P. Savolainen, A. Karppanen, M. Siekkinen, and A. Ylä-Jääski. Practical power modeling of data transmission over 802.11g for wireless applications. In *Proceedings of the 1st International Conference on Energy-efficient Computing and Networking*, pages 75–84. ACM, 2010.

[42] S.-R. Yang. Dynamic power saving mechanism for 3G UMTS system. *Mobile Networks and Applications*, 12(1):5–14, 2007.

[43] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 105–114. ACM, 2010.

Index