UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
INSTITUTE OF COMPUTER SCIENCE

Kristjan Reinloo

Addressing Smartphones Located Behind Firewalls

Bachelor's Thesis (6 ECTS)

Supervisor:
Satish Narayana Srirama, PhD

Author: ..... "....." May 2013

Supervisor: ..... "....." May 2013

Professor: ..... "....." May 2013

Tartu 2013

Contents

Introduction **4**

1 State of the Art **5**
1.1 Smartphones 5
1.2 Related Work 6
1.3 Existing Solutions 6
1.3.1 Skype 6
1.3.2 Zeroconf and Bonjour 7
1.4 Used Technologies 7
1.4.1 Android 7
1.4.2 Node.js 8
1.4.3 MongoDB 9
1.4.4 Google Cloud Messaging for Android 9
1.4.5 Raw Sockets 10

2 Problem Statement **11**
2.1 Addressing Mechanisms 11
2.2 Network Address Translation 12
2.2.1 NAT Principles 12

3 Proposed Solution **14**
3.1 Architecture 14
3.2 Flow Description 15
3.2.1 Device Registration 15
3.2.2 Address Exchange 16
3.2.3 Connection Establishment 18
3.3 Test Results, Identified Limitations 19

Conclusions **21**

Tulemüüride taga paiknevate nutitelefonide adresseerimine **22**

References **24**

Appendices **26**
Appendix A 26

List of Figures

1.1 Overview of Android system architecture 8

3.1 Device and service registration 16
3.2 Exchanging addresses 17
3.3 Sending messages to smartphones through GCM 17
3.4 Connection establishment, NAT hole punching 19

Introduction

In recent years, smartphones have become considerably popular. They are inseparable companions for their owners while offering both entertainment and assistance in daily routines.

Such popularity has different reasons - with the development of technologies like touchscreens and microprocessors, smartphones are becoming more and more powerful in terms of computational power. This sets preconditions for the development of rather sophisticated software as well. Today it is already possible to play 3D games or record high-definition videos with mobiles. Moreover, smartphones and tablets are replacing laptops and desktops since they offer almost the same functionality and user experience while fitting into pockets and weighing only couple of hundred grams.

Most of the applications currently available for smartphones usually consume different kind of resources from the Internet, thus acting as clients. But given the fact that handheld devices have already roughly the same computational power as low-end laptops and netbooks, it is possible to offer services from the smartphones as well which could lead to new types of mobile applications and use cases.

Unfortunately, it is not trivial to access smartphones or any other host from the Internet due to widespread usage of some networking processes, for example, network address translation (NAT). To overcome such difficulties problems, an application suite is proposed in this thesis, which helps smartphones to become accessible service provides not only within local network but also across the Internet by mitigating some common addressing problems.

This thesis is divided into three chapters, followed by conclusions which include summary of created application suite and ideas for future work. Chapter 1 focuses on state of the art, it gives an overview of current smartphones' capabilities, related work and existing proprietary software. Chapter 2 describes existing problems and basic networking principles which prohibit smartphones from acting as servers. Chapter 3 introduces a solution which helps to mitigate problems found in previous chapter followed by test results and limitations of given software.

Chapter 1

State of the Art

This chapter includes short overview of latest developments regarding smartphones, related work in targeting addressing problems and description of available proprietary software which tackles similar above-mentioned problems. Finally, a list of used technologies which were used in the implementation part of this thesis is given.

1.1 Smartphones

In recent years, smartphones have considerably evolved regarding their performance. It is seen that the traditional calling has become a rarely used functionality, since nowadays smartphones are equipped with other communication interfaces, e.g. 3G or WiFi, besides older technologies like GSM. Aforementioned new technologies offer more bandwidth resulting in fast and data-heavy communication, like browsing the Internet or making video calls. Locally, smartphones have several sensors, for example gyroscope, accelerometer, GPS, pressure sensor, temperature sensor and even humidity sensor. These hardware gadgets enable developers to create both interesting and entertaining games and applications. Given the fact, that newer smartphones have already quad-core central processing units (CPUs), there is plenty of computing power for complex applications.

Speaking of software, two of the most popular operating systems found on handheld devices are Apple iOS and Android. Both are grown out of more general operating systems, OS X and GNU/Linux respectively. In fourth quarter of 2012, these operating systems were found in nearly 91% of total devices sold in that period of time [1].

1.2 Related Work

To solve the problem that hosts behind NAT routers can not be accessed from remote networks, this thesis makes use of NAT hole punching which creates temporary tunnels and allows peers to establish connection with each other.

Academics have been interested in NAT technology and addressing for quite some time. Hole-punching method called SYN injection, which is used in this thesis has been described by scientists from University of Duisburg, Germany [2]. In given paper, 9 different routers were used to evaluate the current method. This resulted in $9 \times 8 = 72$ test cases. It was found that the overall success rate of TCP connection establishment was 78%. It is worth mentioning that their testing took place in laboratory, therefore such factors like firewalls which could possibly reduce the success rate, were out of scope.

Detailed description of NAT and its characteristics are described in "Peer-to-Peer Communication Across Network Address Translators" [3], which also looks into UDP connection establishment, in addition to TCP. Several tests were also carried out in that paper, including complex network setups, where peers are behind several NAT layers. Tests included hardware from 9 different vendors and also 3 OS-based NAT solutions. Overall results show that 82% UDP and 64% TCP connections were successful, resulting in true peer-to-peer (P2P) connections.

1.3 Existing Solutions

1.3.1 Skype

Probably one of the best known voice over IP (VoIP) applications which uses combination of client-server and P2P models is Skype. Unfortunately, little is known how Skype internally works since it is a proprietary software. Still, some papers have been published regarding its architecture and protocols which are based mainly on observations.

Baset [4] claims that Skype uses a similiar architecture as KaZaa, where there are different kind of nodes: ordinary nodes and supernodes. While many ordinary nodes are behind firewalls and NAT routers, they still manage to connect seamlessly to other such hosts when establishing call sessions. In [4], it is stated that variations of STUN [5] and TURN [6] protocols are used in order to determine NAT and firewall properties. Guha et al. [7] also support this claim, saying that Skype tries different approaches to establish a connection between two peers and that the final fallback method is to simply relay the session through a publicly accessible supernode.

1.3.2 Zeroconf and Bonjour

Zeroconf is a protocol for interconnected devices to automatically set up a network which includes IP interface configuration, translation between host name and IP address, IP multicast address allocation and service discovery - all without any user intervention [8].

Bonjour is an implementation of given protocol by Apple and among other things it is used for discovering available printers and iTunes libraries on a new network [9]. With the help of dynamic domain name system (DNS) provider, it is possible to statically configure local networks where hosts and their services are accessible from outside the network as well [10].

1.4 Used Technologies

This section describes briefly technologies and available software, which were needed for implementing the proposed solution of this thesis. Chapter 3 includes more details on how any of the given technologies were used.

1.4.1 Android

Android is an operating system which is based on GNU/Linux. It was originally created by Android Inc, a company founded in 2003 for developing a new operating system for mobile devices. In 2005, Google acquired Android Inc and its operating system [11]. Since then, Android has become the most popular operating system for smartphones [1].

Because it is based on GNU/Linux, all kernel changes are published under GNU General Public License version 2 (GPLv2) and rest of the operating system mostly under Apache License version 2.0. The entire operating system can easily be downloaded from Android Git repositories [12]. Buying an Android-based smartphone means also using non-free software since every vendor packs some proprietary software to the device, for example it is common that hardware drivers are only developed in-house and not released for public access.

By default, Android applications run in an isolated environment. More specifically, every application is assigned with a unique user ID under which it is ran [13]. System files are placed under a separate partition which is mount as read-only partition to improve security and avoid any modification of critical files.

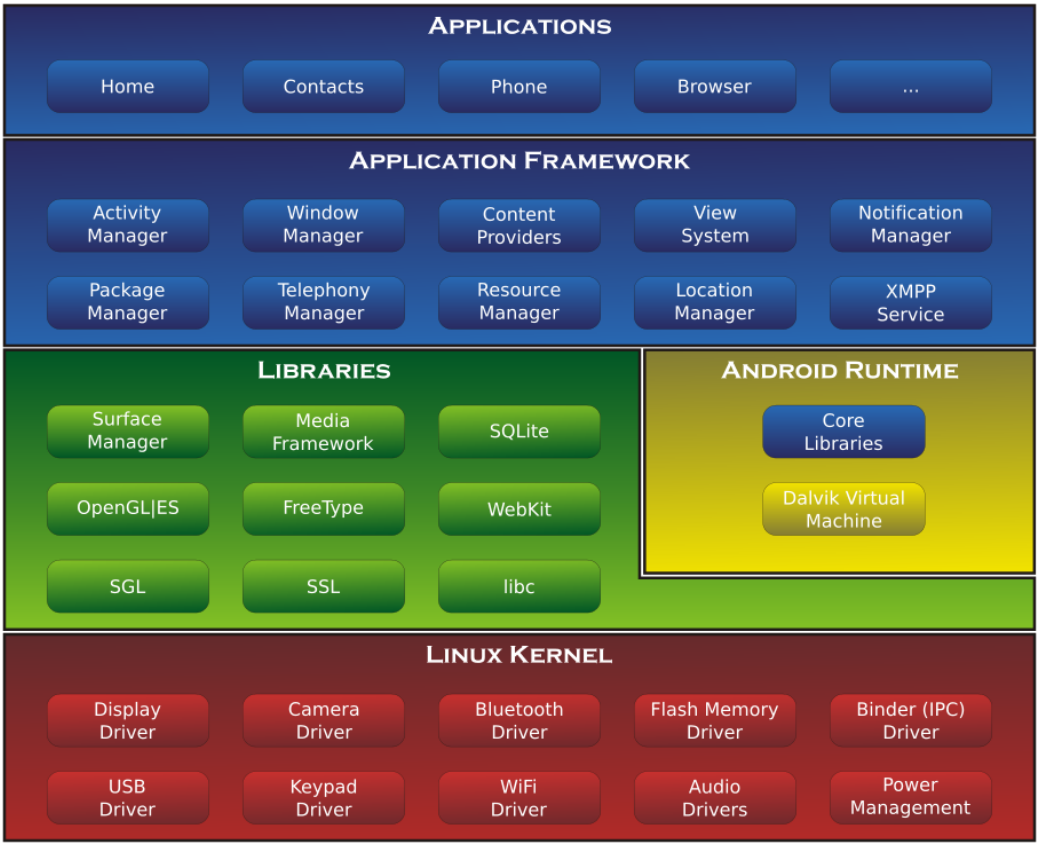

Figure 1.1: Overview of Android system architecture[14]

What is more, users usually do not even have root permissions on the phone, only very few system components run under root privileges. This has both positive and negative effects. Most probably an average user does not even care if he has such privileges or not, because when applications run without any problems this goes unnoticed and user can enjoy safe execution of his applications. On the other hand, advanced users may want to debug certain applications or run some code in root privileges and have absolute control over the operating system. In that case the device has to be "rooted" where a kernel bug or any other security flaw has to be exploited in order to gain such rights. Usually these actions require some technical background and skills, thus too complicated for regular users.

1.4.2 Node.js

Node.js is a server-side JavaScript environment based on V8 JavaScript engine from Google. Its architecture is based on event-driven non-blocking

I/O model which allows development of real-time scalable applications. It was originally created by Ryan Dahl, after he dropped out from his PhD programme in 2009 [15]. Currently Joyent Inc is responsible for the maintenance and development of the project.

Since Node.js uses JavaScript as its programming language, it is fairly easy to get started and learn the basics of it. However, the event-driven model is something that can take time to truly master it.

1.4.3 MongoDB

MongoDB [16] is an open-source document-oriented NoSQL database. As with Node.js, it is built with scalability in mind. MongoDB itself is written in C++, but there are several client libraries for different languages, including JavaScript.

Everything in MongoDB is stored as documents, which are JSON-like objects. Unlike relational databases, objects can have variable amount of data fields and it is important to note that there are no database tables. Instead, MongoDB uses something called collections, which are slightly similar to tables, in sense that they group together objects with similar properties.

Another interesting aspect is the fact that MongoDB does not use traditional database schemas. This means that one does not have to define strict format of data which is stored in the database. However, in most cases it makes sense to store data with similar properties in common structures.

1.4.4 Google Cloud Messaging for Android

"Google Cloud Messaging for Android (GCM) is a service that allows you to send data from your server to your users' Android-powered device." [17].

GCM is a successor to the Android Cloud to Device Messaging (C2DM) framework. Its main purpose is to offer the functionality of sending notification messages to Android devices at any given time. What is more interesting, applications do not even have to be running when the messages are sent to the smartphone - the operating system will wake up the corresponding application if everything is correctly configured. This way applications can still receive updates from external sources while preserving battery as much as possible since they do not have to constantly run in the background.

For developers it is rather easy to migrate given framework to their existing applications. One just has to add the Android library to one's application and implement callbacks which are called when certain events happen (e.g. message has been received, device has been unregistered, etc).

1.4.5 Raw Sockets

Raw sockets do not necessarily refer to a specific technology, it is just a way of manually sending and receiving Internet Protocol (IP) packets for example without following specific protocols. Most of the modern operating systems support such actions, though there are some restrictions - for example on GNU/Linux, root privileges are required if one wants to use raw sockets.

In other words, using raw sockets, developers can cook packets by themselves which are then sent out via network interface controller (NIC) by the operating system. It is possible to manually specify for example media access control (MAC) addresses, IP addresses, port numbers and even add payload to the packet, if needed. This all relates to Open Systems Interconnection (OSI) model layers 2-7.

Raw sockets can be used with both good and bad intentions. For example malicious users can write software which could be used for different network attacks. One such attach type is known as SYN-flooding, where TCP packets with SYN flags set are sent to a server which sends back packet with SYN-ACK flags and waits for final ACK packet from the client in order to finish TCP three-way handshake[1]. Since server allocates small amount of memory for this half-open connection, it is possible to remotely force the server to consume all of its available memory and in worst case scenario, knock the server offline.

[1]TCP connection establishment using the three-way handshake - `https://tools.ietf.org/html/rfc793#section-3.4`

Chapter 2

Problem Statement

This chapter talks about networking principles and addressing problems which currently exist on the Internet due its various architectural design decisions and limitations. A short description of one technique which is used to mitigate some of the defined problems is given in more detail.

2.1 Addressing Mechanisms

Currently most of the Internet uses Internet Protocol Version 4 (IPv4) for exchanging packets on the network layer with respect to OSI model. Since exactly 32 bits are allocated for both source and destination address in the IP packet header, there can be total of 2^{32} distinct addresses. Unfortunately, due to the number of end-points on the Internet, almost all of these addresses are already in use.

The main problem with smartphones (or any other device on the local network in that matter) wanting to offer services not just to local network hosts is the fact that they can not be addressed and accessed from the Internet. This is because due to the shortage of IPv4 addresses, most of the edge routers perform network address translation in order to reduce the amount of required IPv4 addresses.

To overcome this problem, some applications use NAT hole punching, where they trick routers and firewalls to allow temporary tunnels in order to pass incoming traffic from external networks to the local area network without initiating an outgoing connection first.

2.2 Network Address Translation

Network address translation is a technique which includes modifying IP header fields by routers. There are several rationales behind such activity, most notably, it tries to mitigate the problem of IP address space exhaustion in order to make it possible to add new devices on the Internet without requiring more public IP addresses.

Another reason why NAT is so widely deployed is related to different security aspects. By hiding entire networks behind a single IP address prevents network scanning by malicious users. In addition, with the help of NAT it is easy to enforce firewall rules which allow or deny certain connections to remote networks.

Moreover, by default NAT routers act as firewalls and drop all incoming packets which are trying to establish a connection (e.g. TCP packet with SYN flag set), since they do not know what to do with such packets other than to discard them. This helps to avoid any unwanted connections to local area network, thus increasing significantly security of servers which are meant to be accessed from local network only.

2.2.1 NAT Principles

NAT principles where the source address and port number is changed (SNAT) can be described by following example: let there be a local area network which consists of ten computers *C{1-10}*, all of them have a unique IP address in range of 192.168.1.{1–10} and that the routing device R on given LAN has IP address 192.168.1.254. Moreover, R has another IP address, given by its Internet Service Provider (ISP), which is 90.190.30.10. This address shall be called a public IP address.

Now let's say that one of the computers (*C1*) with IP address 192.168.1.1 wants to connect to public web server S on the Internet with IP address 74.125.232.110. In this case *C1* constructs a TCP packet with source port 41000[1] and destination port 80 and wraps it into an IP packet where the source IP address is marked as 192.168.1.1 and destination as 74.125.232.110. *C1* decides with the help of its routing table that it has to send this packet to R which acts as default gateway for this LAN.

When this packet reaches R, it will not be forwarded immediately. Instead, R changes the source IP address to 90.190.30.10 which is the public IP address of R. Besides that, R looks into the TCP header fields and changes the source port to 32123[1]. R has now created a mapping 192.168.1.1:41000

[1] randomly chosen from available port number pool

$\leftrightarrow$ 90.190.30.10:32123 and it forwards the packet according to its routing table. When S responds to *C1*'s request, it sends packet back to R with the destination IP address marked as 90.190.30.10 and destination port 32123 because S knows nothing about the *C1*'s real IP address.

When the packet reaches R, it looks up its mapping table and sees that IP address 90.190.30.10 and port 32123 were mapped to *C1*'s IP 192.168.1.1 and port 41000. R changes the destination IP to *C1*'s IP and destination port to 41000 and forward the packet to *C1*. *C1* has now successfully changed packets with S, while knowing nothing about NAT which was performed by R.

By doing such packet modification, computers $C\{$*1-10*$\}$ require only one public IP address (90.190.30.10) to communicate with other peers on the Internet, since R updates its mapping table every time a connection is initiated by any of the LAN hosts and it remembers where to forward incoming packets with the destination IP address 90.190.30.10. Moreover, many different local networks can use same address space for their network. Three such address spaces which are not routed on the Internet, are especially allocated for this [18].

Chapter 3

Proposed Solution

This chapter gives an overview of the application suite which was created in order to overcome addressing problems in networks where NAT is deployed. At first, a high-level design of the applications is presented, followed by description how connection between peers is established. Finally, initial test results and list of limitations are presented.

3.1 Architecture

In this thesis an application suite was designed and implemented which tries to apply NAT hole punching on smartphones, making them accessible from anywhere on the Internet. It consists of several components: a rendezvous server, an Android application and a library for NAT hole punching. The application suite allows running arbitrary server software on the smartphone without any special preconfiguration and it takes care of forwarding incoming connections to the real server software on the phone.

The rendezvous server runs on publicly accessible server. Main responsibilities of it include handling registration / unregistration of smartphones and their services, GCM messaging and offering assistance during P2P connection establishment. Current implementation is written in Node.js while including NoSQL database (MongoDB) as its storage solution.

The Android application runs on the smartphone, while handling GCM messages which are sent by rendezvous server, establishing P2P connection and offering interface for registering / unregistering the smartphone and services which run on it. It is written in Java, using the official Android SDK.

Library for NAT hole punching and creating the P2P connection is written in C to be portable for both personal computers (only GNU/Linux is currently supported) and Android-based smartphones. Furthermore, raw

sockets' API is not usually available on higher level languages, even though some third-party libraries exist. Java-based Android application can call and run native code through Java Native Interface (JNI).

3.2 Flow Description

3.2.1 Device Registration

In order to access smartphone from the Internet, it first has to register itself with GCM server and rendezvous server (Figure 3.1). At first the smartphone uses the GCM library provided by Google to ask for a registration ID which can later be used to send GCM messages back to the phone. Smartphone has to provide a sender ID which identifies the application (rendezvous server in this case) because otherwise it is not possible to distinguish GCM messages meant for different application on the phone.

After receiving the registration ID, the smartphone uses it as a parameter of a REST[1] call to the rendezvous server. The rendezvous server stores the GCM registration ID in its database and responds with another ID which is used to identify different smartphones on the server. The smartphone stores this ID in its application settings and uses it in subsequent REST calls.

Now that the device has registered itself, it can also register one or more services which are running on the phone. Service registration process is similar to device registration where a HTTP POST request is made to corresponding end-point, but as an additional info, the ID which was received after registration is now added to the request. Services themselves are identified currently only by name.

Unregistration of services and the device is similar to registration processes, but instead of POST, the DELETE is chosen as the HTTP request method.

[1] representational state transfer - `https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm`

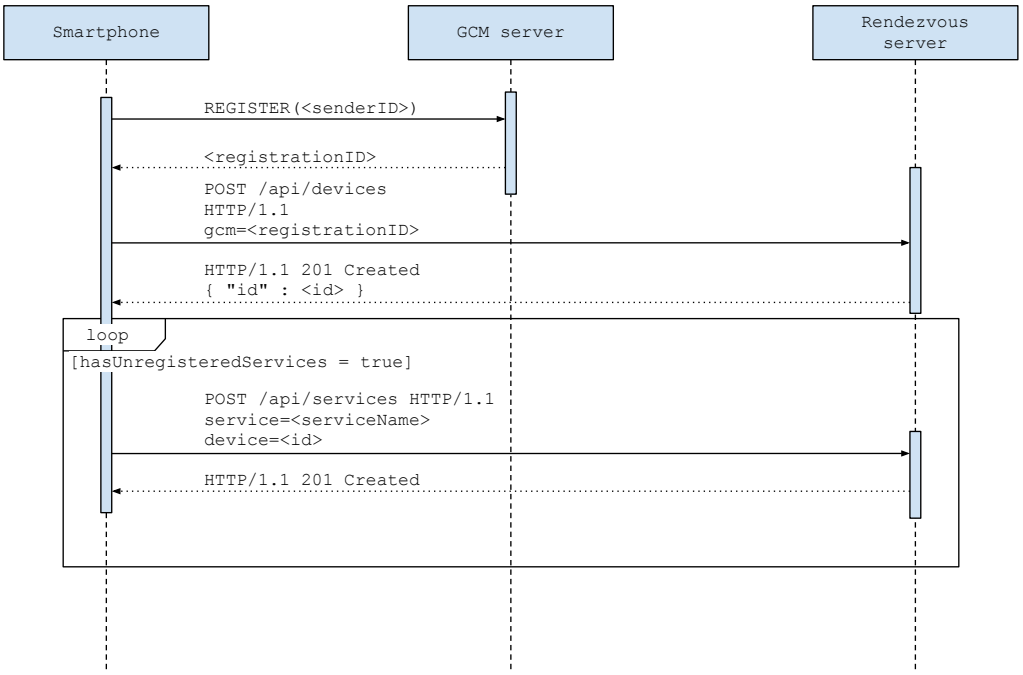

Figure 3.1: Device and service registration

3.2.2 Address Exchange

After registration, it is possible to exchange current public addresses between end-points who wish to establish connections. To do so, initiating side has to open a TCP connection with the rendezvous server and tell which service it wants to access as shown in Figure 3.2.

When the rendezvous server receives a request, it queries from its database for service which was specified in the packet. When no such service can be found, the server simply closes the connection. However, on a successful response from the MongoDB, the server looks up the GCM registration ID of the device with another database query, whose service was returned during the first search.

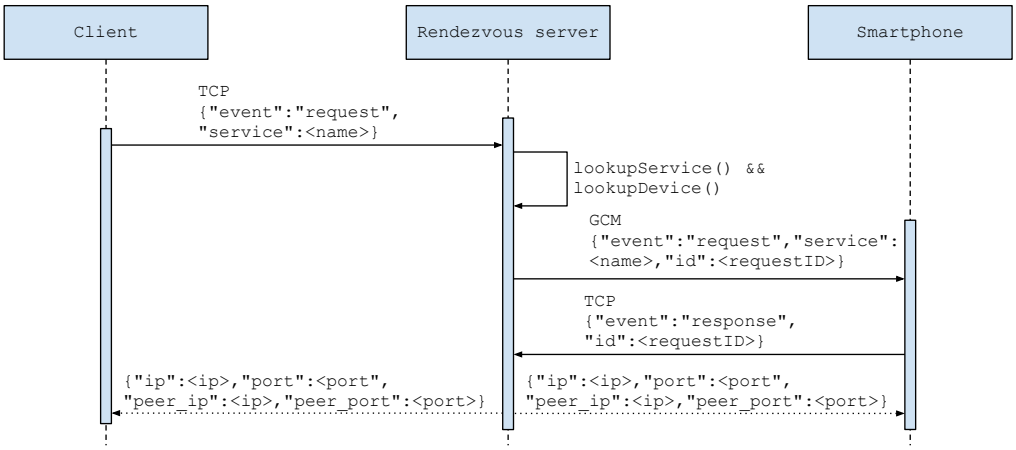

Figure 3.2: Exchanging addresses

The server can now send a GCM message to the corresponding smartphone. This requires making a single HTTP POST request. HTTP headers of an abbreviated example request are shown in Figure 3.3, where the `key` is obtained from the Google API console while registering a new application, `registration_id` is the ID which smartphone used to register itself with the rendezvous server and `data.message` is plain text message which is sent directly to the device.

When the smartphone receives the GCM message, it also opens a TCP connection with the rendezvous server and notifies it by sending a JSON-encoded message to indicate that the smartphone is ready to accept an incoming connection. Now both peer are connected to an agent who knows their public addresses. As the final step here, the rendezvous server sends out one packet to each peer which consists of IP addresses and port numbers of both clients.

```
POST /gcm/send HTTP/1.1
Host: android.googleapis.com
Content-Type: application/x-www-form-urlencoded;charset=UTF-8
Authorization: key=<PRIVATE_API_KEY>

registration_id=<DEVICE_ID>&data.message=<MESSAGE>
```

Figure 3.3: Sending messages to smartphones through GCM

3.2.3 Connection Establishment

The final phase in establishing a connection between two peers includes creating TCP sockets and creating NAT mappings. By this time, both peers know each other's IP addresses and port numbers. NAT hole punching method used in current implementation is a modified version of the SYN injection found in [2].

At first, the initiating side creates a raw socket and a regular TCP socket, deliberately setting the time to live (TTL) to a low value of the latter by using the `setsockopt()` API call. By doing so, the client ensures that when the SYN packet is sent, a NAT mapping is created at the client's router but since the packet never reaches the smartphone's router, it can not responding with a RST packet, which would cause the connection to fail[1]. When the SYN packet is sent, the client captures the initial sequence number and the timestamp of given packet by using the raw socket and sends this data to the rendezvous server using the TCP socket which was used for exchanging addresses. The rendezvous server forwards this captured TCP packet data to the smartphone using also the same communication channel which was established previously.

Similarly to the client, the smartphone also forces its router to create a NAT mapping by sending out a packet with a low TTL value and the client's IP marked as the destination.

[1] not all routers respond with a RST packet

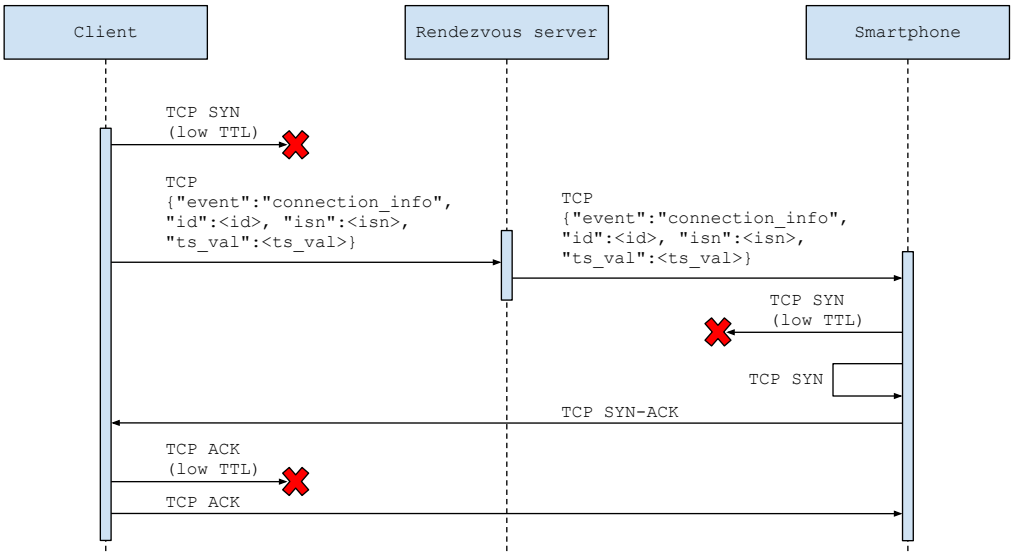

Figure 3.4: Connection establishment, NAT hole punching

Smartphone then crafts a packet where the source address is the client's IP address and the destination is the smartphone's IP address. Also, the sequence number and the timestamp are set using the information received from the rendezvous server. It continues to inject this packet into its own networking stack. Since a server application listens on the port which was specified as the destination port in the injected packet, the kernel sends out a SYN-ACK packet to the client as the second step in the three-way handshake.

The SYN-ACK packet sent from the smartphone reaches the client since a NAT mapping was previously created, so the router in front of the client knows where to forward it. As a response, the client sends out an ACK packet indicating a successful establishment of the connection. Unfortunately, this packet is also sent out with a low TTL value, so the client must capture this packet, increase the TTL value and resend it, in order to finish the connection establishment. As the final step, the client restores the TTL value of its regular socket so that all following packets sent to the smartphone reach their destination.

3.3 Test Results, Identified Limitations

Given solution has been tested in two network setups: a) smartphone is behind a NAT router, connecting client is publicly accessible, i.e. all ports are

open and no network adressing translation is performed; b) both smartphone and connection client are behind NAT routers. It is worth mentioning that in all cases, peers were behind one NAT layer.

For a test scenario, a standalone HTTP server application for Android was written using an open-source HTTP library [19]. The HTTP server was started and registered with the rendezvous server.

In both network setups, the connection establishment was successful and the client could access the HTTP server with a web browser. However, current implementation has some drawbacks and restrictions with regard to network topology and smartphone.

The first prerequisite comes from the fact that in order to use raw sockets, some code needs to be executed with root privileges. This means that the smartphone has to be rooted and the `su` binary has to be installed on it.

Another limitation assumes presence of some specific network attributes. To begin with, the routers in front of connecting peers need to add mappings to their NAT tables in an incremental manner. If a router assigns port numbers randomly from the available port pool, then the connection establishment will fail since there is no way to predict the NAT mapping which router has created.

This sets ground for another restriction, which is that when two peers are trying to establish a connection, any new connection initialized between address exchange (see Section 3.2.2 for details) and hole punching by any of the two peers will cause the connection establishment to fail as well. The reason is that currently, the peer port number is incremented by one when it is received from the rendezvous server. So most likely, on a busy network, the current implementation is not very stable.

Security aspects were not addressed in this thesis, but it is worth mentioning that currently the messages between peers and the rendezvous server are exchanged in plain text format in order to speed up the prototyping process.

Conclusions

In this thesis a solution was proposed how to mitigate addressing problems of hosts which are located behind firewalls. Given solution was implemented by creating an application suite consisting of three separate modules: a rendezvous server, an Android application and a library for NAT hole punching.

It was tested in two scenarios where the smartphone whose services were accessed, was located behind a router which performed network address translation. In both of those cases, the connection establishment was successful. However, some limitations of current implementation were identified.

Identified limitations included executing code with root privileges in the smartphone, incremental NAT mapping by the routers and relatively idle local network of both connecting peers.

As the current implementation is only an initial prototype, it is not yet mature enough to be used in real applications. Nonetheless, given solution could be improved in different ways, e.g. adding more NAT hole punching techniques which have different prerequisites making it more stable in other network environments. Also the functionality to discover the network and NAT properties could be added.

Tulemüüride taga paiknevate nutitelefonide adresseerimine

Bakalaurusetöö (6EAP)
Autor: Kristjan Reinloo
Juhendaja: Dr. Satish Narayana Srirama

Resümee

Nutitelefonid on tänapäeval kogumas järjest enam populaarsust, kuna nende tehniliste parameetrite arengu tõttu on nad hakanud asendama süle- ja lauaarvuteid.

Traditsioonilised nutitelefonide rakendused enamasti ainult tarbivad internetis leiduvat informatsiooni. Kuna aga tarkade seadmete arvutusvõimsus on juba võrreldav kodukasutaja sülearvutitega, saab neid kasutada ka hoopis serveritena. See loob omakorda aga eeltingimused uut tüüpi rakenduste arenguks.

Paraku on tõsine probleem nii nutitelefonide kui ka teiste seadmete adresseerimisega üle interneti, mis asuvad tavalistes kohtvõrkudes nagu näiteks kodudes. Antud probleem on enamasti tingitud tõsiasjast, et kõikide võrgus leiduvate seadmete identifitseerimiseks ei jätku unikaalseid aadresse. Seetõttu on kasutusele võetud võrguaadresside tõlkimine (*network address translation*), mis aga omakorda raskendab suvaliste seadmete adresseerimist.

Antud töös on välja pakutud üks võimalikest lahendustest, kuidas luua ühendus nutitelefonidega, mis asuvad võrguaadresse tõlkivate ruuterite taga. Selleks on kasutatud nutitelefonidele teatiste saatmise võimalust ning manuaalselt internetiprotokolli (IP) pakettide koostamist, et tekitada ajutised tunnelid ruuterites.

Tarkvara, mis selle töö raames kirjutati, testiti olukorras, kus mõnda teenust pakkuv nutitelefon oli võrguaadresse tõlkiva ruuteri taga. Ühenduse loomine oli edukas, kuid antud lahendus nõuab mõningate eeltingumuste täitmist. Näiteks peab nutitelefonis olema võimalik jooksutada koodi ad-

ministraatori õigustes, ruuterid peavad võrguaadresside tõlkimisel kasutama kasvavaid pordinumbreid ning võrk ei tohi olla liialt koormatud.

Kuna valminud lahendus on alles esmane prototüüp, saab seda tulevikus edasi arendada, et see oleks kasutatav ka reaalsetes rakendustes. Töös on välja toodud soovitused lisada ruuterite ja tulemüüride omaduste avastamise funktsionaalsus ning muude tunnelite tekitamise meetodeid, mis töötaksid ka teistsuguste omadustega võrkude ja seadmete korral.

References

[1] Gartner Inc. Gartner Says Worldwide Mobile Phone Sales Declined 1.7 Percent in 2012. https://www.gartner.com/newsroom/id/2335616. Accessed: 21.04.2013.

[2] S. Holzapfel, M. Wander, A. Wacker, and T. Weis. SYNI - TCP Hole Punching Based on SYN Injection. In *Network Computing and Applications (NCA), 2011 10th IEEE International Symposium on*, pages 241–246, 2011.

[3] G. Marchetto, M.P. Manzillo, L. Torrero, L. Ciminiera, and F. Risso. Robustness analysis of an unstructured overlay for media communication. *Communications, IET*, 5(4):409–417, 2011.

[4] Salman A Baset and Henning Schulzrinne. An analysis of the skype peer-to-peer internet telephony protocol. In *IEEE infocom*, volume 6, pages 23–29, 2006.

[5] Jonathan Rosenberg, Joel Weinberger, Christian Huitema, and Rohan Mahy. Stun-simple traversal of user datagram protocol (udp) through network address translators (nats). Technical report, RFC 3489, IETF, Mar, 2003.

[6] J Rosenberg, R Mahy, and C Huitema. Turn: traversal using relay nat. Technical report, Internet draft, Internet Engineering Task Force, 2004.

[7] Saikat Guha and Neil Daswani. An experimental study of the skype peer-to-peer voip system. Technical report, Cornell University, 2005.

[8] A. Williams. Requirements for Automatic Configuration of IP Hosts. http://files.zeroconf.org/draft-ietf-zeroconf-reqts-12.txt. Accessed: 02.05.2013.

[9] Apple Inc. Apple - Support - Bonjour. https://www.apple.com/support/bonjour/. Accessed: 02.05.2013.

[10] Dyn. Bonjour and DNS Service Discovery - Dyn. http://dyn.com/support/bonjour-and-dns-discovery/. Accessed: 04.05.2013.

[11] Bloomberg L.P. Google Buys Android for Its Mobile Arsenal - Businessweek. http://www.businessweek.com/stories/2005-08-16/google-buys-android-for-its-mobile-arsenal. Accessed: 30.04.2013.

[12] Google Inc. Android Git Repositories - Git at Google. https://android.googlesource.com/. Accessed: 30.04.2013.

[13] Google Inc. Android Security Overview — Android Open Source. http://source.android.com/tech/security/. Accessed: 30.04.2013.

[14] File:Android-System-Architecture.svg - Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Android-System-Architecture.svg. Accessed: 11.05.2013.

[15] Ryan Dahl - History of Node.js - YouTube. https://www.youtube.com/watch?v=SAc0vQCC6UQ. Accessed: 11.05.2013.

[16] 10gen Inc. MongoDB. http://www.mongodb.org/. Accessed: 21.04.2013.

[17] Google Inc. Google Cloud Messaging for Android — Android Developers. https://developer.android.com/google/gcm/index.html. Accessed: 27.04.2013.

[18] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address Allocation for Private Internets. https://tools.ietf.org/html/rfc1918. Accessed: 04.05.2013.

[19] Paul Hawke. Nanohttpd/nanohttpd - GitHub. https://github.com/NanoHttpd/nanohttpd. Accessed: 11.05.2013.

Appendices

Appendix A

The source code of the implementation is available on a following web page

`https://github.com/kreinloo/bscthesis`

With the given hash of the last commit

`1fcfda7dc4965ce73978425dcf44cf014197afe7`

Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, Kristjan Reinloo (date of birth: 24.04.1991), herewith grant the University of Tartu a free permit (non-exclusive licence) to:

- reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and
- make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

Addressing Smartphones Located Behind Firewalls

. . . . . ,

supervised by Dr. Satish Narayana Srirama.

- I am aware of the fact that the author retains these rights.
- I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 13.05.2013