

Slovak University of Technology Bratislava
Faculty of Informatics and Information Technologies

FIIT-13428-5796

Kamil Burda

**PORT CONTROL PROTOCOL IN SOFTWARE DEFINED
NETWORKS**

Diploma Thesis

Degree Course: Computer and Communication Systems and Networks

Field of study: 9.2.4 Computer Engineering

Institute: Institute of Computer Systems and Networks, FIIT STU Bratislava

Supervisor: Ing. Martin Nagy

2015, May

Annotation

Degree Course: Computer and Communication Systems and Networks

Author: Kamil Burda

Diploma Thesis: Port Control Protocol in Software Defined Networks

Supervisor: Ing. Martin Nagy

2015, May

User applications, such as instant messaging or VoIP, may have problems traversing the network through middleboxes (NAT gateways, firewalls). Several mitigation techniques exist, including a relatively new protocol called Port Control Protocol (PCP). PCP allows user applications to receive IP address and port mapping directly from the middleboxes. Additionally, PCP allows user applications to optimize the number of keepalive messages sent to the network in order to maintain the connection, reducing the network load and prolonging battery life in mobile devices. Software defined networking (SDN) is a new paradigm in computer networks that allows the network behavior to be programmed. SDN networks increase flexibility and vendor compatibility by providing a standard communication interface for the network elements. The goal of the diploma thesis is to implement PCP over an SDN network (using the OpenFlow protocol) and to measure the reduction of keepalive traffic with PCP enabled in the implemented network, focusing on mobile networks, where the impact of the reduction of the signaling traffic may be considerable.

Anotácia

Študijný program: Počítačové a komunikačné systémy a siete

Autor: Kamil Burda

Diplomová práca: Protokol PCP v softvérovo definovaných sieťach

Vedúci diplomovej práce: Ing. Martin Nagy

máj 2015

Používateľské aplikácie, ako napr. rýchle správy (instant messaging) alebo VoIP, môžu mať problémy s komunikáciou v sieti cez sieťové zariadenia ako napr. brány NAT alebo bezpečnostné brány. Na zmiernenie problémov sa môžu použiť existujúce techniky, resp. protokoly, vrátane relatívne nového protokolu Port Control Protocol (PCP). Protokol PCP umožňuje používateľským aplikáciám získať informácie o mapovaní IP adries a portov priamo z uvedených sieťových zariadení. PCP navyše umožňuje optimalizovať vysielané množstvo tzv. správ keepalive (správy na udržiavanie spojenia), čím sa znižuje záťaž siete a predlžuje sa životnosť batérie na mobilných zariadeniach. Softvérovo definované siete (SDN), ako nový prístup budovania a riadenia sietí, umožňujú naprogramovať správanie sa siete. Siete SDN zabezpečujú vyššiu flexibilitu a kompatibilitu medzi zariadeniami od rôznych výrobcov vďaka štandardnému komunikačnému rozhraniu medzi týmito zariadeniami. Cieľom diplomovej práce je implementovať protokol PCP v sieťach SDN (s použitím protokolu OpenFlow) a určiť redukciu množstva správ keepalive s protokolom PCP v implementovanej sieti, pričom dôraz je kladený na mobilné siete, kde má redukcia signálizačnej sieťovej premávky značný vplyv.

Acknowledgments

I would like to thank my supervisor, Martin Nagy, for consultation regarding the diploma thesis. I would also like to thank my family for providing moral support.

Contents

Introduction.....	1
1. Middleboxes.....	2
1.1. NAT Gateway.....	2
2. User Applications.....	4
2.1. Keepalives.....	4
2.1.1. TCP Keepalives.....	4
2.1.2. Keepalives and Middleboxes.....	5
2.2. Middlebox Traversal Methods.....	5
2.2.1. STUN, TURN, ICE.....	5
2.2.2. Middlebox Signaling Protocols.....	5
2.2.3. UPnP IGD.....	6
2.2.4. Application Layer Gateway.....	6
3. Port Control Protocol.....	7
3.1. PCP Messages.....	8
3.1.1. PCP Request.....	8
3.1.2. PCP Response.....	9
3.1.3. MAP Opcode.....	10
3.1.4. PEER Opcode.....	11
3.1.5. PCP Server Recovery with ANNOUNCE Opcode.....	12
3.1.6. Options.....	12
3.2. PCP Request Processing by PCP Server.....	13
3.2.1. Learning, Modifying and Maintaining Mapping Lifetime.....	16
3.3. PCP Server Discovery.....	17
3.4. PCP Client Implementation.....	17
3.5. Available PCP Software.....	18
3.5.1. PCP Testing Tool.....	18
3.5.2. PCP Client Library.....	18
3.6. Comparison of PCP and Middlebox Traversal Methods.....	19
4. Mobile Networks.....	20
4.1. 3G Networks.....	20
4.2. Radio Resource Control.....	21
4.2.1. RRC States.....	22
4.2.2. RRC Inactivity Timers.....	23
4.2.3. RRC State Transitions with Keepalives.....	24
4.3. Conclusions.....	24
5. Software Defined Networking.....	26
5.1. OpenFlow.....	27
5.1.1. OpenFlow Switch Overview.....	27
5.1.2. Communication with Controller.....	28

5.1.3. Flow Entries.....	28
5.1.4. Instructions.....	29
5.1.5. Ports.....	29
5.2. SDN Software.....	30
5.2.1. Forwarders.....	30
5.2.2. OpenFlow Controller Software.....	30
6. Analysis Summary.....	32
7. Specification.....	34
7.1. Goals.....	34
7.2. Requirements.....	34
7.2.1. Port Control Protocol.....	35
8. Design.....	36
8.1. Architecture.....	36
8.1.1. Components.....	37
8.1.2. Network Application Components.....	38
8.2. PCP Client Mapping Request – Processing.....	39
8.3. Edge Forwarder.....	39
8.4. NAT Forwarder.....	40
9. Implementation.....	42
9.1. Implementation Environment.....	42
9.2. Implementation Description.....	43
9.2.1. ARP Message Processing.....	44
9.2.2. NAT Table.....	45
9.2.3. Managing Mapping Lifetime.....	46
9.3. Verification.....	46
10. Evaluation.....	47
10.1. Battery Life Extension.....	48
10.1.1. Battery Power Consumption Figures.....	48
10.1.2. Formulas.....	48
10.1.3. Results.....	50
10.2. Signaling Traffic Reduction.....	53
10.2.1. Network Traffic in WCDMA Networks.....	53
10.2.2. Formulas.....	53
10.2.3. Results.....	53
10.3. Conclusions.....	55
10.3.1. Determining PCP Mapping Lifetime.....	56
Summary.....	57
Conclusion.....	58
References.....	59

Appendices

A. Attached DVD Contents.....	63
B. Installation.....	64
C. Using the Software.....	66
D. Plan of Work.....	71
Resumé.....	

Abbreviations

ALG	Application layer gateway, Application-level gateway
API	Application programming interface
ARP	Address Resolution Protocol
CGN	Carrier-grade NAT
DHCP	Dynamic Host Configuration Protocol
EDGE	Enhanced Data rates for GSM Evolution
EIR	Equipment Identity Register
GGSN	Gateway GPRS Support Node
GMSC	Gateway Mobile Switching Center
GPRS	General packet radio service
GSM	Global System for Mobile Communications
HLR	Home Location Register
HSPA	High Speed Packet Access
IANA	Internet Assigned Numbers Authority
ICE	Interactive Connectivity Establishment
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPsec	Internet Protocol Security
LTE	Long-Term Evolution
MAC	Media Access Control
MSC	Mobile Switching Center
NAPT	Network address and port translation
NAT	Network address translation
NAT-PMP	NAT Port Mapping Protocol
OS	Operating system
PCP	Port Control Protocol
PDN	Packet data network
PSTN	Public Switched Telephone Network
RFC	Request for Comments
RNC	Radio Network Controller
RRC	Radio Resource Control
SDN	Software defined networking

SGSN	Serving GPRS Support Node
STUN	Session Traversal Utilities for NAT
TURN	Traversal Using Relays around NAT
UMTS	Universal Mobile Telecommunications System
UPnP	Universal Plug and Play
UPnP IGD	Universal Plug and Play Internet Gateway Device
VLR	Visitor Location Register
VoIP	Voice over IP Protocol
WCDMA	Wideband Code Division Multiple Access
XML	Extensible Markup Language

Figures

Figure 1.1: Example of address translation by a NAT gateway.....	2
Figure 3.1: Usage of PCP in networks.....	8
Figure 3.2: PCP request message format.....	9
Figure 3.3: PCP response message format.....	10
Figure 3.4: Message format for MAP opcode for PCP request.....	11
Figure 3.5: Message format for PEER opcode for PCP request.....	12
Figure 3.6: Generic header for PCP options.....	13
Figure 3.7: Basic processing of a PCP request by the PCP server.....	14
Figure 3.8: Building a PCP response from the corresponding PCP request by the PCP server.....	15
Figure 3.9: Processing of PCP MAP and PEER requests.....	16
Figure 4.1: Architecture of 3G networks.....	21
Figure 4.2: Radio Resource Control (RRC) states.....	23
Figure 4.3: Possible RRC state transitions when sending a single keepalive.....	24
Figure 5.1: SDN architecture overview.....	26
Figure 5.2: Overview of an OpenFlow switch (forwarder).....	27
Figure 5.3: Flow entry structure in an OpenFlow switch.....	28
Figure 6.1: PCP Deployment in Traditional Networks.....	32
Figure 8.1: Architecture of the network.....	36
Figure 8.2: NAT table entry fields.....	38
Figure 8.3: Network application components.....	38
Figure 8.4: PCP request processing by the network application.....	39
Figure 8.5: Edge forwarder flow entries.....	40
Figure 8.6: NAT forwarder flow tables and entries.....	41
Figure 9.1: Test topology for verification.....	42
Figure 9.2: Forwarder implementation overview.....	43
Figure 9.3: Example of a MAC overwriting flow entry installation with proxy ARP approach.....	45
Figure 10.1: Amount of battery power saved based on keepalive intervals relative to reference values and cost of 0.15 mAh per keepalive.....	51
Figure 10.2: Amount of battery power saved based on keepalive intervals relative to reference values and cost of 0.6 mAh per keepalive.....	51
Figure 10.3: Amount of battery lifetime saved based on keepalive intervals relative to reference values and cost of 0.15 mAh per keepalive.....	52
Figure 10.4: Amount of battery lifetime saved based on keepalive intervals relative to reference values and cost of 0.6 mAh per keepalive.....	52
Figure 10.5: Number of signaling messages reduced based on keepalive intervals and reference values (40 observed messages).....	54
Figure 10.6: Number of signaling messages reduced based on keepalive intervals and reference values (50 observed messages).....	54
Figure 10.7: Number of signaling messages reduced based on keepalive intervals and reference	

values (20 unobserved messages).....55

Figure 1: PCP Request processed by the PCP server in the controller.....67

Figure 2: Packet trace from host 1.....68

Figure 3: Packet trace from host 2.....68

Tables

Table 4.1: Average current (power consumption) of a mobile device in RRC states [30] [5] [29]....	23
Table 10.1: Measured battery power consumption of keepalives in 3G WCDMA networks [5].....	48
Table 10.2: Amount of battery power consumption saved of a mobile device connected a WCDMA network given battery capacity and reference values.....	50
Table 10.3: Reduction of the number of signaling messages given the number of messages per keepalive and reference values.....	55
Table 1: Plan of work for the diploma thesis.....	71
Table 2: Plan of work – assessment.....	71

Introduction

User applications running on hosts, such as instant messaging (IM) or VoIP, may have problems traversing the network through the so called *middleboxes* placed in computer networks, especially NAT gateways. In order to mitigate the incompatibility of user applications with NAT, several approaches exist, such as Session Traversal Utilities for NAT (STUN).

With the middleboxes in the network, the user applications have to keep the connections alive to avoid the middleboxes closing the connections prematurely. This is accomplished by sending *keepalive* messages from the user applications to the destination host in regular intervals. Given the fact that the user applications do not know the keepalive timers set on the middleboxes, they tend to send the keepalives in very short intervals, increasing the network load.

In mobile networks, each message sent over the network causes a substantial number of signaling messages to be generated and sent over the network. With a large number of mobile devices connected to the network and running one or more applications, this introduces increased network load in the network and delay in communication. From the perspective of a mobile device, sending an excessive amount of keepalives drains its battery life faster.

Port Control Protocol (PCP) is a relatively new protocol that allows user applications to receive mapping information directly from the middleboxes, including external IP address and port in case of NAT gateways, and the mapping timer. The applications can consequently optimize their keepalive timers using this information.

Software defined networking (SDN) is a new paradigm in computer networks that allows the network behavior to be programmed in a simpler manner. SDN networks increase flexibility and improve vendor compatibility by providing a standard communication interface between the network elements, such as *OpenFlow*.

The goal of the diploma thesis is to implement PCP over SDN networks and to measure the keepalive traffic reduction with PCP enabled in the networks, focusing on mobile networks, where the impact may be considerable.

Chapters 1 and 2 of the document describe middleboxes and their role in the computer networks, user applications and their traversal issues over the middleboxes and related methods that provide solutions to the traversal issues. Chapter 3 focuses on one of the traversal methods – the most essential part of this thesis – the PCP protocol. Chapter 4 gives a brief overview of mobile networks and further focuses on 3G networks using the Wideband Code Division Multiple Access (WCDMA) access technology. Chapter 5, the last part of the analysis, discusses the concept of SDN. Chapter 6 summarizes the analysis and gives an overview of the current state of the networks and issues to be resolved.

Chapter 7 specifies the goals of the diploma thesis and the requirements for the solution. Chapter 8 describes the design of the solution. Chapter 9 describes the most important aspects of the implementation of the solution and describes how to verify the solution. Chapter 10 evaluates the reduction of battery power consumption of mobile devices and the reduction of signaling traffic in WCDMA networks with PCP deployed in the network.

1. Middleboxes

Middlebox is a term that refers to “any intermediary device performing functions other than the normal, standard functions of an IP router on the datagram path between a source host and destination host” [1].

Source [1] identifies several types of middleboxes, mainly:

- NAT gateways,
- firewalls,
- application-level gateways (ALGs).

Generally speaking, IP routing and the corresponding routers are transparent to end hosts. This transparency is broken by introducing middleboxes in the network, as the middleboxes alter the packet forwarding. Thus, end hosts must now cope with middleboxes as well when establishing communication [1].

While ALGs can be classified as middleboxes, they enable end hosts to properly traverse other middleboxes – NAT gateways and firewalls. Hence, ALGs as one method to traverse other middleboxes are described in section 2.2.4.

1.1. NAT Gateway

A NAT gateway is a network device that performs network address translation [2]. A carrier-grade NAT (CGN) is a NAT gateway located in service-provider networks [4]. Figure 1.1 illustrates an example of a packet being subject to translation by a NAT gateway in the network. NAT allows hosts from private networks (such as 192.168.0.0/16) to communicate with hosts on the Internet – on the public network with publicly routable IP addresses.



Figure 1.1: Example of address translation by a NAT gateway

More importantly, NAT can conserve IPv4 address space by translating multiple internal (private) IP addresses to one external (public) IP address with different transport protocol ports. This type of NAT is called Network Address and Port Translation (NAPT). When overwriting packet fields, NAT gateways must recompute the checksum of each relevant header (IP, transport protocol, Ethernet frame).

In this document, unless otherwise specified, “NAT” refers to the general network address translation process to simplify discussion. It does not imply NAPT or address translation exclusive to IP.

The translation between an internal IP address, protocol and port and an external IP address, protocol and port can be referred to as a *mapping*. NAT gateway stores mappings for each connection. Mappings can be created statically (manually configured by the network administrator) or dynamically (created once a packet traverses through the NAT gateway). In case of a dynamic mapping, the NAT gateway allocates an external IP and an external port from the pool of defined external IP addresses and ports.

An idle timer may be associated for each mapping. If no packet traverses through the NAT gateway for this connection, the mapping is removed by the gateway. Employing a timer for a mapping may have several reasons.

NAT gateways do not participate in the end-to-end connection between the communicating hosts. Hence, NAT gateways generally cannot determine when the connection is terminated. For TCP connections, NAT gateways may track segments with FIN or RST flags, in which case the gateways may remove the mapping immediately upon receiving such segments. Other transport protocols, such as UDP, do not indicate when the connection terminates. Additionally, the NAT gateways cannot detect one or both communicating hosts suddenly terminating the connection (e.g. by crashing) [2]. Another reason to add timers to mappings is to conserve memory on NAT gateways or keep the NAT pool for dynamic mappings from being depleted too quickly.

According to several vendor devices specified in source [5], the default timer values for TCP range from 30 to 150 minutes and for UDP from 60 to 300 seconds.

2. User Applications

Certain user applications need to communicate with hosts behind NAT or similar middleboxes. Such applications have problems traversing the middleboxes because of unexpected IP address and port rewrite on the route to the destination. These applications include VoIP, social networks, instant messaging or online gaming, and are sometimes referred to as *always-on applications*.

The data transmitted in these applications are intermittent. That is, no data may be sent for a certain period of time. This poses another problem for the applications – middleboxes shut down idle connections and the applications would have to establish the network connections again.

To avoid the network connections being shut down by middleboxes, user applications send *keepalive* messages [6] to the destination. The keepalives usually contain little to no payload to conserve network load.

2.1. Keepalives

Keepalives are messages sent by end-user applications to check for broken connections or to prevent disconnection due to inactivity [7]. Without keepalives, the connection can be broken if there are middleboxes on the path between the end hosts, such as NAT gateways. Middleboxes maintain mapping information for each connection and assign a timeout for the mapping. If no messages are sent within the connection for the time specified by the timeout, the middlebox removes the mapping and the connection is broken.

The format of keepalives and their usage depends on the communication protocol used. Keepalives tend to be short in length to preserve the network bandwidth [7].

2.1.1. TCP Keepalives

Once established, a TCP connection lasts until it is closed explicitly by either host. There is no connection timeout associated with the established TCP connection if no messages are sent for a long time [8]. This implies that if, for example, the remote host crashes, the local host has no way of learning that the remote host no longer maintains the connection. Periodically sending TCP keepalives can detect such broken connections [6].

A host desiring to maintain a TCP connection sends an empty ACK segment to the destination host. The destination host replies with another ACK segment. The size of the segment payload can be zero (i.e. no data need to be included in the segments) [7].

The implementation of TCP keepalives is optional. If sending TCP keepalives is implemented, it must be turned off by default [6]. Applications may enable or disable keepalives and may adjust the keepalive time. By default, the keepalive time must be set to at least two hours [6].

Among the more popular platforms, Windows¹, Linux² and OS X³, support TCP keepalives and also support setting the keepalive time. Popular platforms for mobile devices, such as Windows Phone, Android, iOS, also support TCP keepalives and keepalive time, given that these platforms

¹ <https://msdn.microsoft.com/en-us/library/dd877220%28VS.85%29.aspx>

² <http://tldp.org/HOWTO/TCP-Keepalive-HOWTO/usingkeepalive.html>

³ <http://serverfault.com/questions/216956/how-to-check-tcp-timeout-in-linux-macos/275506#275506>

are based on Windows, Linux and OS X, respectively.

2.1.2. Keepalives and Middleboxes

For communication passing through NAT gateways, IETF requires setting the mapping lifetime (timeout) for UDP to at least 120 seconds (with 300 seconds recommended) [9] and for TCP to at least 124 minutes [10]. For IPSec ESP connections, the keepalive interval is locally configurable, with the default value of 20 seconds [11].

2.2. Middlebox Traversal Methods

2.2.1. STUN, TURN, ICE

Session Traversal Utilities for NAT (STUN) [12] is a protocol that helps application protocols cope with NAT traversal. STUN uses a client-server model for message exchange. STUN can also be used as a keepalive mechanism.

STUN can be used by hosts – STUN clients – to determine their external IP address and port allocated by a NAT gateway from a STUN server. The STUN server usually resides on the public Internet.

The operation of STUN is simple. The STUN client sends a request to the STUN server on the public Internet. From the request, STUN server sees the source IP address and port as the external IP address and port of the STUN client, since the NAT gateway translated the source IP address and port of the request. The STUN server encapsulates the external IP address and port to the payload of a response message that is then sent back to the STUN client. The host thus receives its external mapping information.

The advantage of STUN is that it does not require modifications of NAT gateways. The disadvantage of STUN is that it does not work properly with symmetric NAT⁴.

Traversal Using Relays around NAT (TURN) [13] defines an intermediate node – a relay (or a TURN server). End hosts use the relay to forward data traffic through (such as voice). In case of TURN, it is possible for one host to communicate with multiple other hosts with the same external IP address and port. While TURN can support symmetric NAT, the relays are subject to heavy network load.

Interactive Connectivity Establishment (ICE) [14] is a technique that combines STUN and TURN and chooses the most effective way of communication between hosts behind a NAT⁵.

2.2.2. Middlebox Signaling Protocols

This subsection gives a brief overview of protocols that, unlike the previously mentioned protocols, communicate with middleboxes directly.

NAT Port Mapping Protocol (NAT-PMP) [3] is the predecessor to the Port Control Protocol (described in chapter 3). NAT-PMP allows the host to receive its external IP address and external port. NAT-PMP works only on NAT gateways located one hop away from the host.

⁴ More information at: <http://think-like-a-computer.com/2011/09/19/symmetric-nat/>

⁵ More information at: http://www.pjsip.org/pjnath/docs/html/group__PJNATH__ICE.htm

Two more middlebox signaling protocols include Middlebox Communication Architecture and Framework (MIDCOM) [15] and NEC's Simple Middlebox Configuration Protocol (SIMCO) [16]. These protocols appear to be outdated as they were not widely deployed in networks despite being in existence many years.

2.2.3. UPnP IGD

The Internet Gateway Device (IGD) is a device on the edge of a LAN and a WAN network, allowing end users to connect to the Internet. IGD as a UPnP-based protocol allows users to control and configure multimedia devices connected to the network, including the configuration of DHCP, DNS and also the network address translation on the IGD device [17].

2.2.4. Application Layer Gateway

An application layer gateway⁶ is a software component that manages specific protocols notorious for having problems with NAT traversal, such as Session Initiation Protocol (SIP) or File Transfer Protocol (FTP). ALG can examine the payload of packets and determine whether NAT needs to be performed. The use of ALGs has been discouraged [18].

⁶ More information at: <https://www.juniper.net/techpubs/software/junos-es/junos-es93/junos-es-swconfig-security/application-layer-gateways-algs.html>

3. Port Control Protocol

Port Control Protocol (PCP) [18] [19] [20] allows network hosts to communicate directly with middleboxes (NAT gateways or firewalls). With PCP, the host can receive or explicitly request a mapping from an internal IP address, protocol and port to an external address, protocol and port. This way, the host can traverse NAT gateways properly and communicate with other hosts behind NAT.

PCP does not provide a mechanism to inform the remote host about the host's external mapping. This responsibility is left to the user application and is usually handled by a rendezvous (proxy) server, accessible in the public network by both communicating hosts.

The mapping assigned by the middlebox to the host also contains *mapping lifetime* – the timer associated with the mapping. PCP also retrieves the lifetime value from the mapping. Given the mapping lifetime, the application running on the host can optimize the interval of sending keepalives over the network. Reduced keepalive traffic can extend the battery life of mobile devices and reduce network traffic overhead [18] [21].

PCP originated as an alternative to application layer gateways (ALGs) [18], existing protocols for NAT traversal such as STUN⁷, and existing protocols facilitating communication with middleboxes, such as UPnP IGD [3]. PCP as a relatively new protocol was standardized by IETF in April 2013 as RFC 6887 [18] and is the successor to *NAT Port Mapping Protocol* (NAT-PMP) [3].

PCP can be deployed in several scenarios [18]:

- home networks with NAT gateways (e.g. integrated in routers),
- carrier-grade NAT,
- simple firewalls.

PCP supports both IPv4 and IPv6 address mapping and transport protocols with 16-bit port numbers. PCP also supports protocols that do not use port numbers (such as IPsec ESP or ICMP) for firewalls, but not NAT gateways [18].

As defined in RFC 6887, PCP can be operational only in single-homed networks. If a network is single-homed, only one route exists to the Internet. The recently released RFC 7488 provides support for multi-homed networks [22].

Figure 3.1 shows the typical deployment and usage of PCP in networks. The host runs a user application that attempts to connect to the application server. The application retrieves mapping information from the middlebox in order to establish the connection or optimize the interval of sending keepalive messages.

The application invokes the *PCP client* to request mapping information from the middlebox. The middlebox runs the *PCP server*, which processes the request of the PCP client and sends back mapping information.

⁷ <https://datatracker.ietf.org/doc/charter-ietf-pcp/>

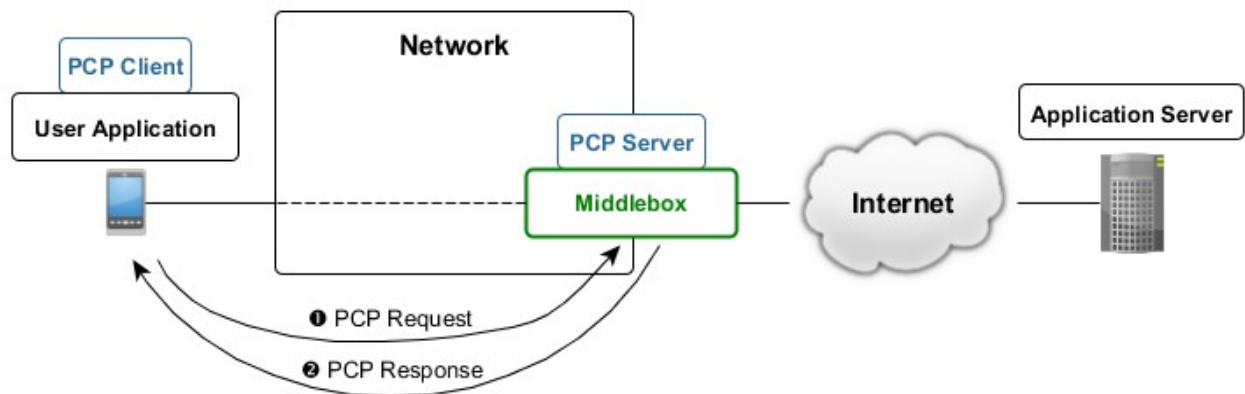


Figure 3.1: Usage of PCP in networks

3.1. PCP Messages

PCP defines two message types: *PCP request* and *PCP response*.

PCP messages are sent over UDP and are not acknowledged. PCP request uses destination UDP port 5351 and PCP response uses source UDP port 5351.

The PCP request is used by the host to request mapping information from the PCP server. The PCP response is used by the PCP server to inform the PCP client of the state of the mapping, usually informing the PCP client that the mapping information has been assigned to the PCP client.

PCP can be considered a request/response protocol. This point of view may not be accurate – unlike other request/response protocols, a PCP request does not necessarily have to be followed by a PCP response. If a PCP request sent by the PCP client was lost on the path to the PCP server, the PCP client may retransmit the same message. The PCP client can also use the same PCP request to renew the mapping information. The PCP server usually generates PCP responses to PCP requests sent by PCP clients. The PCP server may also send a PCP response to inform the PCP client about the new state of the mapping, e.g. because of middlebox reconfiguration or failure. Given the message exchange model, RFC 6887 refers to PCP as a *hint/notification* protocol [18].

PCP messages contain fields with IP addresses, such as the external IP address of the host assigned by the PCP server. IP addresses in PCP messages are always formatted as IPv6 addresses. IPv4 addresses are represented as IPv4-mapped IPv6 addresses:

```
::ffff:<IPv4 address>
```

3.1.1. PCP Request

Figure 3.2 shows the common header format for PCP requests.

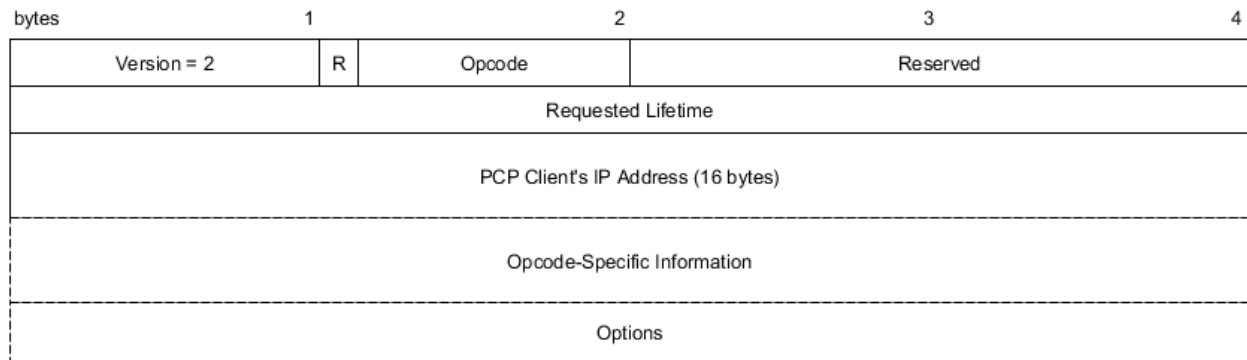


Figure 3.2: PCP request message format

The PCP request contains the following fields:

- *Version* – for PCP, this value is always set to 2. This field can be used to determine the supported version of PCP on the PCP server, should newer versions of PCP are defined.
- *R* – 1-bit field – 0 indicates PCP request.
- *Opcode* – 7-bit operation code for the PCP request. In RFC 6887, the following opcodes are defined: MAP (1), PEER (2) and ANNOUNCE (0). Opcodes are discussed in sections 3.1.3, 3.1.4 and 3.1.5, respectively.
- *Reserved* – zero-padded bits, ignored by PCP server.
- *Requested Lifetime* – mapping lifetime requested by PCP client. If the field is set to zero, the PCP server deletes the mapping.
- *PCP Client's IP Address* – IP address of the PCP client in IPv6 format. This field is used by the PCP server to determine additional middleboxes along the path from the PCP client that do not run the PCP server.
- *Opcode-Specific Information* – additional fields defined by the corresponding opcode.
- *Options* – a set of optional fields in the type-length-value format. Options can be ignored by the PCP server. A brief overview of options is given in section 3.1.6.

3.1.2. PCP Response

The format of a PCP response message is shown in Figure 3.3.

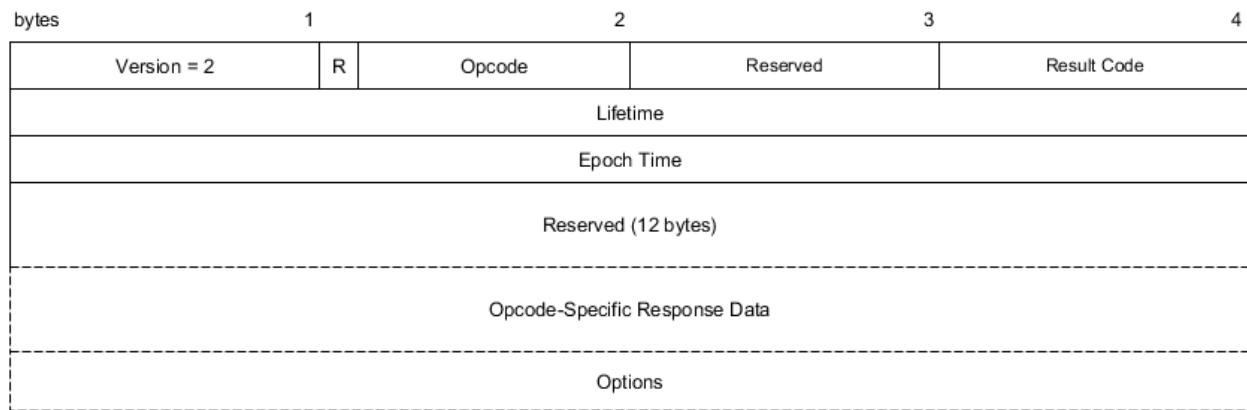


Figure 3.3: PCP response message format

The PCP response contains the following fields:

- *Version* – set to 2 by the PCP server.
- *R* – 1-bit field set to 1, indicating PCP response.
- *Opcode* – copied from the PCP request.
- *Reserved* – zero-padded bits.
- *Result Code* – value indicating a successfully processed PCP request (0, SUCCESS) or processing failure (values 1-14, depending on the type of failure). The meaning of each result code is described in RFC 6887 [18].
- *Lifetime* – mapping lifetime assigned by the PCP server to the PCP client.
- *Epoch Time* – time in seconds since the PCP server started operation. This is used by the PCP client to determine whether the PCP server lost state (e.g. if the PCP server crashed and rebooted). If so, the PCP client recreates its mapping information as per RFC 6887 [18].
- *Reserved (12 bytes)* – if the PCP request was parsed successfully, *Reserved* contains zero bits. Otherwise, the field contains the last 12 bytes (96 bits) of the *PCP Client's IP Address* field in the PCP request.

Opcode-specific information and options for PCP responses are covered in subsequent sections.

3.1.3. MAP Opcode

A PCP client uses a PCP MAP request in case a user application desires to host a server (for online gaming, a web server, etc.) and listen for incoming traffic from the public network. After the PCP client received mapping information, it is the responsibility of the application to announce its external (public) IP address, protocol and port to a rendezvous server, as mentioned in the introduction of this chapter, as PCP does not provide this function [18].

The format of the MAP opcode for a PCP request is shown in Figure 3.4.

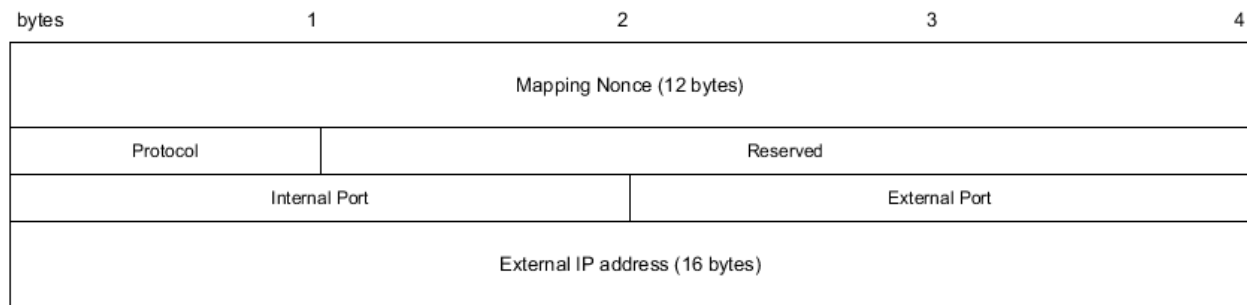


Figure 3.4: Message format for MAP opcode for PCP request

The MAP opcode for PCP requests contains the following fields:

- *Mapping Nonce* – random value generated by the PCP client. The PCP client uses this value to validate PCP responses [18].
- *Protocol* – protocol above the IP header. The protocol numbers are identical to those defined by IANA⁸.
- *Reserved* – zero-padded bits.
- *Internal Port* – internal port that the user application wishes to use to establish a connection.
- *External Port* – external port preferred by the user application. If the application does not require a specific external port, 0 is assigned.
- *External IP address* – external IP address in IPv6 format preferred by the user application. If the application does not require a specific external IP address, all-zeros IPv6 address is assigned (i.e. “:”).

The PCP response containing the MAP opcode copies all fields from the corresponding MAP request except *External Port* and *External IP address*. The PCP server assigns the external IP address and port according to the mapping entry created by the underlying middlebox.

3.1.4. PEER Opcode

A PCP client uses a PCP PEER request if a user application wishes to establish an outbound connection to an application server (i.e. the application acts as a client) in the public network or in another local network behind NAT [18].

A PCP PEER request can also be used by the PCP client to query existing mapping, e.g. in case the middlebox created an implicit mapping without the application communicating with the PCP server first.

The format of the PCP PEER request is shown in Figure 3.5. The message format is content-wise almost identical to PCP MAP messages. PCP PEER messages define two additional fields:

- *Remote Peer Port* – port of the remote host the user application wishes to establish connection with.

⁸ <http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml>

- *Remote Peer IP address* – IP address in IPv6 format of the remote host the user application wishes to establish connection with.

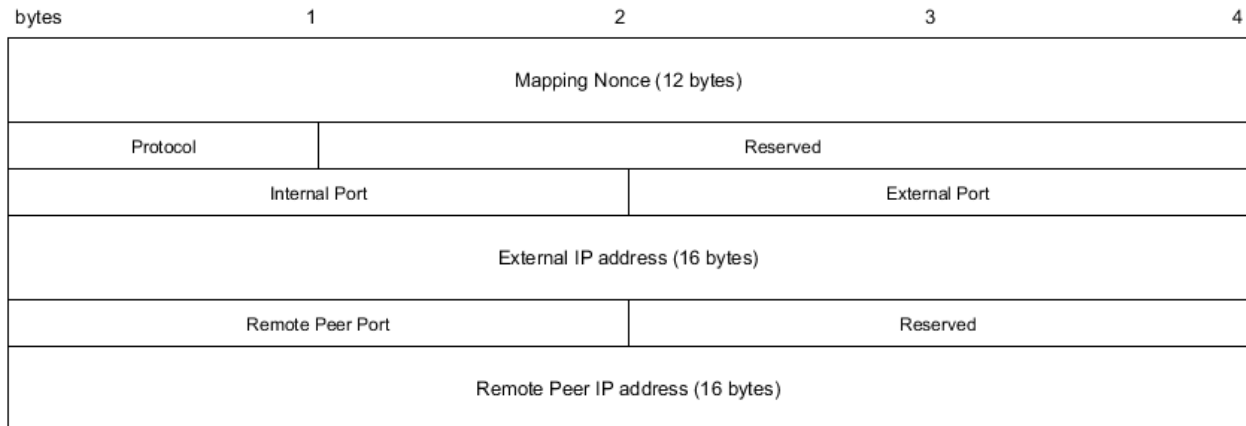


Figure 3.5: Message format for PEER opcode for PCP request

3.1.5. PCP Server Recovery with ANNOUNCE Opcode

PCP offers a mechanism for recovering mappings if the PCP server lost its state (e.g. crashed and rebooted). This mechanism allows PCP clients to recover mappings within seconds. Otherwise, the PCP clients would not know that the state of the PCP server was reset, until they send the next keepalive, which may be a long time ahead depending on the keepalive interval. This would encourage applications to request shorter keepalive intervals, which would increase network load.

Once the PCP server reboots, it resets its epoch time to zero and sends PCP ANNOUNCE response to PCP clients to multicast address `224.0.0.1:5350` (or `[ff02::1]:5350` in case of IPv6). A PCP client, having received the PCP response with invalid epoch time (set to zero), determines that the PCP server lost its state. The PCP client then sends a PCP request with the appropriate opcode (MAP or PEER) and include the external IP address and port assigned by the PCP server before losing its state, reminding the server of the mapping assigned.

The PCP ANNOUNCE request can be used by the PCP client to determine whether the PCP server is running or maintains its state (by checking the epoch time). PCP ANNOUNCE messages do not contain any opcode-specific fields [18].

3.1.6. Options

This section briefly introduces the available options for PCP messages. A generic format for options is shown in Figure 3.6. *Option Code* determines the option type, *Option Length* defines the length of the option header and *Option Data* contains option-specific data.

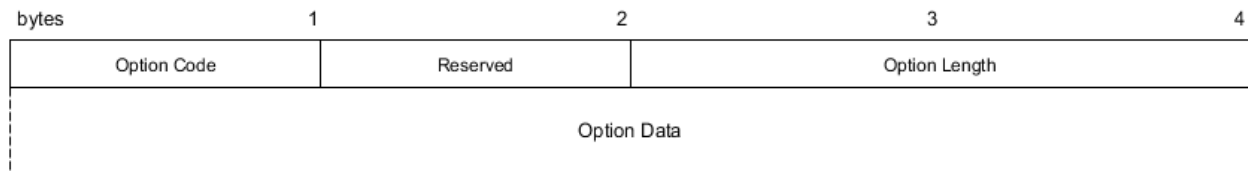


Figure 3.6: Generic header for PCP options

THIRD_PARTY

This option allows the PCP client to handle PCP requests on behalf of another host (specified by the internal IP address in the option data). It is recommended to not use this option due to security concerns. This option is valid for MAP and PEER opcodes [18].

PREFER_FAILURE

Without this option, if the PCP client requires a specific external IP address and port to be assigned and the PCP server cannot comply to the request, the PCP server assigns different external IP address and port. With this option, the PCP server will not create a mapping with explicit external IP address and port if it cannot create it and returns a PCP response indicating failure.

This option may be necessary in scenarios where the user application must explicitly specify an external IP address or port. It is expected that, with the potentially wider deployment of PCP in networks, this option will be deprecated in the future. This option is valid for MAP and PEER opcodes [18].

FILTER

This option allows the PCP client to filter incoming traffic with unwanted IP addresses and ports. To filter the traffic, the PCP client specifies the remote peer IP address and port which the application allows to receive traffic from. All other traffic is filtered by the middlebox. The FILTER option also allows specifying an entire subnet of remote hosts to be permitted (by using the *Prefix Length* field).

The FILTER option is useful for mobile devices that have to change their connection state solely for the purpose of rejecting unwanted traffic. Using the FILTER option prevents this and consequently saves battery life of mobile devices. Connection states and their impact on the battery life of mobile devices is further discussed in chapter 4.

In practice, the FILTER option can be used by applications hosting a server whose public IP address and port are known to other hosts on the Internet. One such case may be a game server that is published in a list of available game servers that the players (other client devices) can connect to. Filtering unwanted traffic or restricting the traffic to specific players can be desirable.

This option is valid for the MAP opcode only [18].

3.2. PCP Request Processing by PCP Server

This section contains an overview of how the PCP server processes a PCP request. Several details and edge cases were omitted to focus on the most important aspects of PCP message processing.

RFC 6887 contains detailed information about PCP message processing [18].

The basic processing steps are shown in Figure 3.7. A request may be invalid due to incorrect version or invalid length of the message. In that case, the PCP server sends a PCP response back to the PCP client with the result code corresponding to the type of failure. If a request is valid, the PCP server processes the opcode-specific data and options, if any. Finally, the PCP server builds a PCP response according to the flowchart in Figure 3.8 and sends it toward the PCP client. The processing of MAP and PEER opcode data is shown in Figure 3.9.

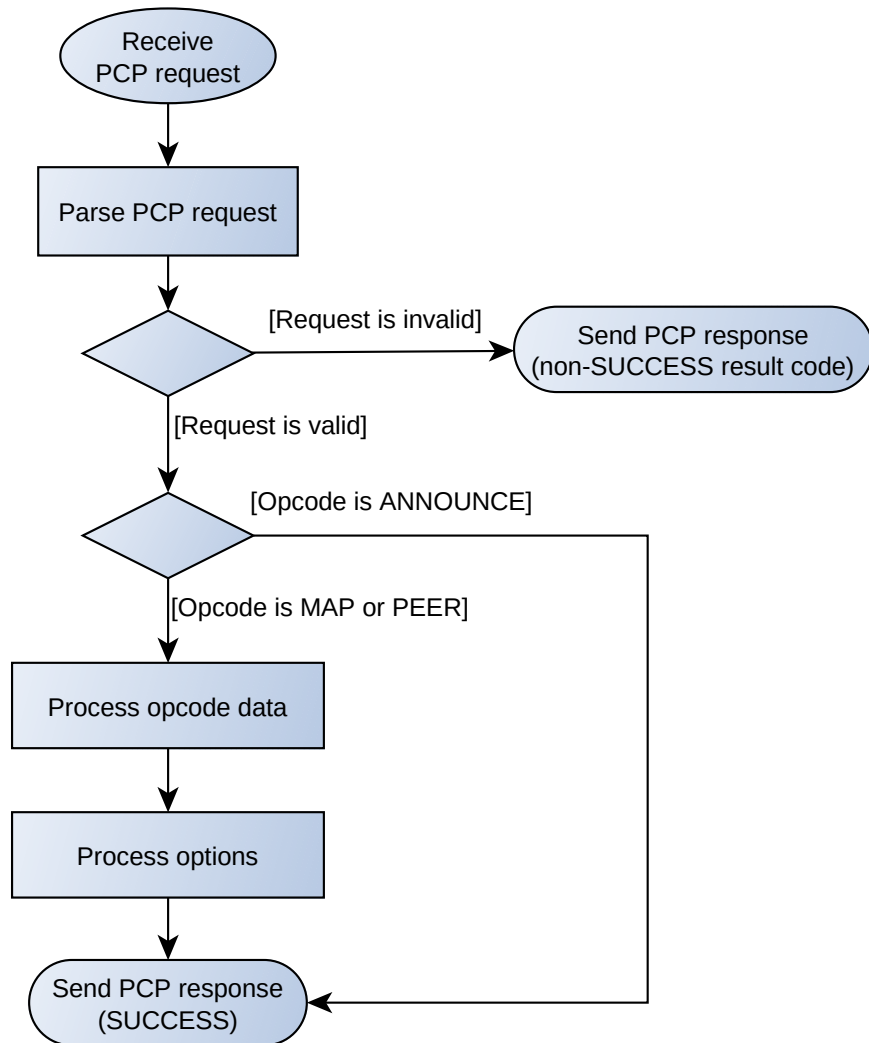


Figure 3.7: Basic processing of a PCP request by the PCP server

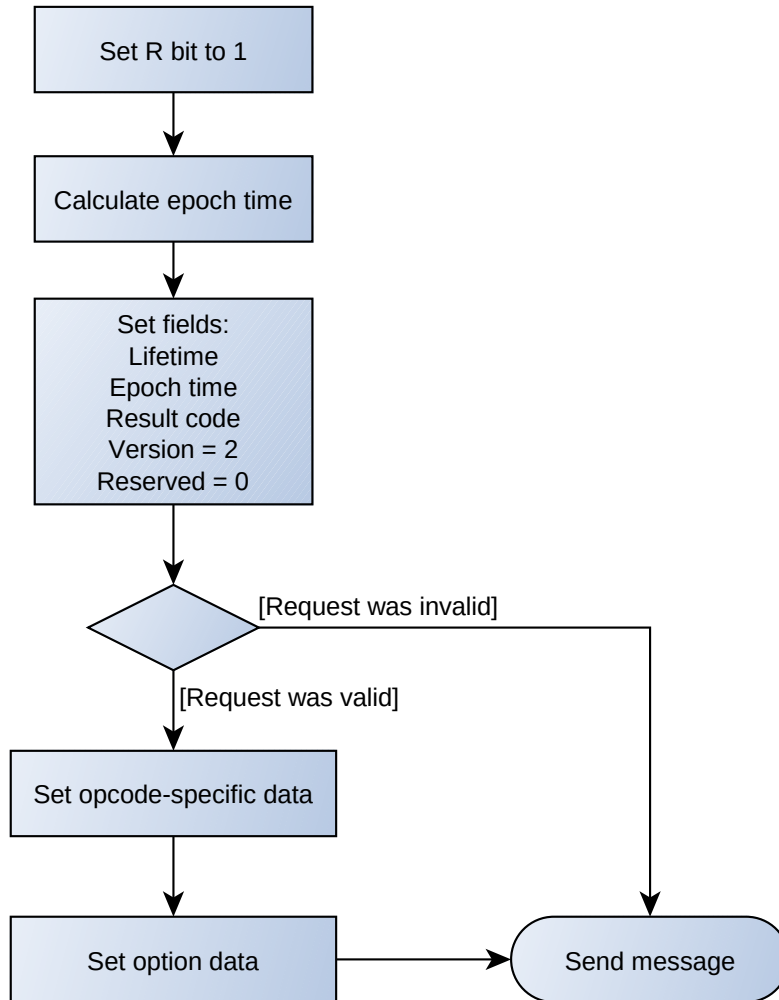


Figure 3.8: Building a PCP response from the corresponding PCP request by the PCP server

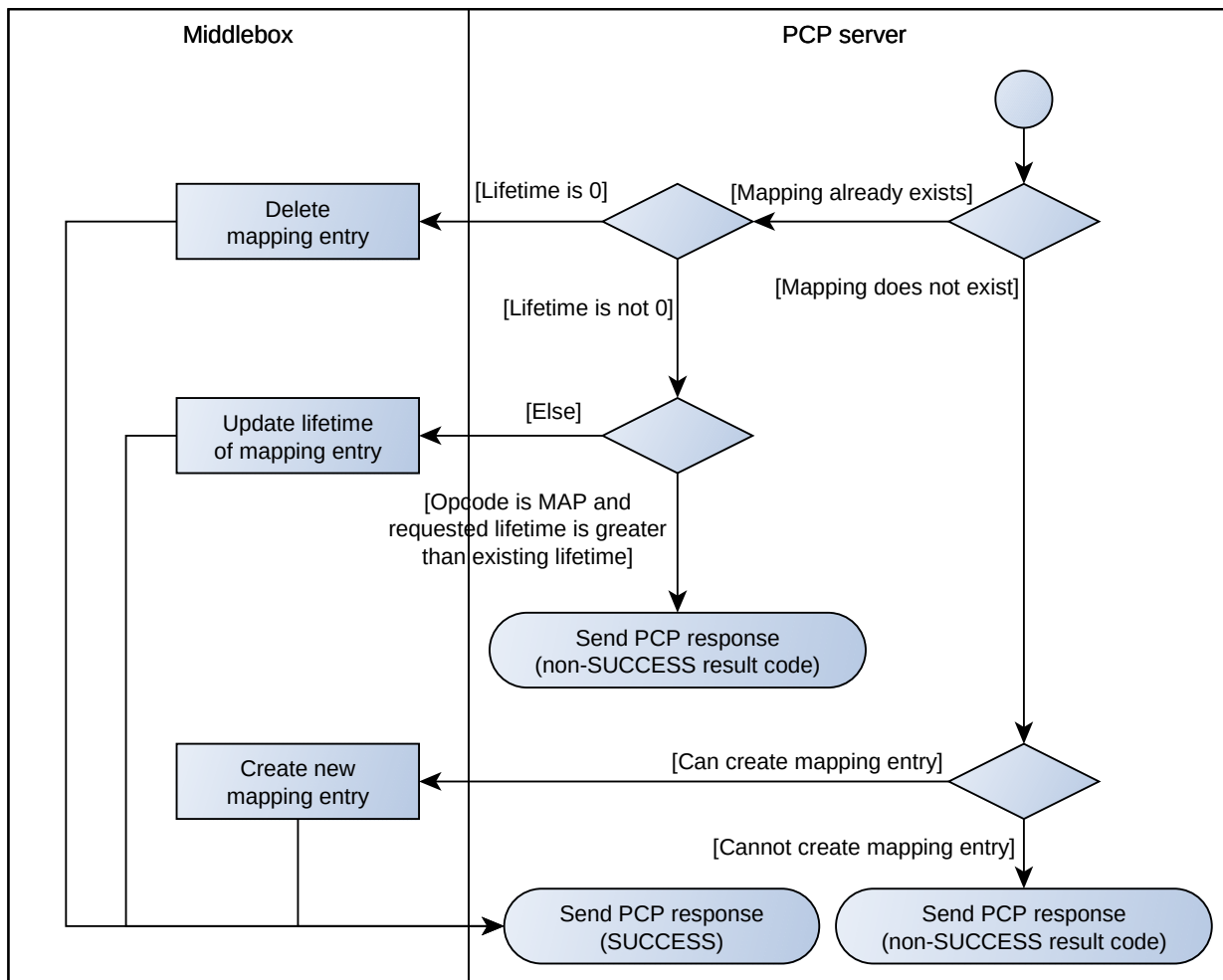


Figure 3.9: Processing of PCP MAP and PEER requests

Creating, updating or deleting a mapping entry is the responsibility of the underlying middlebox. The PCP server provides the middlebox with mapping information according to its configuration. If the middlebox created a new mapping, the PCP server receives the created mapping and sends it back to the PCP client.

If the external IP address is 0, the PCP server orders the middlebox to allocate an external IP address. The same applies to the external port.

If the PCP client explicitly specified an external IP address, the PCP server checks whether the middlebox can create a mapping with such IP address. If not, the PCP server then allocates a valid external IP address. If the `PREFER_FAILURE` option is specified, the PCP server will not allocate a different IP address and instead sends back a PCP response with `CANNOT_PROVIDE_EXTERNAL` result code [18].

3.2.1. Learning, Modifying and Maintaining Mapping Lifetime

If a user application established connection with a remote host without negotiating a mapping entry from the PCP server first, the PCP client can then send the PCP PEER request to learn the mapping lifetime and the application can thus optimize the keepalive interval.

PCP-PEER-created mappings

Using the PCP PEER request, the PCP client can extend the mapping lifetime (i.e. maintain the mapping). The PCP PEER request cannot be used to reduce the mapping lifetime or delete the mapping [18]. To delete the mapping, the PCP client and the PCP server have to let the mapping entry on the middlebox expire on its own.

To maintain the mapping, the PCP client should send PCP PEER requests regularly to the PCP server. The interval of sending PCP requests is $1/2$ to $5/8$ of the lifetime (randomly chosen). Sending PCP PEER requests to maintain the mapping is not mandatory – if the PCP client does not send PCP PEER requests, the mapping behaves as a mapping implicitly created by the middlebox [18].

Assuming that implicitly created mappings on middleboxes that were once maintained by the PCP server use the same lifetime, it is sufficient for the PCP client to send keepalives to the remote host (without sending any PCP PEER requests to the PCP server).

PCP-MAP-created mappings

Mappings created by PCP MAP requests can only have their lifetime reduced or deleted.

To maintain a MAP-created mapping, the PCP client must send MAP requests to the PCP server in the same interval as PCP PEER requests mentioned above. Additionally, as per the requirements of the user application, the application may have to send keepalives to the remote host (e.g. to check for connectivity) [18].

3.3. PCP Server Discovery

PCP clients need to know the address of the PCP server in order to be able to request mapping information. The following alternatives are suggested in RFC 6887:

- PCP clients configure the PCP server address manually,
- PCP clients receive one or more PCP server addresses via DHCP. RFC 7291 provides direct support for this approach [23].
- PCP clients assume that the PCP server IP address is the address of the host's default router (default gateway).

A related RFC draft suggests sending a PCP anycast address to discover PCP servers in the network [24].

3.4. PCP Client Implementation

RFC 6887 does not specify how the PCP client should be implemented – whether as an OS-level service or in each user application individually.

Implementing custom modifications of the PCP client outside the scope of the PCP RFC standard should be avoided since it is not expected that custom features could be widely deployed in PCP clients.

Implementing the PCP client as an OS-level service requires that operating systems support the PCP client service. For practical reasons, it is not expected that the current or older versions of

operating systems (desktop and mobile alike) will add support for the PCP client service. Each user application would still have to be PCP-aware in that it would have to interact with the PCP client service.

It is more practical to implement the PCP client per-application. The potential disadvantage is the fact that each application may have to determine the PCP server address individually. Using the DHCP approach mitigates this shortcoming, as it allows any application to determine the PCP server address.

3.5. Available PCP Software

This section gives a brief overview of chosen software that implements PCP.

3.5.1. PCP Testing Tool

*PCP Testing Tool*⁹ is a software tool consisting of two components – a web application and a PCP client.

The web application allows the user to specify a PCP message by filling in individual fields. Once the user specified the fields and confirmed to create the message, the web application generates a message containing XML-formatted fields of a PCP message to the PCP client. The IP address and port of the PCP client that listens to the requests from the web application can be configured. The web application requires *Apache* web server and *PHP* to run.

The PCP client is a PCP client implementation of PCP. If the PCP client receives a message from the web application, the PCP client constructs a PCP request from the message to the PCP server specified in the corresponding configuration file. The configuration file allows to specify IP addresses in IPv6 format – if an IPv4 address needs to be specified, it must be specified as an IPv4-mapped IPv6 address.

Given its web interface, this tool is user-friendly in the sense that it easily allows the user to build a PCP message. The disadvantage of this tool may be that it does not provide a command-line interface to build messages, which may be useful when automating the building of PCP messages (e.g. via shell scripts).

3.5.2. PCP Client Library

The PCP client library¹⁰ contains a library of functions implementing the PCP client that can be integrated in user applications, a lightweight command-line PCP client application and a module that allows to build packets using the *scapy*¹¹ command-line packet builder.

With the application, the user can send a PCP request to a PCP server. An example usage of this application is shown below:

```
pcp -i <internal host IP address>:<internal port> -s <PCP server  
address> -l 3600
```

The `-i` option specifies the address of the internal host that is to be mapped to an external

⁹ Available at: <http://sourceforge.net/projects/pcptestingsuits/>

¹⁰ Available at: <https://github.com/libpcp/pcp>

¹¹ Available at: <http://www.secdev.org/projects/scapy/>

address, and an internal port to be mapped to an external port. While it may seem redundant to specify the internal IP address, the application will not work without it. This is also necessary to specify in case the host has multiple IP addresses (multiple interfaces). The `-l` option specifies the requested mapping lifetime in seconds.

The command above generates a PCP MAP request for TCP. To specify a PCP PEER request, the remote peer IP address and port have to be specified:

```
pcp -i <internal host IP address>:<internal port> -s <PCP server  
address> -l 3600 -p <remote peer IP address>:<remote peer port>
```

The `-u` option creates a mapping for UDP. An explicit IP address and port can be specified by the `-e` option:

```
pcp -i <internal host IP address>:<internal port> -s <PCP server  
address> -l <lifetime> -e <external IP address>:<external port>
```

3.6. Comparison of PCP and Middlebox Traversal Methods

Compared to similar protocols or middlebox-traversal methods, PCP has the following advantages:

- PCP can optimize keepalive traffic,
- PCP can resolve NAT traversal issues and eliminate the need to deploy ALGs [18],
- PCP has a simple protocol design.

PCP imposes the following requirements on the network, which may be seen as disadvantages:

- each user application must implement a PCP client,
- each middlebox that is supposed to be PCP-aware must run a PCP server.

While UPnP-IGD allows end users to configure mapping information, even programatically, PCP transmits fewer messages (therefore is more bandwidth-efficient) and does not need to be configured by users [3].

While the original PCP RFC [18] states that PCP can be used for simple firewalls, a relatively recent RFC draft has been published that adds support for new PCP message types that support advanced firewall functionality in managed networks, such as software defined networks (SDN) [25].

4. Mobile Networks

Mobile networks allow end users to connect to the Internet and communicate with each other in a wireless manner using mobile devices. Over the decades, several generations of mobile networks have been developed and deployed to cope with the increasing demand of users staying connected while moving.

The first generation of networks (developed in the 1980s) allowed users to establish phone calls. The data transmission in 1G networks was analog, unlike later generations, which used digital data transmission.

2G networks originated in the later 1980s, of which Global System for Mobile Communications (GSM) became the most popular and widespread 2G technology. Despite newer mobile network technologies, GSM is still widely used in the present time due to its widespread coverage and network stability. GSM uses digital data transmission to allow users to establish phone calls and send SMS messages. General Packet Radio Service (GPRS) is a 2G technology deployed over GSM that enables packet-switched transmission of data. Enhanced Data rates for GSM Evolution (EDGE) is another popular 2G technology that improves transmission data rate compared to GPRS.

3G networks, of which Universal Mobile Telecommunications System (UMTS) is the most widely adopted technology, allow higher data rates than 2G networks. UMTS employs the Wideband Code Division Multiple Access (WCDMA) radio access technology. High Speed Packet Access (HSPA) technologies improve the data rate even further to a few tens of Mbit/s [26].

Long-Term Evolution (LTE) is a relatively new mobile technology that further improves the transmission data rate, reduces round-trip time and reduces cost for provisioning networks [26].

The rest of this chapter focuses on 3G networks, particularly on the WCDMA access technology used in UMTS. In WCDMA networks, mobile devices are in different connection states depending on the amount of data to be transmitted. Transitioning to a different connection state causes a considerable number of signaling messages to be generated. Moreover, connection states in which mobile devices transmit data reduce their battery life. WCDMA thus proves to be a source of continuous research on how to improve the efficiency of the network and preserve battery life of mobile devices connected to the network. Connection states are discussed in more detail in section 4.2.

4.1. 3G Networks

Figure 4.1 shows the architecture of 3G networks¹². Mobile devices, known also as *user equipments* (UE), connect to the Internet through the radio access network, UTRAN. A UE communicates wirelessly with a base station, *Node B*. Multiple Node B stations are connected to and managed by a single Radio Network Controller (RNC).

RNCs are connected to the core network, which is responsible for forwarding traffic to the desired destinations and for managing subscribers. Mobile Switching Center (MSC) manages

¹² More information at: <http://www.radio-electronics.com/info/cellulartelecomms/umts/umts-wcdma-network-architecture.php>

circuit-switched connections, such as phone calls. Gateway MSC (GMSC) acts as a public interface between the network core and a telephone network (Public Switched Telephone Network, PSTN).

Serving GPRS Support Node (SGSN) is responsible for mobility management, session management (establishing and managing data sessions known as PCP contexts) and billing.

Gateway GPRS Support Node (GGSN) acts as an interface between the core network and the external packet-switched networks (packet data networks, PDN). From the perspective of the Internet, GGSN acts as an IP router. For traffic directed toward a UE, GGSN determines the corresponding SGSN that currently manages the UE.

Other nodes in the core network include Home Location Register (HLR), which is a database containing information about each subscriber; VLR (Visitor Location Register), which is a subset of HLR and is used in areas the UE is visiting, and EIR (Equipment Identity Register), which checks whether a UE is allowed to access the network.

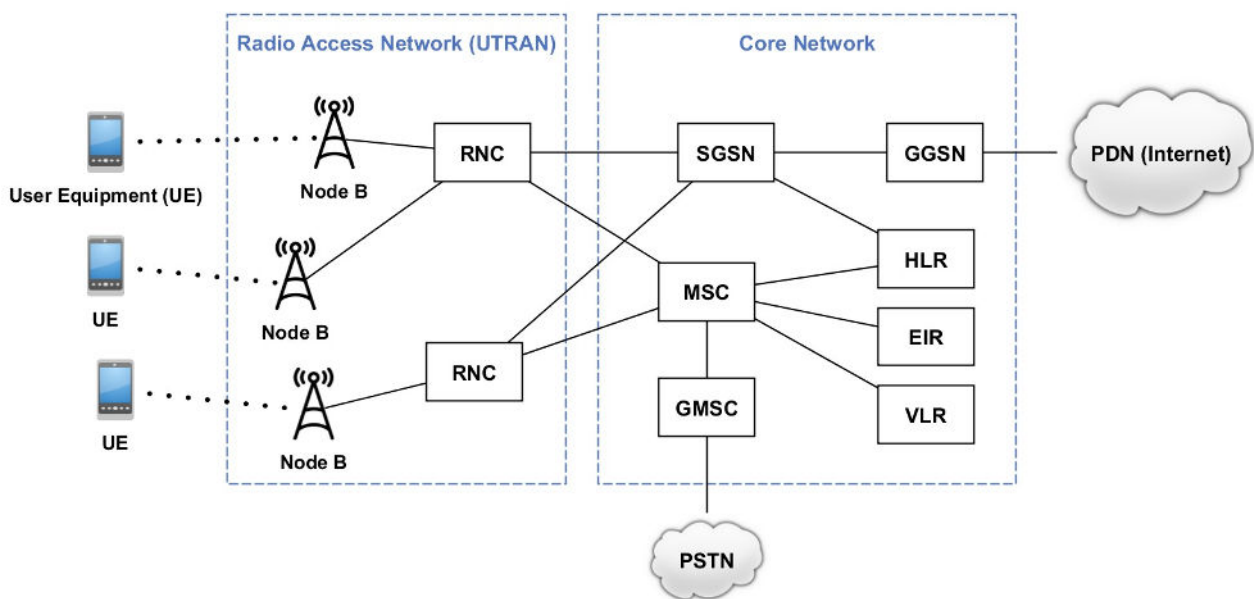


Figure 4.1: Architecture of 3G networks

In the radio network, each Node B covers a certain area, known as a *cell*, with its wireless signal. When a UE is moving from one cell to another, *handover* is performed, which transfers control of the UE from one Node B to another. To track the location of a UE within a cell, Node B establishes a communication with the UE, also known as *paging*.

4.2. Radio Resource Control

Radio Resource Control (RRC) is a protocol in WCDMA networks that manages signaling between a mobile device and the radio access network, UTRAN [27] [5]. RRC, among the numerous functions it performs [27], provides establishment, maintenance and release of an RRC connection and its associated radio resources between a mobile device and the radio access network, and also paging. RRC states also apply to HSPA technologies [28].

4.2.1. RRC States

Figure 4.2 shows the possible RRC states and transitions that can occur for a mobile device [27] [5] [29]. When the device is not connected to the network, it is in the *RRC Idle* mode. Once the device wishes to establish a connection with a remote host, the RRC connection between the device and the network is established first, and the device is now in the *RRC Connected* mode. Table 4.1 shows the average power consumption of a device in each state.

CELL_DCH (*Dedicated Channel*) state is used when the device transmits data over the network, unless the amount of data is very small. For the CELL_DCH state, the network allocates a dedicated data channel for the mobile device [5]. The average power consumption of a device in this state is the highest, as shown in Table 4.1.

In **CELL_FACH** (*Forward Access Channel*) state, the device shares a channel with other devices. This state is used if there is a small amount of data to be transmitted by the device. Source [30] states that a data rate low enough to be transmitted in the CELL_FACH state is up to 64 kbit/s for downlink transmissions and up to 8-16 kbit/s for uplink transmissions, although these values are dependent on the implementation of the RNC. If the data rate (traffic volume) exceeds a defined threshold, the devices transitions to the CELL_DCH state [31].

In **CELL_PCH** (*Paging Channel*) state, the device is not capable of sending or receiving packets. This state is used by the network for paging. If a packet is sent towards the device, the device enters the CELL_FACH or CELL_DCH state. This state consumes very little battery power compared to the CELL_FACH or CELL_DCH states. Not all networks currently use the CELL_PCH state [5] [30].

URA_PCH (*UTRAN Registration Area Paging Channel*) state, similar to the CELL_PCH state, does not allow data to be transmitted. URA_PCH is beneficial in cases where the device is moving fast and changing cells frequently as a result^{13,14} [29]. URA_PCH has approximately the same power consumption as CELL_PCH. Given that the URA_PCH state is not known to be implemented in mobile networks [5] [30] and it can be considered equivalent to CELL_PCH in terms of power consumption and inactivity timers [30], this state is not further referenced in this document.

In *RRC Idle* mode, the device does not have an RRC connection, but the network can still communicate with the device via paging. The power consumption is comparable to that of CELL_PCH and URA_PCH states [5].

13 More information at: http://www.telecomsource.net/showthread.php?2428-Difference-between-URA_PCH-and-CELL_PCH

14 More information at: http://www.telecomsource.net/showthread.php?1737-What%20is%20URA%20and%20URA_PCH%20state?

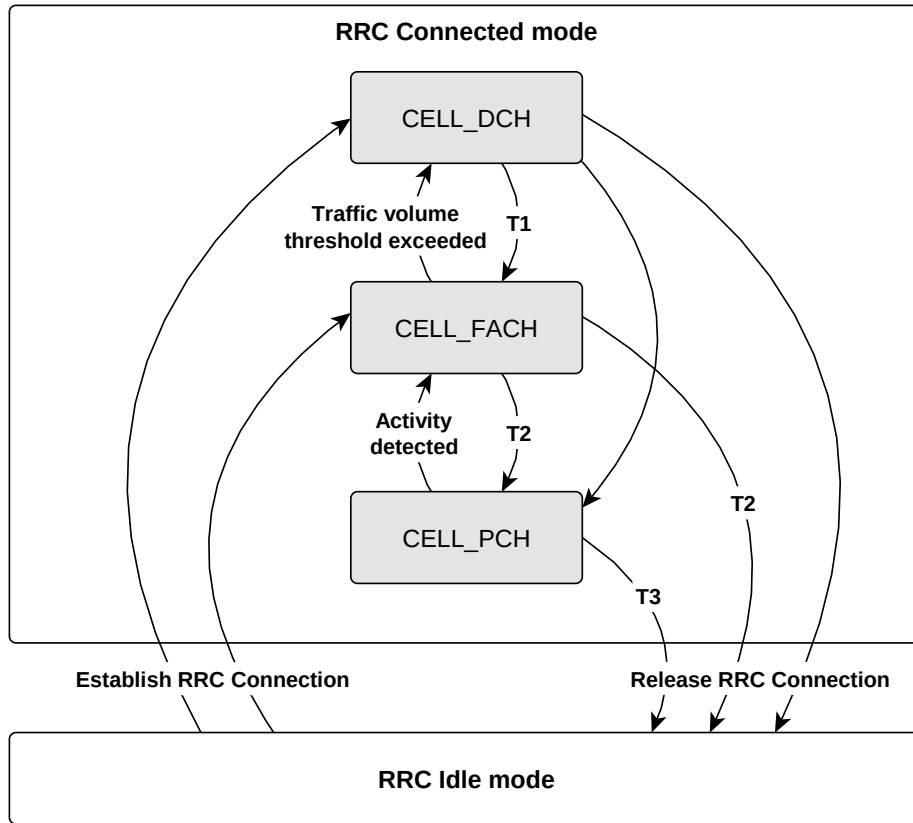


Figure 4.2: Radio Resource Control (RRC) states

Table 4.1: Average current (power consumption) of a mobile device in RRC states [30] [5] [29]

RRC State	Power consumption
CELL_PCH, URA_PCH	approx. 5 mA
CELL_FACH	100 – 150 mA
CELL_DCH	200 – 400 mA

4.2.2. RRC Inactivity Timers

If the device persists in an RRC state for a certain amount of time without sending any data, it descends to a lower-power RRC state. Each such transition is associated with an *inactivity timer*. These inactivity timers, as shown in Figure 4.2, can be referred to as T1, T2 and T3 [30] [5].

T1 timer is used in the CELL_DCH state. If the connection is idle for T1 seconds, or the data rate is low enough, the device transitions to CELL_FACH state. The data rate threshold is dependent on the concrete implementation of the RNC in the network. If there is traffic exceeding the threshold data rate, the T1 timer is reset and the device remains in the CELL_DCH state. Typical values for the T1 timer range up to 5 seconds [30] [5].

T2 timer is used in the CELL_FACH state. If no packets are sent over T2 seconds, the device transitions to the CELL_PCH state. In case the network does not support the CELL_PCH state, the device releases its RRC resources and enters the *RRC Idle* mode. As with T1, typical values for T2

timer range up to 5 seconds [30].

If no packets are sent and the device stays in the CELL_PCH state for T3 seconds, the device transitions from the CELL_PCH state to *RRC Idle* mode and releases its RRC resources. T3 value ranges typically from a few minutes to a few tens of minutes [30] [5].

4.2.3. RRC State Transitions with Keepalives

If a device sends a single keepalive message toward a destination host, there are a few possible state transitions to consider, which are illustrated in the decision tree in Figure 4.3. Transitioning to CELL_FACH or CELL_DCH depends on the network configuration. It is assumed that, before sending a keepalive, the device is in *Idle* mode or CELL_PCH state and that no other data packets are sent over the network at that time.

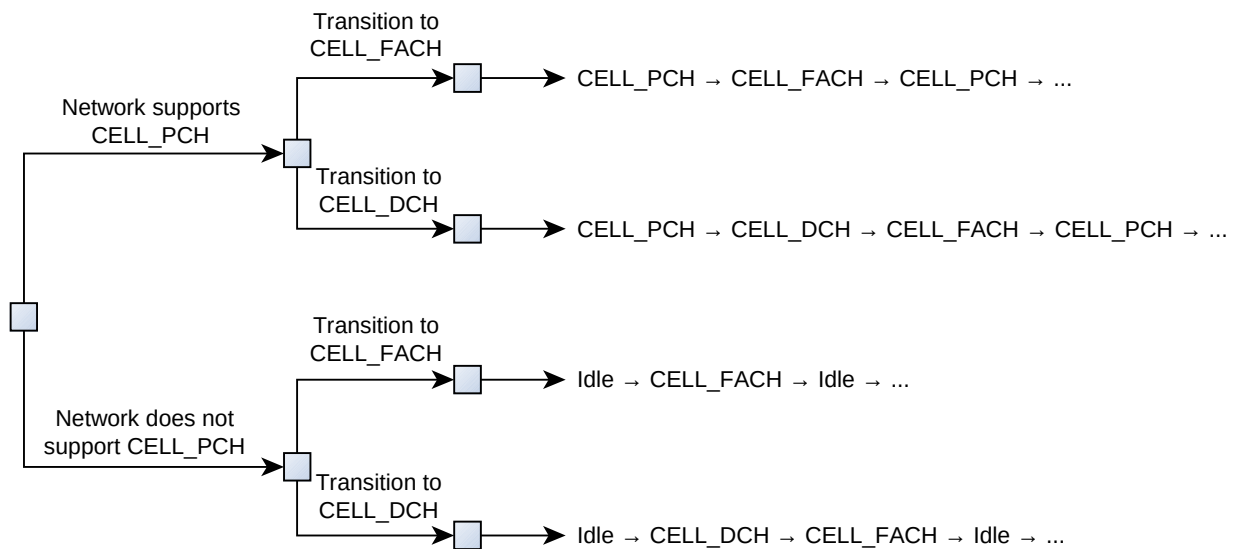


Figure 4.3: Possible RRC state transitions when sending a single keepalive

If a user application sends a keepalive that should be acknowledged (such as a TCP keepalive, or an application keepalive over UDP), then the device may have to return to a higher RRC state or repeat the cycle of state transitions again, depending on the round-trip time of that keepalive. In case the round-trip time is greater than the timer for the active RRC state the device is currently in (CELL_DCH or CELL_FACH), the device transitions to a lower state and, upon receiving the acknowledgment, back to the higher state. This increases the power consumption of the device and generates more signaling traffic due to more state transitions.

4.3. Conclusions

With an increasing amount of smartphones connected to mobile networks, the amount of signaling traffic increases significantly, especially considering the widespread usage of always-on applications such as social networks or instant messaging. One cause of the increased signaling overhead is the frequent transmission of keepalives, which generates signaling messages due to RRC state transitions of mobile devices. The increased signaling overhead imposes considerably

higher processing requirements on mobile network elements and may eventually cause slower data rate or network congestion.

Due to sending keepalives frequently, mobile devices remain longer in the active RRC states (CELL_DCH and CELL_FACH), which contributes to higher battery power consumption of mobile devices.

A new concept in the field of computer networks called *software defined networking* (SDN) has a great potential of improving the effectiveness and scalability of computer networks, including mobile networks. SDN is discussed in the next chapter.

5. Software Defined Networking

Software defined networking (SDN) is a relatively new concept in the field of computer networks. SDN emerged from the requirements of enterprises and end users that today's traditional networks cannot cope with [32] [33].

The main idea of SDN is to separate the control and the data planes of network nodes. The control plane is logically centralized in an element called the *controller* and the data plane remains in the network nodes, which are now called *forwarders*. Forwarders usually perform packet forwarding and basic packet processing, such as overwriting fields in packet headers [34].

Figure 5.1 shows the basic architecture of SDN networks [34]. The controller manages forwarders via a standard communication interface, such as *OpenFlow*¹⁵ or *NETCONF* [35]. This is also called the *southbound interface*. *Network applications* are software programs that define the network behavior (hence the term “software defined networking”), ranging from very basic programs, such as packet switching, to more advanced applications, such as a firewall. The controller accepts requests from the network applications and translates them to low-level commands that the forwarders are able to process. Each network application communicates with the controller via a separate *northbound interface*.

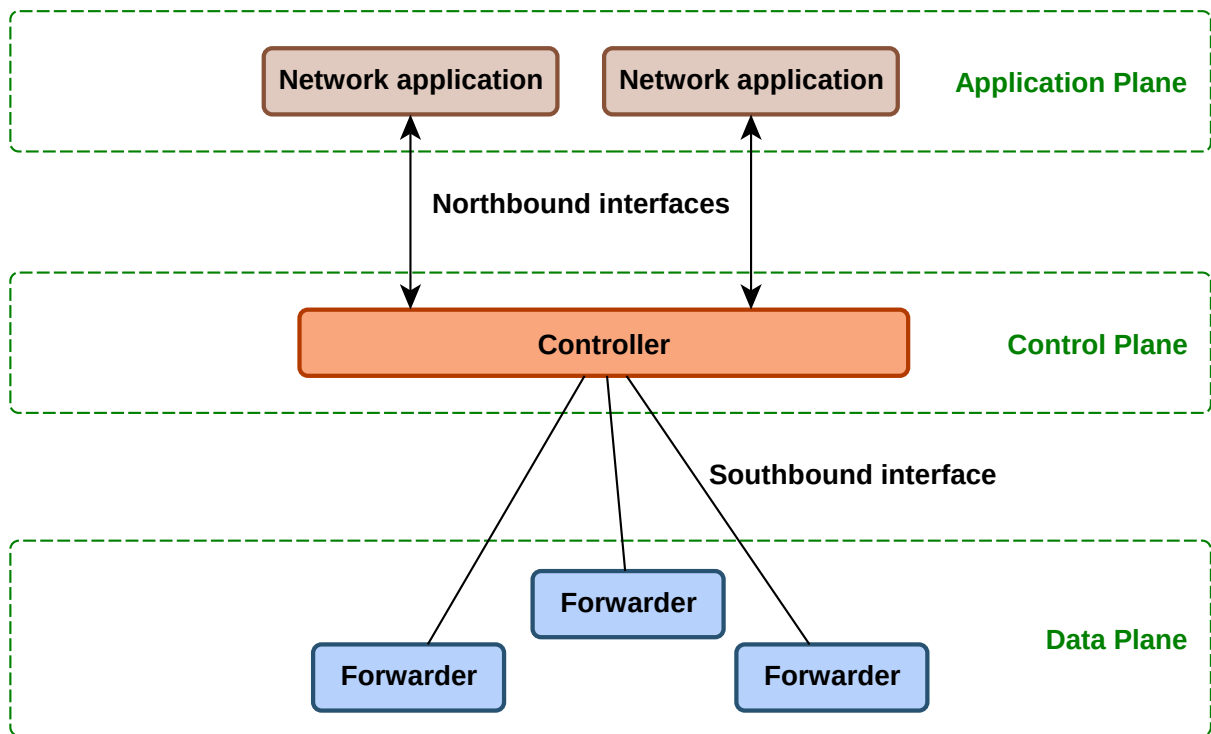


Figure 5.1: SDN architecture overview

In general, SDN networks have the following advantages over traditional networks [32] [33]:

¹⁵ More information at: <https://www.opennetworking.org/sdn-resources/openflow/57-sdn-resources/onf-specifications/openflow?layout=blog>

- ease of adding new or modifying existing network applications,
- improved automation and management of network devices,
- given the standard communication protocol between the controller and the forwarders, network devices from multiple vendors can be deployed in networks,
- improvement of the user experience due to the ability of SDN networks to easily adapt to the needs of end users.

Migration from a traditional network to a pure SDN network may be costly. Is it not uncommon to for *hybrid SDN networks* to exist that contain a mix of traditional network elements, forwarders and one or more controllers [32].

5.1. OpenFlow

OpenFlow is a popular standard for communication between a controller and forwarders. Forwarders are referred to as *OpenFlow switches*. OpenFlow protocol and switch are defined by the OpenFlow switch specification [36].

At the time of writing this thesis, the most recent version of the OpenFlow specification is 1.5.1 [37]. Given that the solution described in this thesis uses SDN controller and forwarder software compatible with OpenFlow 1.3.0, this section discusses the OpenFlow specification 1.3.0 [36]. This section covers only the components of an OpenFlow switch used in the thesis.

5.1.1. OpenFlow Switch Overview

Figure 5.2 shows the basics of an OpenFlow switch. The switch communicates with the controller via the OpenFlow protocol. When a packet is received on an ingress port, it is processed through a set of *flow tables*. The packet processing is also called the *pipeline*. Each flow table contains a set of rules called *flow entries*.

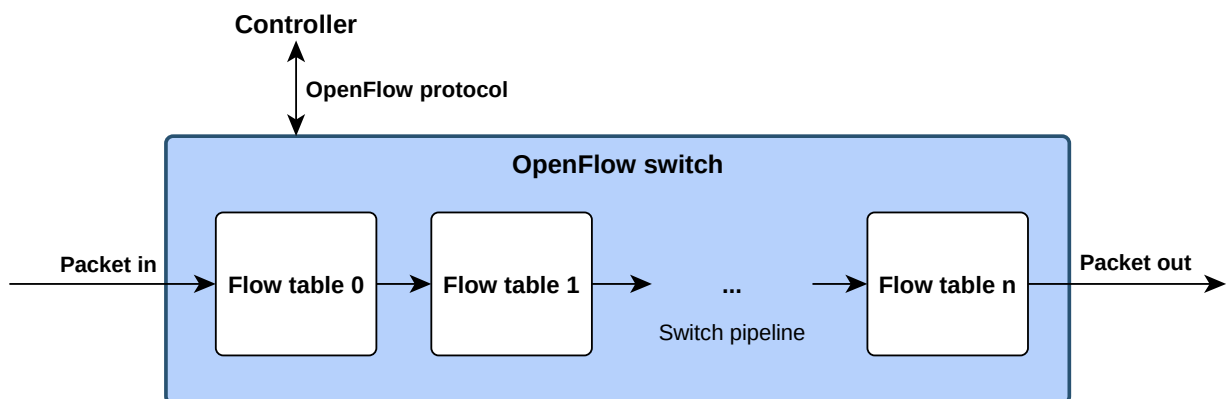


Figure 5.2: Overview of an OpenFlow switch (forwarder)

Datapath refers to a part of the OpenFlow switch that comprises the ports, flow tables and the pipeline. *Control channel* refers to the communication interface between the switch and the controller. Each datapath is defined by a unique 64-bit *datapath ID*. The lower 48 bits define the

MAC address of the switch and the upper 16 bits are implementation-specific [36].

5.1.2. Communication with Controller

The communication of the OpenFlow switch and the controller is provided by the control channel. Using the control channel, the controller and the switch exchange control messages or forward data packets to and from the controller.

The message delivery between the controller and the switch is guaranteed. There is no need for a network application to explicitly check and acknowledge that a message sent from the controller to the switch was received successfully.

Features message is sent by the controller during the establishment of the control channel to query the switch about its capabilities. *Modify-State* messages are sent by the controller to add, modify or remove flow entries or to modify the properties of ports. *Packet-Out* messages are used by the controller to send packets through the switch to the destination. *Packet-In* messages are sent by the switch to forward packets to the controller (via the reserved `CONTROLLER` port).

If a controller installs a flow entry on a switch with the `OFPPFF_SEND_FLOW_REM` flag set and the flow entry expires or is deleted by the controller, the switch informs the controller that the flow entry has been removed by sending a *Flow-Removed* message to the controller.

5.1.3. Flow Entries

Each flow entry contains components shown in Figure 5.3.

Match fields	Priority	Counters	Instructions	Timeouts	Cookie
--------------	----------	----------	--------------	----------	--------

Figure 5.3: Flow entry structure in an OpenFlow switch

Match fields determine whether a packet matches the flow entry. If so, the *instructions* for this flow entry are executed. Match fields may contain the following [36]:

- ingress port ID,
- header fields (e.g. *EtherType* from the Ethernet frame header, *Time To Live* from the IPv4 header),
- metadata specified by a previous table.

A match for a flow entry must contain all its associated prerequisites. For example, to match a UDP datagram with a specified destination port, the match must also contain the corresponding lower-layer protocols – in this case, *EtherType* field for IPv4 (`0x0800`) and *IPv4 Protocol* field for UDP (`0x11`). If the prerequisites are not specified, the switch sends an error message to the controller with `OFPET_BAD_MATCH` type and `OFPBMC_BAD_PREREQ` flags set. A flow entry that matches any packet and has priority equal to 0 is called the *table-miss* flow entry.

Priority defines the precedence of a flow entry. If a packet matches multiple flow entries, only the flow entry with the highest priority is considered and other entries are discarded. If there are

multiple flow entries with the same priority, the behavior of the switch is undefined (not even OpenFlow 1.5.1 defines this behavior). It is assumed that this behavior is defined by a concrete implementation of the switch software. The controller may prevent adding flow entries with the same priority and intersecting matches by setting the `OFPPF_CHECK_OVERLAP` flag in *Modify-State* messages.

Counters are incremented every time a packet matches the flow entry. Examples of per-flow-entry counters include the number received packets or received bytes. Other types of counters are defined e.g. per flow table or per port.

If a packet matches the flow entry, a set of *instructions* is executed. Instructions are discussed in section 5.1.4.

Timeouts are optional and are used by the controller to set the flow entry to expire after a specified amount of time in seconds. If `hard_timeout` is set, the flow entry expires after the specified number of seconds since its addition to the switch. If `idle_timeout` is set, the flow entry expires after the specified number of seconds if no packet matched this flow entry for `idle_timeout` seconds.

Cookie is a value that the controller associates with the flow entry. Cookie can be used by the controller to filter messages dealing with flow modification, flow deletion or flow statistics.

5.1.4. Instructions

Generally speaking, instructions modify the packet, the set of actions or the pipeline processing. Commonly used instructions are the following:

- *Write-Action* – adds the specified actions into the action set.
- *Apply-Actions* – applies the action set to the packet immediately, without modifying the action list. This can be used to execute the action set multiple times.
- *Clear-Actions* – clears the action set.
- *Goto-Table* – causes the pipeline to jump to the flow table specified by its ID. The flow table ID must be greater than the current flow table ID.

Actions

The *action set* is a list of *actions* applied to the packet. The action set can be modified by instructions, such as those mentioned above. Some commonly used actions include:

- *Set-field* – modify a header field. While this action is not specified as mandatory in the OpenFlow switch specification, its inclusion in the implementations of OpenFlow switches greatly improves the usefulness of the switches. The *set-field* action allows to overwrite IP addresses and ports, thus allowing to implement a simple NAT.
- *Output* – send the packet out the specified port.

5.1.5. Ports

OpenFlow defines several types of switch ports, including the following:

- *Physical ports* – these ports correspond to the physical interfaces on the switch.
- *Logical ports* – these ports can be used to represent e.g. VLAN ports or loopback interfaces.
- *Reserved ports* – these ports have a special meaning in the context of packet forwarding. `CONTROLLER` port represents the port (the control channel) to the controller and can be used to forward packets to the controller. `TABLE` port can be used by controller when the controller sends a packet to a switch and desires to process the packet through the flow tables in the switch. `IN_PORT` can be used to send a packet out the ingress port on a switch.

5.2. SDN Software

This section briefly reviews existing software for SDN controllers and forwarders.

5.2.1. Forwarders

CPqD OFSoftswitch

*CPqD OFSoftswitch*¹⁶ is a forwarder implementation compatible with OpenFlow 1.3. It consists of the following main components:

- `ofdatapath` – the OpenFlow switch implementation,
- `ofprotocol` – the control channel between the switch and a controller,
- `dpctl` – a command-line tool to query information about the switch or modify the switch.

OFSoftswitch has a well-documented source code and supports timeouts and modifying packet headers (i.e. the *Set-Field* action). *OFSoftSwitch* resolves multiple matching flow entries with the same priority by considering only the flow entry added as the first^{17,18}. When installing a *Set-Field* action on a flow entry, *OFSoftSwitch* automatically recomputes the checksums of relevant headers (TCP/UDP, IP, Ethernet).

5.2.2. OpenFlow Controller Software

Ryu

Ryu is a framework to create a custom SDN controller. *Ryu* also allows to write custom network applications over the controller. *Ryu* is written in Python language – being a high-level language, it allows for rapid prototyping and easy writing of programs.

Ryu is provided with a well-documented API¹⁹. If the documentation is missing information or it is unclear how to use certain classes or functions, examining the source code is another option as it is likewise well-structured and documented. *Ryu* supports OpenFlow 1.0, 1.2 and 1.3, and also NETCONF and OF-Config 1.1.

Despite *Ryu* being a framework and not a controller per se, the available documentation and the

¹⁶ Available at: <https://github.com/CPqD/ofsoftswitch13>

¹⁷ https://github.com/CPqD/ofsoftswitch13/blob/2836522c1fdd2d5a0b759935c8b914abf41af441/udatapath/flow_table.h#L43

¹⁸ https://github.com/CPqD/ofsoftswitch13/blob/c532c3167523564d4ea9f9754628900a0e96000f/udatapath/flow_table.c#L107

¹⁹ More information at: <https://ryu.readthedocs.org/en/latest/>

ease of writing a controller and network applications compensate for the minor inconvenience of not being provided with a proper controller.

6. Analysis Summary

This chapter reviews the topics described in the analysis and discusses existing issues with middleboxes, keepalive traffic, SDN and PCP.

Figure 6.1 shows the typical deployment of PCP in traditional networks.

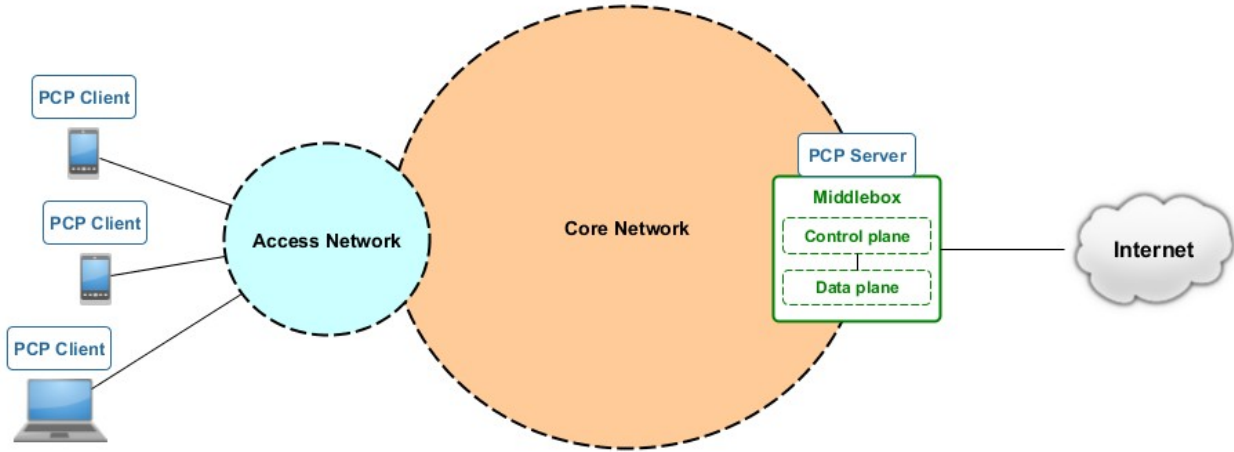


Figure 6.1: PCP Deployment in Traditional Networks

Hosts with PCP clients are connected through an access network to the core network containing a middlebox on its edge to an external network. The middlebox runs a PCP server to process requests from PCP clients.

In traditional networks without PCP, user applications have to utilize other protocols or methods to traverse the middleboxes. Software defined networking (SDN) allows to centralize the control of the network and define the network behavior programmatically. Beside other advantages, SDN increases the flexibility and vendor device compatibility in the networks. SDN as a concept is still in its early stages and is not widely deployed in networks.

In SDN networks, separating the control and the data plane of a middlebox fosters greater scalability and flexibility of the network. For example, instead of a single middlebox in traditional networks (such as a carrier-grade NAT), multiple forwarders with only the middlebox data plane can be installed, which reduces network load.

Middleboxes usually set table entries to expire over time if they are idle. In SDN networks, OpenFlow switches support flow entry expiration (by setting the `idle_timeout` field) and notifying the controller that a flow entry has expired (by enabling the `OFPFF_SEND_FLOW_REM` flag). Without these functions on the forwarders, it would not be possible to separate the control and data plane of a middlebox. In such case, if the middlebox were to be installed in an SDN network, it would have to be provided with a communication interface with the SDN controller, such as the aforementioned OpenFlow standard.

Keepalive traffic sent by user applications increases network load. Mobile networks require a substantial amount of signaling traffic for each packet sent, therefore amplifying the network load

and reducing the battery life on mobile devices.

Compared to traditional networks, implementing PCP over SDN has the following advantages:

- Middleboxes do not need to run a PCP server, thereby reducing their computation overhead.
- The lack of the PCP server on middleboxes increases vendor compatibility and avoids the need to upgrade the middleboxes or purchase new ones to support PCP server functionality.
- If multiple, separate middleboxes are placed in the network, such as a NAT gateway and a firewall, mapping lifetime does not have to be determined from each middlebox individually, but rather from one central point – the SDN controller.
- Middleboxes remain transparent to the client, because the PCP server is installed on the controller rather than on the middleboxes.

NAT gateways mitigate insufficient IPv4 address space. It is expected that IPv4 and IPv6 will coexist for several years. Even though NAT gateways may be redundant in pure IPv6 deployments, firewalls will still exist in core networks. The role of PCP is therefore still valid for pure IPv6 deployments.

7. Specification

This chapter specifies goals of the diploma thesis and requirements for the solution.

7.1. Goals

This diploma thesis aims to fulfill the following goals:

- implement PCP protocol over SDN networks,
- allow end-user applications to receive address and port mapping information directly from NAT gateways in order to be able to facilitate communication with nodes in other networks behind NAT,
- reduce the amount of keepalives sent by the end-user applications to the network in order to reduce network load and prolong battery life of mobile devices.

To verify the goals, the following will be performed:

- Using PCP, verify that a user application on a host behind NAT successfully establishes communication with another host in a public network.
- Quantify the amount of battery power saved in WCDMA networks when reducing keepalive traffic with PCP.
- Quantify the reduction of signaling messages in WCDMA networks when reducing keepalive traffic with PCP.

7.2. Requirements

The implementation of PCP must support at least the following components from the PCP RFC:

- PCP client requesting mapping information from the PCP server,
- PCP client attempting to explicitly set mapping information on the PCP server,
- PCP server processing PCP requests and sending PCP responses back to the PCP client.

The implementation of the PCP server will not support the following components from the PCP RFC:

- recovery of PCP server (e.g. after a reboot),
- processing of options in PCP messages by the PCP server.

In order to verify PCP-supported NAT traversal by end-user applications and the reduction of keepalives, NAT gateway functionality will be implemented in the SDN network. Firewall will not be implemented, because there are no expected differences in results between a NAT gateway and a firewall.

The NAT gateway should support at least the following:

- IPv4-to-IPv4 address translation,

- dynamic IP address and transport protocol port translation (NAPT),
- UDP and TCP as transport protocols.

The access network used to evaluate the signaling traffic reduction and battery life extension is WCDMA.

7.2.1. Port Control Protocol

Scalability

In networks with a very large number of connected hosts, issues with scalability in the core network may occur in conjunction with PCP. Considering that each application implementing the PCP client requests a mapping for each connection to the middlebox (which is also called per-flow mapping), a very high number of PCP messages may be sent to the network. PCP scalability concerns will not be addressed in this diploma thesis.

Security

Security of PCP is not resolved and is currently under discussion [20] [18]. An RFC draft exists that specifies PCP authentication mechanism [38]. Security of PCP will not be addressed in this diploma thesis.

8. Design

This chapter describes the design of the solution. The solution comprises a network application implemented in an SDN core network.

8.1. Architecture

Figure 8.1 shows the architecture of the network. Generally speaking, there is no dedicated middlebox in the proposed design. The control plane of a middlebox resides in the controller as a network application, and the data plane of a middlebox resides in a forwarder. In case of NAT gateway functionality in the network, the control and data plane is split to the NAT control plane and the NAT forwarder, respectively.

While the architecture could be generalized to any middlebox, including a firewall, other middleboxes may have their own specifics that would have to be integrated into the architecture.

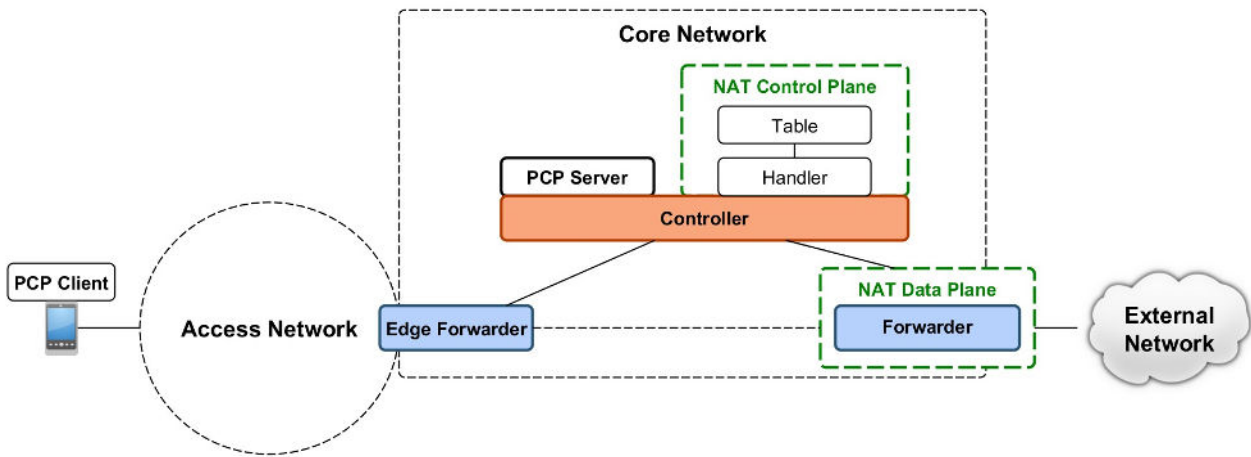


Figure 8.1: Architecture of the network

The architecture assumes that only one NAT forwarder is installed in the network. Therefore, only one NAT table is maintained in the NAT control plane and there is no need to use unique identifiers for each pair of a NAT forwarder and a NAT table.

The network application, comprising the PCP server and the NAT control plane, is integrated into the controller. The communication between the controller and the network application is restricted to an application programming interface (API) for better source code manageability.

The architecture is not concerned with IP routing to the external network and with packet forwarding from the core network to hosts through the access network. These network features are redundant for the evaluation of the solution and are therefore omitted from the architecture.

The components of the architecture are introduced in section 8.1.1 and some of them are further described in subsequent sections. To avoid ambiguity of the terms “controller” and “PCP server”, the term “controller” will be used when referring to the SDN elements – forwarders and controllers – and the term “PCP server” when referring to the PCP elements – PCP client and PCP server.

OpenFlow is used as the standard to implement PCP over SDN. OpenFlow is widely supported

by vendors. Existing, freely available OpenFlow-compliant software for controllers and forwarders will be used (as described in section 5.2).

8.1.1. Components

PCP Client

A host located behind NAT runs user applications that try to establish communication with another host in an external network. Applications use the PCP client to receive or explicitly request mapping information from the PCP server.

Edge Forwarder

The edge forwarder is an OpenFlow switch that resides on the edge of the core network and the access network. The edge forwarder:

- forwards PCP messages between the PCP client and the PCP server,
- forwards other packets, according the configuration set by the controller, toward the external network through the NAT forwarder.

Controller

The SDN controller:

- manages forwarders (e.g. by adding, modifying or removing flow entries),
- runs network applications, including the PCP server and the NAT control plane.

PCP Server

The PCP server:

- processes PCP requests sent by the PCP client,
- instructs the NAT control plane to create a mapping for the PCP client,
- generates PCP responses to the PCP client.

NAT Control Plane

The NAT control plane contains the control logic of the NAT gateway. To better manage the complexity of the design, the NAT control plane is split into two subcomponents: *NAT handler* and *NAT table*.

NAT handler:

- processes requests from the PCP server to create new NAT table entries,
- removes NAT table entries whose lifetime expired on the NAT forwarder,
- instructs the controller to add or remove flow entries on the NAT forwarder corresponding to the NAT table entries.

NAT table stores NAT table entries. NAT table lets the NAT handler determine the available external IP addresses and ports that can be assigned to the applications. Each NAT table entry contains fields shown in Figure 8.2. *IP address family* can be IPv4 or IPv6. As per the requirements

specification (in chapter 7), only IPv4 address family is currently supported. *Protocol* refers to the protocol above the IP header – as per the specification, only TCP and UDP are currently supported.

IP address family	Protocol	Internal IP address	Internal port	External IP address	External port	Mapping lifetime
-------------------	----------	---------------------	---------------	---------------------	---------------	------------------

Figure 8.2: NAT table entry fields

NAT table entries are uniquely identified by two pairs – source IP address and source port. Each NAT table entry is added as a flow entry to the NAT forwarder.

NAT table also defines a pool of internal and external IP addresses and ports to be involved in translation.

NAT Forwarder

The NAT forwarder is an OpenFlow switch containing flow entries that perform NAT, i.e. rewrite IP addresses and transport protocol ports in incoming packets. These flow entries are managed by the NAT control plane using the NAT table. The NAT forwarder resides on the edge of the core network and the external network.

8.1.2. Network Application Components

Figure 8.3 shows the components of the network application in more detail. *NAT installer* is a part of the NAT handler that manages flow entries on the NAT forwarder. *PCP message processor* parses received PCP requests from the PCP client and builds PCP responses that the PCP server then sends to the PCP client. *PCP installer* installs flow entries on the edge forwarder that forward PCP messages.

Network application entry point, as its name suggests, is the main entry point for the application. This component receives PCP messages forwarded by the edge forwarder and control messages between the forwarders and the controller (*Packet-In* messages). One such type of control messages is the OpenFlow *Features* message, which are received when the presence of each forwarder is detected by the controller.

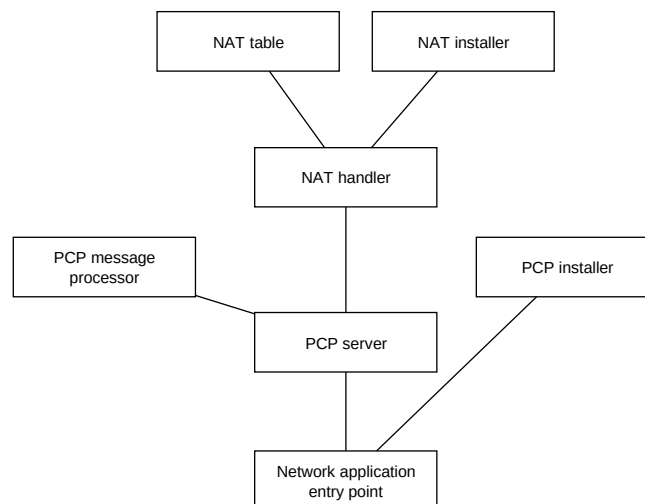


Figure 8.3: Network application components

8.2. PCP Client Mapping Request – Processing

Figure 8.4 describes the process of the PCP client determining its external IP address and port. To simplify the description of the process, it is assumed that the PCP client already knows the IP address of the PCP server. When the controller receives the PCP request, it passes the request to the PCP server.

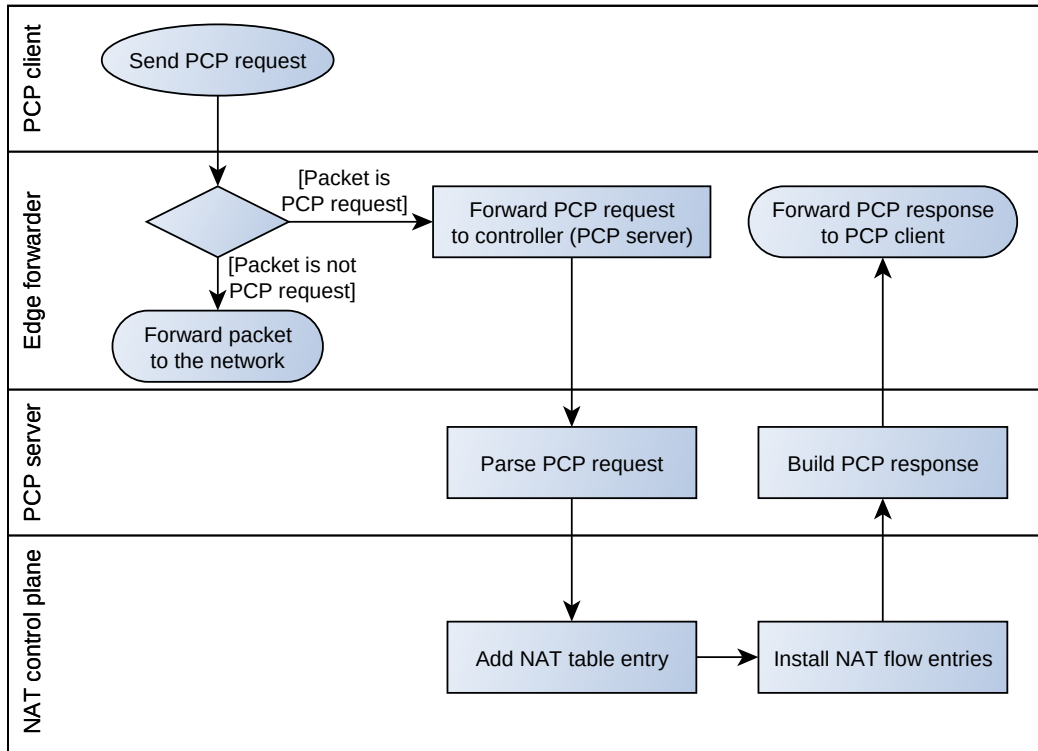


Figure 8.4: PCP request processing by the network application

Updating and uninstalling flow entries is handled similarly – instead of adding a new NAT table entry and installing new NAT flow entries, the existing ones are updated or removed. Processing the PCP request, including parsing the request and building the response closely follow the processes described and depicted in section 3.2.

8.3. Edge Forwarder

Flow entries on the edge forwarder, shown in Figure 8.5, detect PCP request and PCP response messages, respectively. *in_port* matches ingress ports, *pcp_server_address* is the IP address of the PCP server that the PCP client uses to send PCP requests, and port 5351 is the UDP port for PCP communication assigned by IANA²⁰.

The *Packet Forwarding* flow table is an abstraction that denotes additional packet processing (such as switching or routing). This allows the edge forwarder to be integrated to an existing SDN network.

When forwarding the PCP response, the *send to access network* action does not specify which

²⁰ <http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml>

access port is used as the egress port. This is implementation-dependent – it can be the same port as the port on which the PCP request was received, or it could be a different port in case the network implements additional network mechanisms such as load balancing.

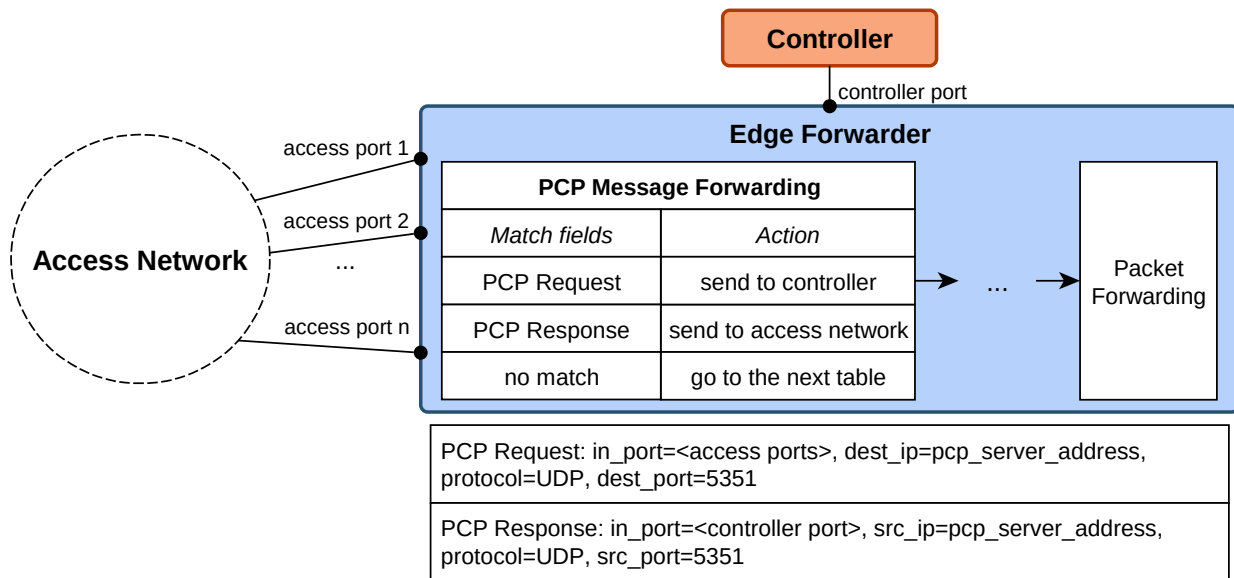


Figure 8.5: Edge forwarder flow entries

8.4. NAT Forwarder

NAT forwarder comprises the data plane of a NAT gateway. A detailed structure of the NAT forwarder is shown in Figure 8.6.

The first flow table determines whether the port is an external port. If so, the packet is subject to translation (flow table 3) – its destination IP address and port is translated to the corresponding internal IP address and port. For non-external ports (i.e. ports within the core network), the translation is reversed (flow table 2) – source IP address and port is translated to the corresponding external IP address and port. As on the edge forwarder, the *Packet Forwarding* table is an abstract flow table that handles additional packet processing.

To ensure that an idle connection is closed, each flow entry has a `idle_timeout` timer initially set to the mapping lifetime specified in the corresponding NAT table entry. If a flow entry expires, the NAT forwarder sends the *Flow-Removed* message to the controller so that the NAT control plane can remove the NAT table entry. This table entry can thus be reused later for another connection.

The mapping lifetime from the corresponding NAT table entry is directly tied to `idle_timeout` timeout on the flow entries. This way, the NAT entries on the forwarder, including their timeout values, are fully under the control of the PCP server.

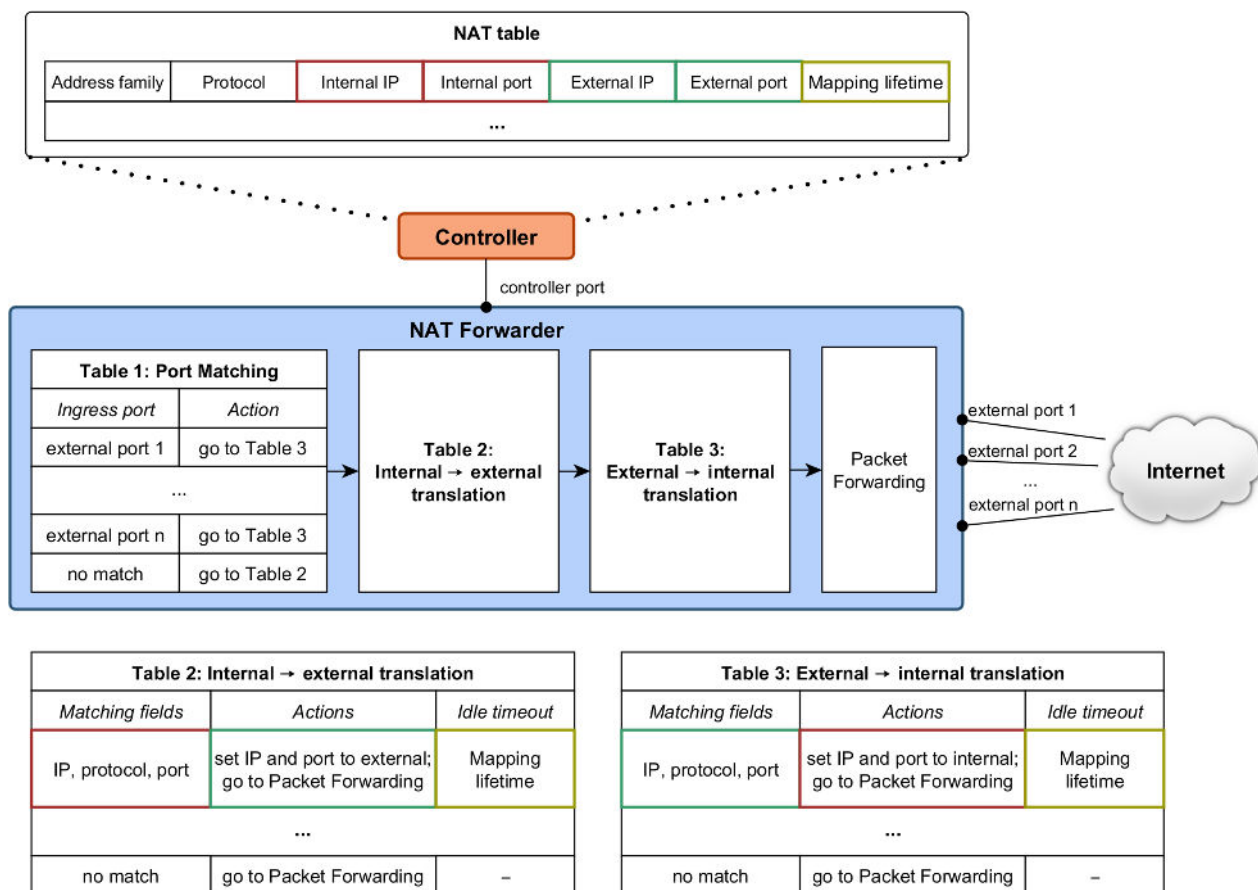


Figure 8.6: NAT forwarder flow tables and entries

9. Implementation

This chapter describes the implementation of the solution introduced in previous chapters. The chapter focuses only on the more important aspects of the implementation. The complete documentation for each module, class and method implemented in the software is embedded in the source code.

9.1. Implementation Environment

The following software is used for the implementation:

- controller: *ryu* framework²¹,
- forwarder: *ofsoftswitch* 1.3²²,
- PCP client library²³,
- *nmap* command suite²⁴, of which the `nping` command is used to test TCP/UDP communication between hosts.

The network application is implemented over the *ryu* framework in Python language and developed in *Eclipse IDE*²⁵ with *PyDev*²⁶ add-on.

In order to verify the functions of the network application, a test topology is created as shown in Figure 9.1. The test topology is created and run on a virtual machine with *Ubuntu Server*, 14.04.1, 64-bit. The virtual machine runs on *VirtualBox*²⁷. In the virtual machine, *Host 1* and *Host 2* are virtual hosts created by the `ip-netns`²⁸ command.

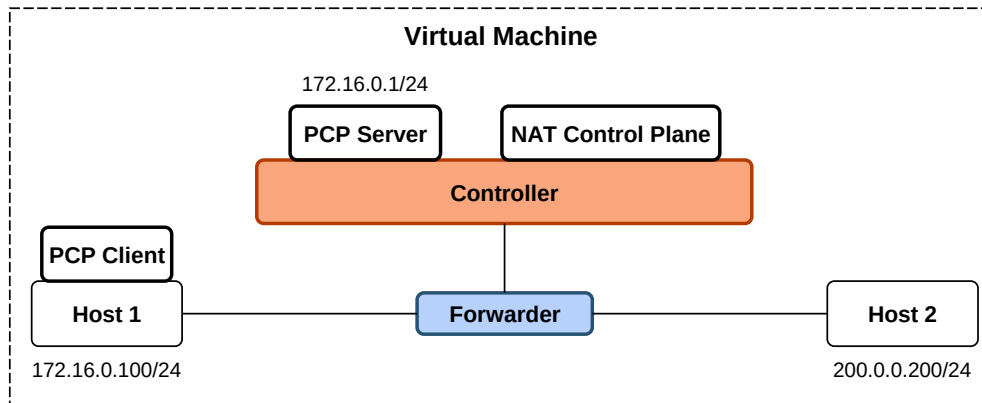


Figure 9.1: Test topology for verification

21 Available at: <https://github.com/osrg/ryu>

22 Available at: <https://github.com/CPqD/ofsoftswitch13>

23 Available at: <https://github.com/libpcp/pcp>

24 Available at: <https://nmap.org/>

25 More information at: <https://eclipse.org/>

26 More information at: <http://pydev.org/>

27 More information at: <https://www.virtualbox.org/>

28 More information at: <http://man7.org/linux/man-pages/man8/ip-netns.8.html>

To simplify the installation of the software required to create, run and test the network, user documentation and scripts from the *UnifyCore*²⁹ project are used.

Although the *PCP Testing Tool* has a more user-friendly interface, the PCP client library is faster to work with, especially when repeating the same command (by retrieving it from the command history in the shell environment).

9.2. Implementation Description

The architecture of the network solution, as shown in Figure 8.1, contains two separate forwarders – edge forwarder and NAT forwarder – each having their distinct roles. In the implementation, both forwarders were merged into one forwarder with multiple flow tables. This simplifies implementation and, consequently, its verification. Figure 9.2 shows the overview of the implementation on the only forwarder in the test network.

Flow entries for PCP message forwarding and NAT forwarding are managed separately to emphasize the fact that they are separate components, conceptually. This is achieved by managing the flow entries in two separate modules – NAT installer and PCP installer, as shown in Figure 8.3.

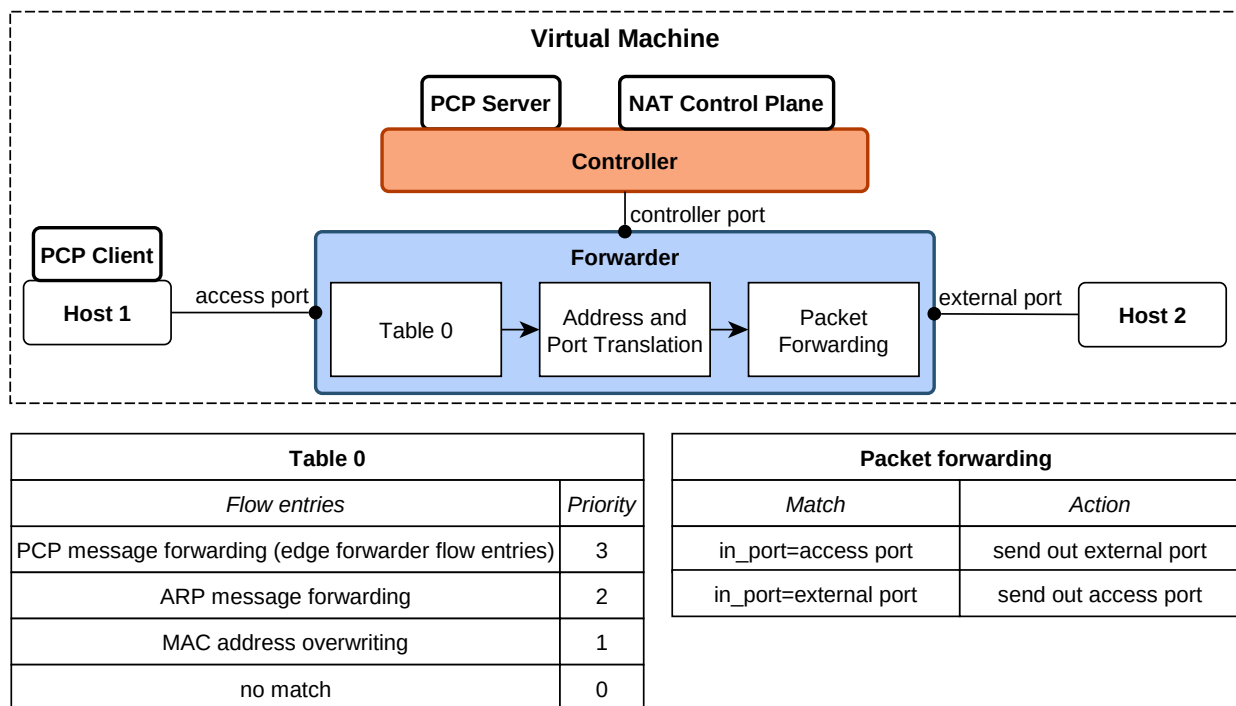


Figure 9.2: Forwarder implementation overview

Given that only one access port exists in the network, the PCP message forwarding behavior is greatly simplified – the forwarder sends the PCP response out the same port the corresponding PCP request was received on.

The NAT flow tables (represented as *Address and Port Translation* in Figure 9.2) do not contain a table-miss flow entry. That is, if a packet does not match any flow entry in the internal-to-external or external-to-internal NAT flow tables, the packet is not forwarded to the *Packet Forwarding* flow

²⁹ More information at: <http://www.unifycore.com/>

table and is consequently dropped. This effectively means that the packet cannot be forwarded if no matching NAT flow entries for that packets are installed, i.e. the internal host did not request a mapping from the PCP server.

9.2.1. ARP Message Processing

While the network design addresses the desired network behavior (message forwarding, IP address and port translation), it does not cope with one important feature of IP networks – destination MAC address resolution (ARP processing).

When a host tries to establish connection with another host, it must first know its MAC address if the second host is on the same local network, or the MAC address of the default gateway if the second host is in a different network. In order to determine the MAC address, the host sends an ARP request and expects an ARP reply with the correct MAC address.

Although the ARP message forwarding is seemingly handled automatically by network switches or routers (when considering traditional networks), this behavior is missing in pure SDN networks and must be implemented.

Two approaches to ARP processing were considered in the implementation – proxy ARP and routing – of which the former was eventually implemented.

Proxy ARP

This implementation forwards ARP requests to the controller, which handles the requests on behalf of the source host. The process is illustrated in Figure 9.3 where Host 1 requests the MAC address of Host 2.

The ARP handler is responsible for processing ARP messages and installing flow entries on the forwarder that overwrite MAC addresses. Additionally, the ARP handler stores ARP entries in order to associate the source MAC address with the destination host MAC address.

This implementation assumes that a host requests the MAC address of the destination host, even if the destination host is in another network. To simulate this scenario in the test topology, both hosts must have a default static route set:

```
route add -net '0.0.0.0' netmask '0.0.0.0'
```

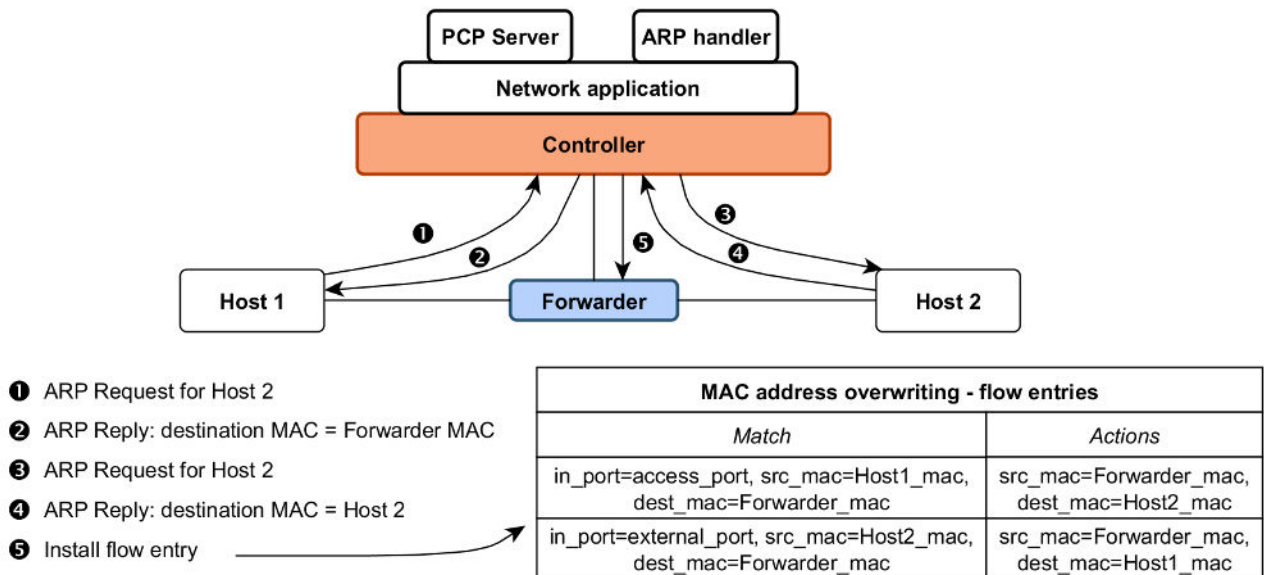



Figure 9.3: Example of a MAC overwriting flow entry installation with proxy ARP approach

The advantage of this approach is its simpler implementation and faster packet forwarding – there is no need to wait for the first data packet to trigger the ARP process for the destination host.

Without NAT, this approach works properly. Once NAT is introduced in the network, Host 1 to Host 2 communication also works properly. The problem arises when Host 2 first tries to communicate with Host 1. Host 2 requests the MAC address for the external IP address of Host 1. Although the controller can possibly translate the external IP address to the associated internal IP address, ports must also be considered. It may happen that for two distinct connections, Host 1's external IP addresses are different. In such case, the translation is ambiguous and thus cannot be performed. The second approach does not have this issue. Due to time constraints, the second approach was not implemented.

The second approach effectively implements IP routing in the network. The first host sends ARP request to its default gateway (in this case, the controller) and the gateway responds with its MAC address (forwarder MAC address). When the first host sends the first packet, the controller receives this packet and sends an ARP request to the external network, using the source IP address of the default gateway for the external network and the source MAC address of the forwarder. The destination host sends back ARP reply to the controller. The controller then installs corresponding flow entries that correctly translate MAC addresses of incoming packets.

9.2.2. NAT Table

Flow entries that translate IP addresses and ports have a higher priority set than the no-match flow entry in order to ensure that the NAT flow entries have a precedence.

The NAT table defines a NAT pool – range of internal and external IP addresses and ports involved in the translation. If the PCP client requests an external IP address or port that is already assigned or is outside the defined pool, the PCP server assigns a valid external IP address and port. External IP addresses and ports are both allocated using the round-robin algorithm.

The NAT table is represented by a dictionary (hash table). The key to each table entry is defined by a concatenation of the internal IP address and internal port.

9.2.3. Managing Mapping Lifetime

PCP server is responsible for determining the mapping lifetime. Assuming that the host is a mobile device connected to a WCDMA network, the lifetime is expected to be high enough to reduce the battery power consumption and the amount of signaling traffic to an acceptable minimum. Further assuming that mappings created by PCP MAP and PCP PEER requests exhibit different traffic patterns (see section 3.2.1 for more information), two different lifetime values need to be defined for PCP MAP and PCP PEER mappings. Acceptable lifetime values for WCDMA networks are determined in chapter 10.

9.3. Verification

This section describes how to verify the implementation. The verification comprises the following:

- Verify that a user application running a PCP client can request mapping information from the PCP server.
- Verify that, with the mapping information acquired by the PCP client, two hosts can communicate with each other, with the first host being behind a NAT.

Figure 9.1 shows the test topology for verification. Assuming the topology is set and the network application is running, the verification can be performed in the following steps:

1. Host 1 generates a PCP request using the `pcp` command from the PCP client library. Currently, the implemented forwarder does not consider what IP address the PCP client specifies as the PCP server address, so the PCP client may choose an arbitrary address. PCP server running on the controller parses the PCP request, instructs the NAT handler to create a table entry and install flow entries on the forwarder, and finally sends back a PCP response to the PCP client. The `pcp` command now displays the mapping returned by the PCP server.
2. Host 2 runs the `nping` command as a server and listens to incoming TCP connections.
3. Host 1 runs the `nping` command as a client and sends test TCP segments to Host 2. Host 2 replies back to Host 1.

If the communication is successful, the implementation functions properly. To verify that the addresses are translated properly, one can capture traces from both hosts to separate *pcap* files using the `tcpdump` command and determine that the IP addresses for the TCP segments are translated properly.

Appendix B contains a user guide that replicates the above steps in detail.

To verify the communication in the topology without PCP, there is no need for the implemented SDN network solution. The verification can be performed with the use of the `iptables` program, which includes rules for address and port translation³⁰.

³⁰ Examples of NAT configuration using `iptables` can be found at:
http://www.karlsruher.net/en/computer/nat_tutorial

10. Evaluation

This chapter evaluates how much PCP can reduce battery power consumption of mobile devices and the amount of signaling traffic in WCDMA networks.

As discussed in chapter 4, sending frequent keepalives causes increased battery power drain and generates a considerable number of signaling messages. If the keepalive interval is increased, the battery of the mobile device consumes less power and the network generates fewer signaling messages over time.

In order to ensure that mobile devices can send keepalives in increased intervals while still maintaining mapping entries on middleboxes, the mobile devices can use PCP to determine the mapping lifetime and consequently optimize their keepalive interval.

The goal of the evaluation is to:

- determine how much battery life and signaling traffic can be saved by increasing the keepalive interval,
- determine the acceptable mapping lifetime that PCP server should assign to the PCP client based on the computed keepalive interval.

The evaluation is subject to the following restrictions:

- Only one always-on user application runs on the mobile device.
- With PCP enabled, the overhead of sending a PCP request and receiving a PCP response prior to establishing a connection is ignored, as the connection itself is assumed to be established immediately afterwards, during which several more messages are exchanged. Given that these messages do not correspond to the traffic pattern of sending a single keepalive, their presence is ignored to avoid distorting the results.
- Only keepalives, PCP messages and the signaling messages associated with them are sent over the network.
- Any signaling unrelated to sending keepalives is ignored (such as signaling associated with handovers).
- If CELL_FACH is enabled in the network, PCP messages are considered small enough for the device to transition only to the CELL_FACH state instead of the CELL_DCH state. In this case, PCP messages behave as keepalives, such as in case of PCP MAP mappings (see section 3.2.1 for more information), hence their presence cannot be ignored.
- When the device sends keepalives with acknowledgments (such as TCP keepalives) or PCP requests followed by PCP responses, the acknowledgments/responses are received after a delay, causing the RRC inactivity timer to be refreshed. For the sake of simplicity, the round-trip time is assumed to be small enough that the additional time the mobile device stays in its RRC state can be neglected. This applies to the evaluation of the battery power consumption.
- Any other sources of battery power consumption are not considered. One consequence of

this restriction the fact that the display of the mobile device must be turned off.

The rest of this chapter is structured as follows. Section 10.1 determines the impact of keepalives on the battery power consumption of mobile devices. Section 10.2 determines the impact of keepalives on the number of signaling messages generated in the network. Section 10.3 determines the acceptable lifetime values for PCP-aware mappings given the results from the previous sections.

10.1. Battery Life Extension

Battery charge and power consumption can be measured by several methods, such as by voltage levels or by ampere-hours³¹ (or, more commonly, milliampere-hours, mAh). The latter is used in this section.

10.1.1. Battery Power Consumption Figures

Source [5] specifies measured values for battery power consumption. One set of values (Table 2 from source [5]) was measured with varying keepalive intervals, with $T2 = 2$ s and with CELL_PCH power state enabled in the network. A different set of values (Table 3 from source [5]) was measured in a different mobile network (implying different network conditions, signaling, etc.) with varying T2 timers and with the keepalive interval of 40 s. Table 10.1 contains the results of the measurements for a single keepalive from source [5].

Table 10.1: Measured battery power consumption of keepalives in 3G WCDMA networks [5]

Timer T2 [s]	Average current in CELL_FACH state [mA]	Cost of a single keepalive in 3G [mAh]
2	120	0.15 – 0.6

10.1.2. Formulas

In order to quantify the battery power consumption reduction, reference values must be defined. For example, suppose that an application currently uses a keepalive interval of 20 seconds (such as IPsec ESP [11]). If the keepalive interval is increased, the mobile device consumes that much less battery power compared to the original (reference) keepalive interval. Furthermore, a fixed time period must be defined over which keepalives are sent.

Given the criteria above, the following parameters can be defined:

- T – time period over which the keepalives are sent
- t_{ref} – original (reference) keepalive interval without PCP
- t_{new} – new keepalive interval with PCP
- $cost$ – cost of a single keepalive (in mAh)

The number of keepalives n sent over time period T given the interval t_{new} can be determined as follows:

³¹ http://www.otherpower.com/otherpower_battery_metering.html

$$n = T \cdot \frac{1}{t_{new}} \quad (1)$$

where $\frac{1}{t_{new}}$ is the number of keepalives sent per second.

The amount of battery consumption saved (in mAh) can be determined as follows:

$$reduction(t_{new}) = k_{ref} - k = (n_{ref} - n) \cdot cost = (T \cdot \frac{1}{t_{ref}} - T \cdot \frac{1}{t_{new}}) \cdot cost = cost \cdot T \left(\frac{1}{t_{ref}} - \frac{1}{t_{new}} \right) \quad (2)$$

where k is the total cost of keepalives over time period T , i.e. the number of keepalives sent multiplied by the cost of a single keepalive:

$$k = n \cdot cost \quad (3)$$

In order to determine the relative amount of battery power saved given the desired and reference keepalive intervals (t_{new} and t_{ref} , respectively), the battery capacity C of the mobile device (in mAh) must be known.

The relative amount of battery used when sending keepalives can be determined as follows:

$$\frac{k}{C} \quad (4)$$

In other words, if the relative amount is multiplied by 100%, the keepalives consume $\frac{k}{C} \cdot 100\%$ of the battery over time T .

The relative reduction of the battery power consumption (i.e. the battery power saved) given the battery capacity C can then be determined as follows:

$$battery\ power\ saved = \frac{k_{ref}}{C} - \frac{k}{C} = \frac{k_{ref} - k}{C} = \frac{reduction(t_{new})}{C} \quad (5)$$

From the last formula, one can conclude that, by using a higher keepalive interval t_{new} , such percentage of battery consumption was saved over time T .

From the end-user perspective, an alternative measure may better indicate the power consumption reduction – how much longer the battery will last before recharging it. Suppose that the following values are known:

- $cost$ – cost of a single keepalive,
- $\bar{I}_{keepalive}$ – average current while sending a single keepalive.

From these two values, one can compute the amount of time the battery life is shortened by sending a single keepalive:

$$\tau = \frac{cost}{\bar{I}_{keepalive}} \quad (6)$$

The total time of the battery life saved can then be computed given the cost of a single keepalive, the measuring time period T , the reference keepalive interval t_{ref} and the new keepalive interval t_{new} :

$$battery\ lifetime\ saved = (n_{ref} - n) \cdot \tau = (n_{ref} - n) \cdot \frac{cost}{\bar{I}_{keepalive}} = \frac{reduction(t_{new})}{\bar{I}_{keepalive}} \quad (7)$$

As seen from the formula, the battery lifetime saved does not depend on the battery capacity.

10.1.3. Results

Figures 10.1 and 10.2 show the battery power saved with increasing keepalive interval, given the time period, cost of a single keepalive, battery capacity and several values of the reference keepalive interval (t_{ref} in the figures).

Table 10.2 contains the battery power saved given chosen values for the battery capacity. Typical smartphones can thus save 1-4% of the battery life, while smart watches with 3G capabilities can significantly benefit from the keepalive reduction by saving as much as 34.2% of the battery life. The increase in the keepalive interval proves to be much less substantial for devices with relatively high battery capacity, such as tablets (0.35%–1.4%).

Table 10.2: Amount of battery power consumption saved of a mobile device connected a WCDMA network given battery capacity and reference values

Reference values: $t_{ref} = 20$ s, $t_{new} = 400$ s, $T = 3600$ s, $cost$: 0.15-0.6 mAh

Battery capacity	Battery power saved
300 mAh (<i>Samsung Gear S smart watch</i> ³²)	8.5-34.2%
2550 mAh (<i>Samsung Galaxy S6 phone</i> ³³)	1-4%
7340 mAh (<i>iPad Air 2 tablet</i> ³⁴)	0.35-1.4%

As seen in Figures 10.3 and 10.4, approx. 13-52 minutes of battery life can be saved for the cost ranging from 0.15 to 0.6 mAh, the average current of 120 mA and the reference interval of 20 seconds.

The percentage of the battery power and the battery lifetime saved increase significantly when the keepalive interval is increased by the first few tens of seconds from the reference interval. Above 400-600 seconds, the difference in the increase starts to be negligible.

32 <https://www.samsung.com/uk/consumer/mobile-devices/wearables/gear/SM-R7500ZKABTU>

33 <http://arstechnica.com/gadgets/2015/04/samsung-galaxy-s6-review-its-whats-on-the-outside-that-counts/>

34 <http://arstechnica.com/apple/2014/10/the-ipad-air-2-a-host-of-hidden-upgrades-in-one-skinny-package/>

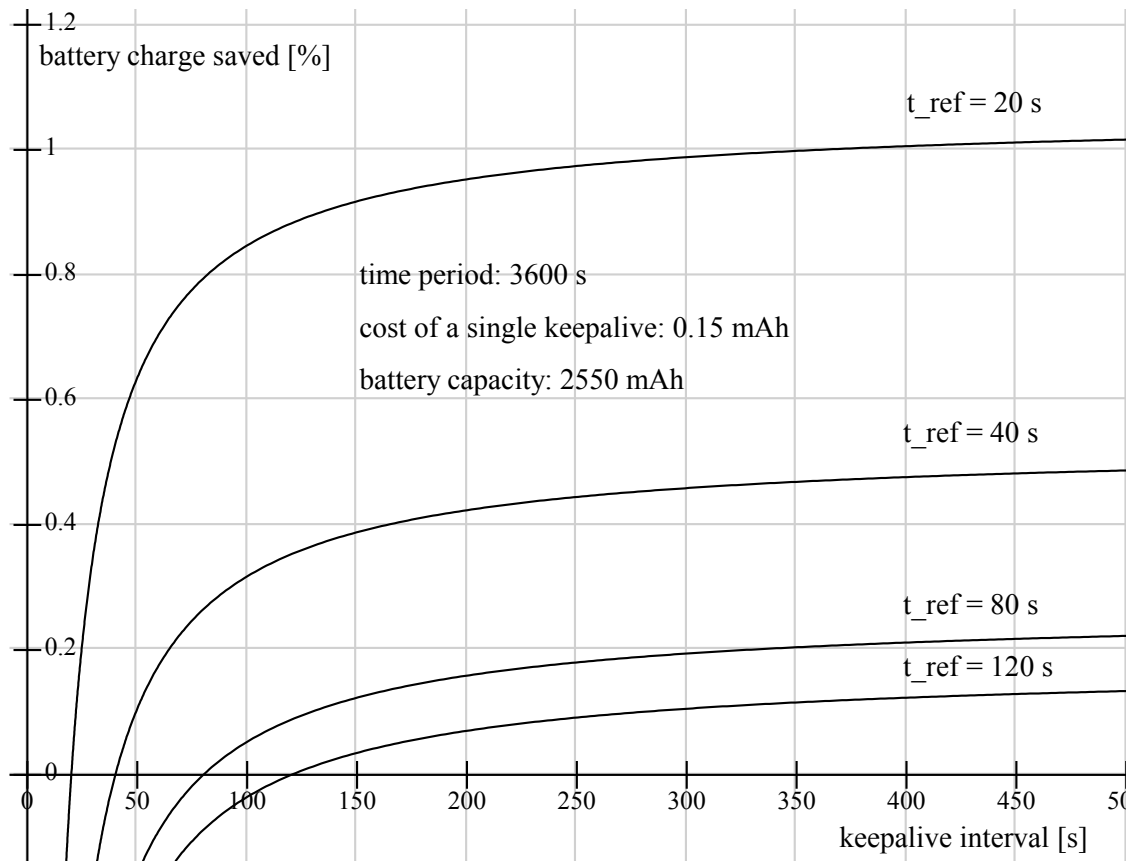


Figure 10.1: Amount of battery power saved based on keepalive intervals relative to reference values and cost of 0.15 mAh per keepalive

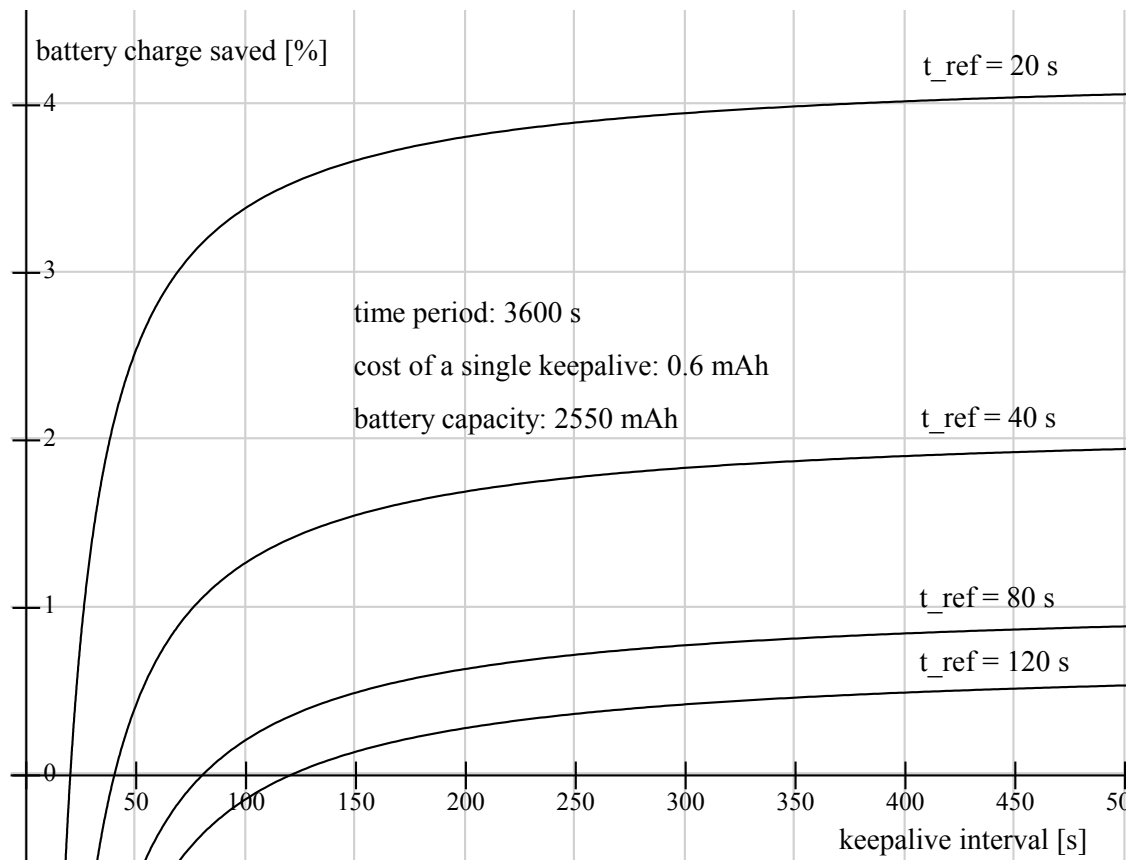


Figure 10.2: Amount of battery power saved based on keepalive intervals relative to reference values and cost of 0.6 mAh per keepalive

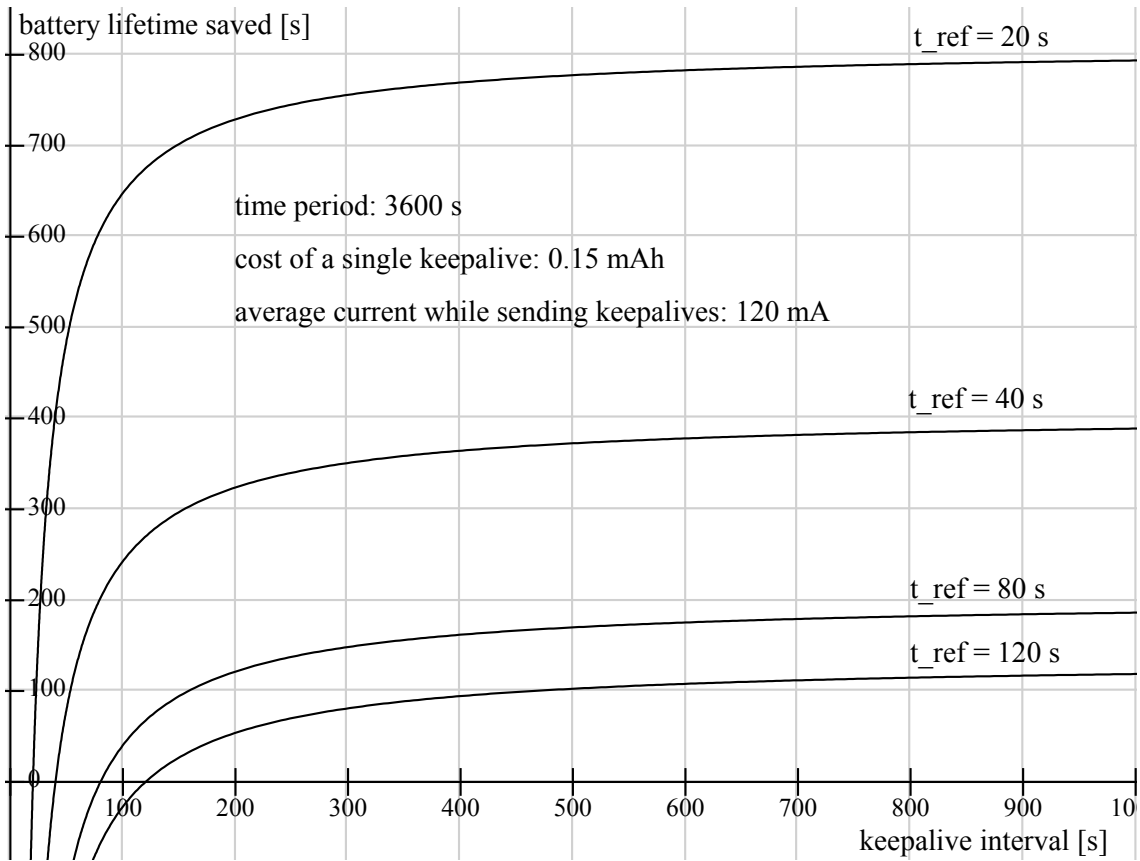


Figure 10.3: Amount of battery lifetime saved based on keepalive intervals relative to reference values and cost of 0.15 mAh per keepalive

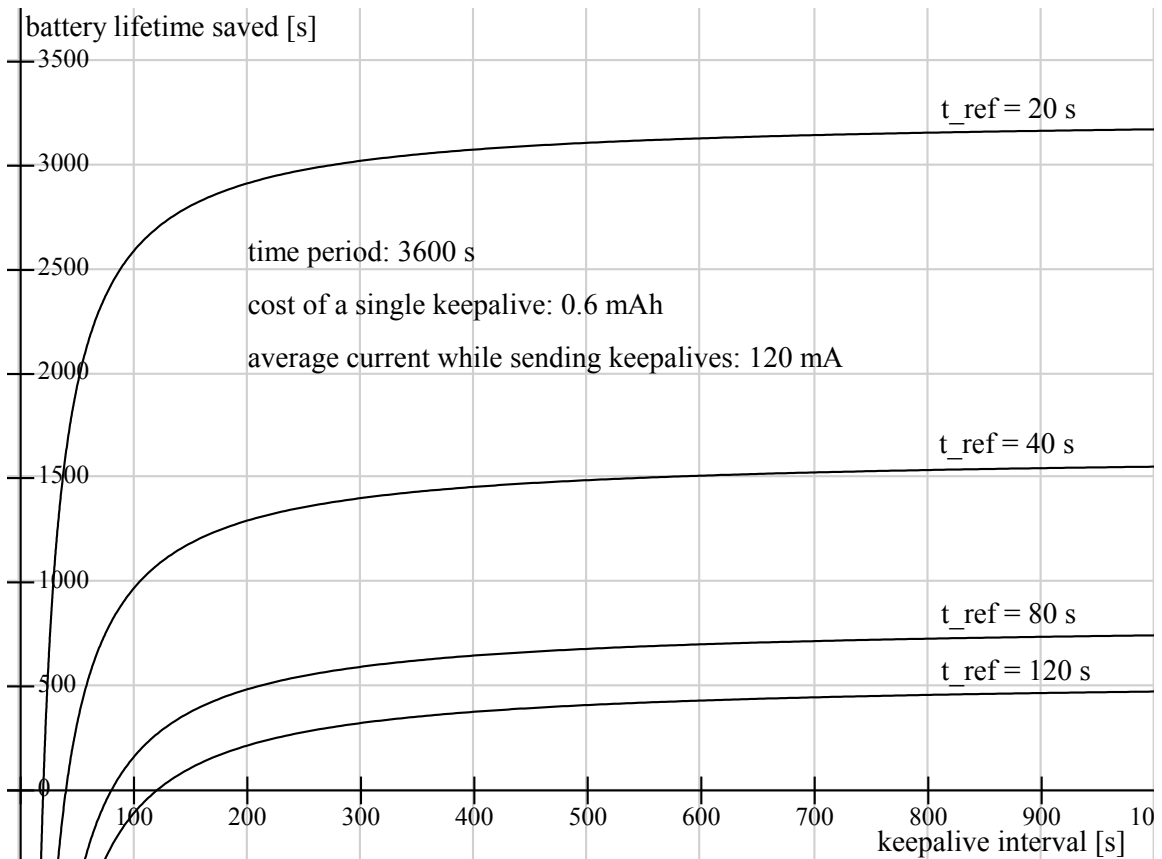


Figure 10.4: Amount of battery lifetime saved based on keepalive intervals relative to reference values and cost of 0.6 mAh per keepalive

10.2. Signaling Traffic Reduction

10.2.1. Network Traffic in WCDMA Networks

Source [30] contains measurements performed in two 3G WCDMA networks, observing the number of signaling messages generated in the networks and the battery power consumption in mobile devices.

The data and signaling traffic was captured on mobile devices. A certain amount of signaling traffic was generated in the core network that cannot be observed on mobile devices (referred to as “unobserved” signaling traffic in source [30]).

In one of the measurements, the mobile devices sent keepalive messages to the network. In the observed networks, the mobile devices entered the CELL_DCH state when sending a keepalive.

According to the results, sending one keepalive causes 40-50 observed signaling messages to be exchanged between a mobile device and the network, and estimated 20 unobserved signaling messages [30].

10.2.2. Formulas

To determine the reduction of the number of signaling messages, the following parameters need to be defined:

- T – time period over which the keepalives are sent
- t_{ref} – original (reference) keepalive interval without PCP
- t_{new} – new keepalive interval with PCP
- $cost$ – cost of a single keepalive (in mAh)
- s – number of signaling messages per a single keepalive
- S – total number of signaling messages sent over T given keepalive interval t_{new}

Using the formulas defined in section 10.1, S can be computed as

$$S = n \cdot s \quad (8)$$

The number of signaling messages reduced in the network can then be computed as:

$$reduction(t_{new}) = S_{ref} - S = (n_{ref} - n) \cdot s = \left(\frac{1}{t_{ref}} - \frac{1}{t_{new}} \right) \cdot s \cdot T \quad (9)$$

10.2.3. Results

As seen in Figures 10.5, 10.6 and 10.7, the reduction of the number of signaling messages grows rapidly up to the keepalive interval of approx. 400 seconds. The growth of the reduction starts to be negligible from approx. 1800 seconds, which can be considered an acceptable keepalive interval for WCDMA networks. Table 10.3 quantifies the results for reference keepalive intervals of 20 s and 120 s.

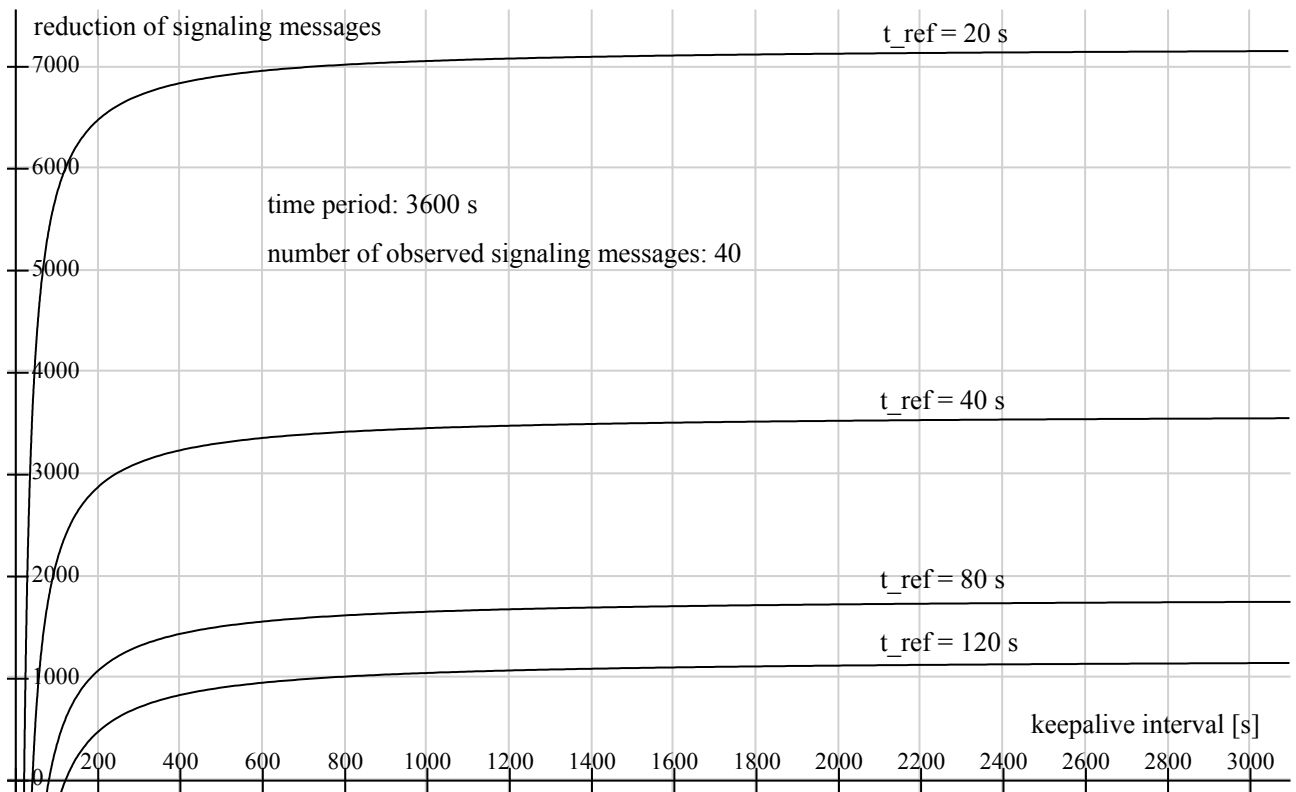


Figure 10.5: Number of signaling messages reduced based on keepalive intervals and reference values (40 observed messages)

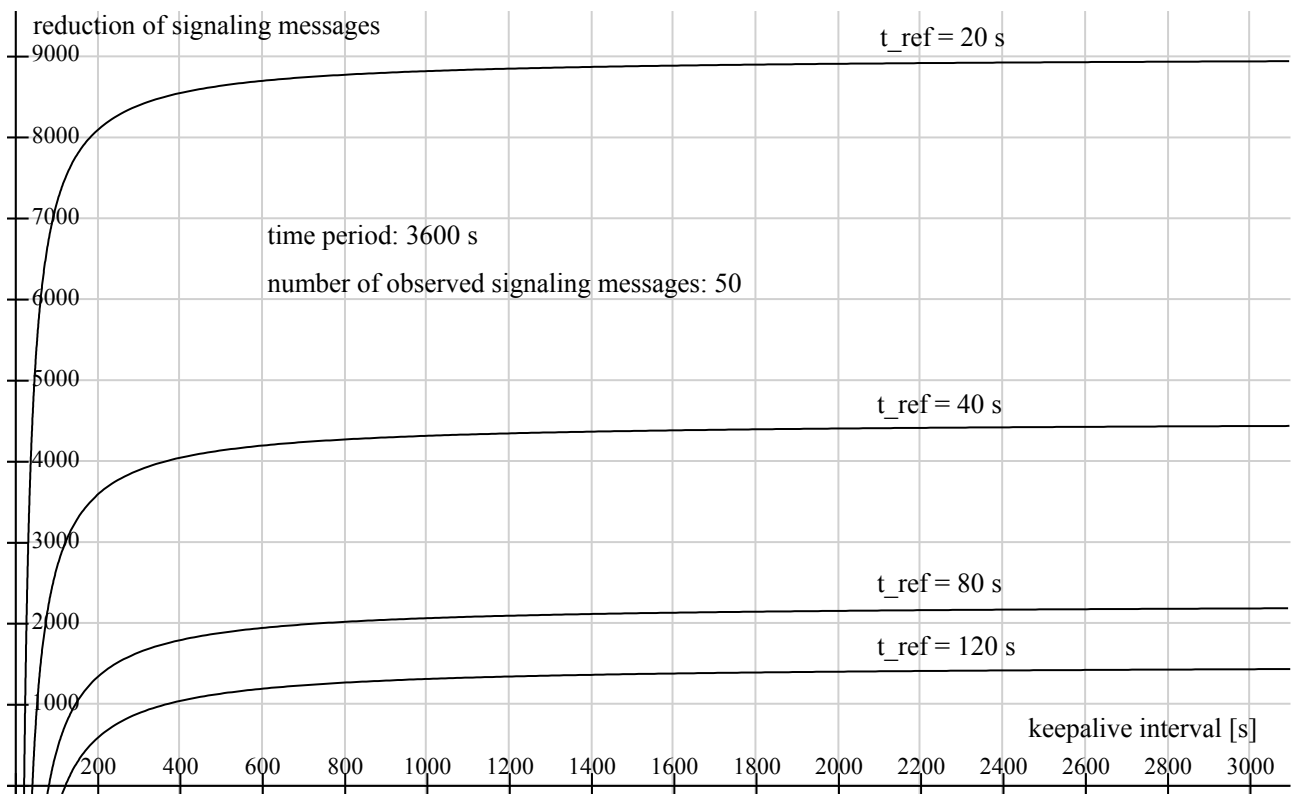


Figure 10.6: Number of signaling messages reduced based on keepalive intervals and reference values (50 observed messages)

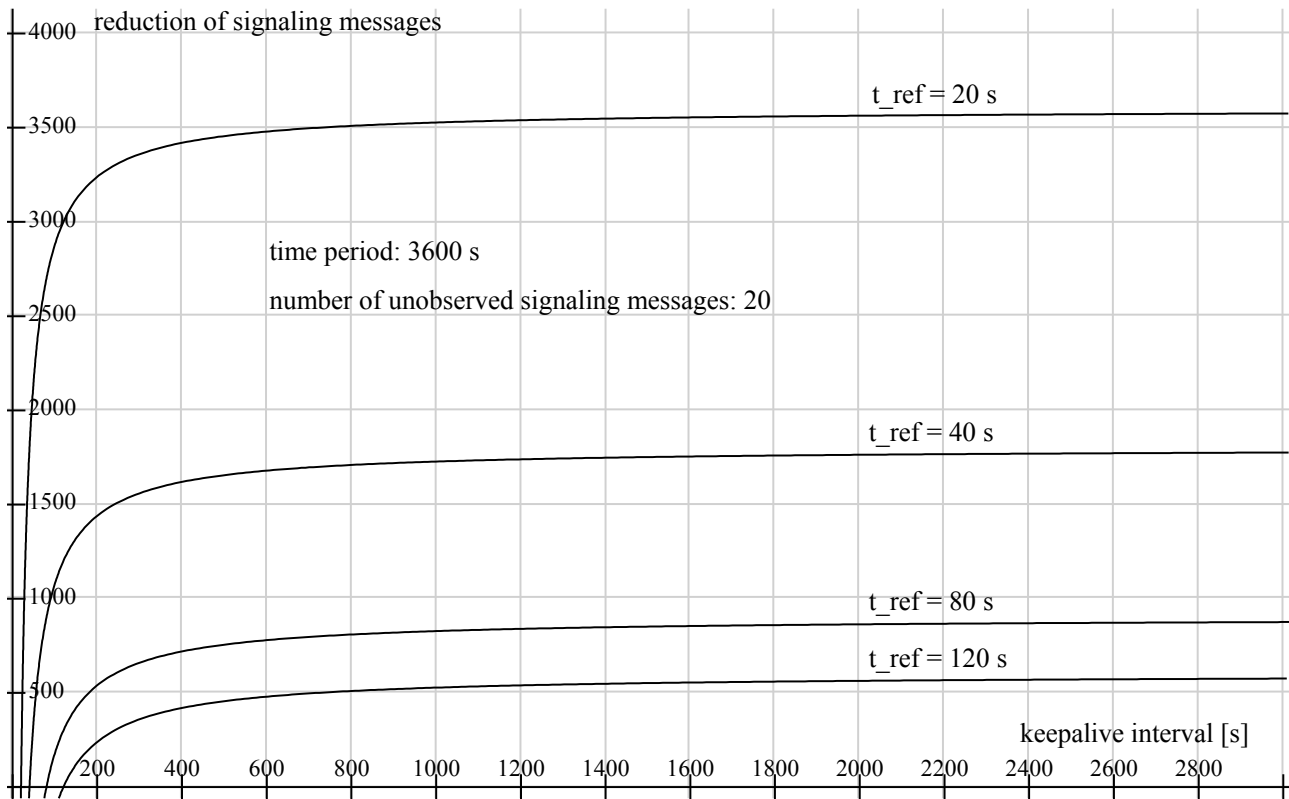


Figure 10.7: Number of signaling messages reduced based on keepalive intervals and reference values (20 unobserved messages)

Table 10.3: Reduction of the number of signaling messages given the number of messages per keepalive and reference values

Reference values: $t_{new} = 1800$ s, $T = 3600$ s

Number of signaling messages per keepalive	Reference keepalive interval	Reduction of signaling messages	Reference keepalive interval	Reduction of signaling messages
40 (observed)	20 s	7120	120 s	1120
50 (observed)	20 s	8900	120 s	1400
20 (unobserved)	20 s	3560	120 s	560

It should be noted that the reduction of the number of signaling messages was computed for one mobile device running a single application. Considering that hundreds of thousands of mobile devices are connected to a network, each running one or more always-on applications, the decrease in the network load on elements in the network core may prove to be significant.

10.3. Conclusions

From the perspective of a mobile device and its battery life, the keepalive interval of 400–600 seconds is suitable for most applications. The higher the keepalive interval, the smaller the amount of battery power is saved. When considering the amount of signaling traffic generated in a mobile network, the keepalive interval of approx. 1800 seconds is sufficient to greatly reduce the signaling traffic caused by sending keepalives over the network.

The amount of battery power saved by using higher keepalive intervals is more significant in devices with relatively small battery capacity, such as smartphones and smart watches. Tablets with greater battery capacity benefit little from the increase keepalive interval. By observing the amount of battery lifetime saved instead (which does not depend on the battery capacity), the extra minutes in stand-by mode can prove helpful for the user before the battery charge is depleted.

Higher keepalive intervals further reduce the battery consumption, but too high values may fail to determine connections that died between sending two consecutive keepalives. With higher keepalive intervals, middleboxes keep the mapping entries longer, which may result in reaching their memory limits, causing in turn to prematurely remove mapping entries.

10.3.1. Determining PCP Mapping Lifetime

In order to ensure that the applications will increase their keepalive intervals, they need to utilize PCP to request or receive mapping lifetime.

For PCP mappings created by PCP PEER requests (i.e. user applications function as clients), the mapping lifetime should be somewhat higher than the keepalive interval in order to allow the application to send keepalives early enough. Given the relatively high keepalive interval of 1800 seconds, it may be sufficient for the application to send the keepalives $7/8$ of the mapping lifetime. Therefore, the mapping lifetime for PCP PEER mappings could be approx. 2060 seconds.

For PCP MAP mappings (i.e. user applications functioning as servers), user applications need to send PCP MAP messages at the interval of at least $1/2$ of the mapping lifetime. In order to sustain the interval of 1800 seconds, the mapping lifetime for PCP MAP mappings should be at least 3600 seconds. Beside PCP MAP requests, applications may still have to send keepalives to the destination host to maintain the end-to-end connection.

Given the determined lifetime values above, the following recommendations apply to WCDMA networks with PCP and the custom solution described in this thesis:

- For mappings created by PCP MAP requests, send PCP MAP requests and keepalives at the same time to avoid frequent RRC state transitions.
- For mappings created by PCP PEER requests, send only keepalives to the remote host and do not send any PCP PEER requests to the PCP server. Given the design of the solution described in this thesis, the PCP server has complete control over the middleboxes, including the timeouts of the flow entries, hence the PCP client does not have to be concerned about any special behavior of the middlebox.

It is expected that, given the widespread use of always-on applications such as social networks, hosts establish outbound connections more commonly than hosting servers (i.e. establish inbound connections). Hence, PCP messages with the PEER opcode are expected to be more common than PCP MAP messages, which is more beneficial for the network, as PCP PEER mappings may send fewer messages in total than PCP MAP mappings.

Summary

The goal of the diploma thesis was to implement Port Control Protocol (PCP) in software defined networks (SDN), specifically the PCP server, and to quantify how much battery life of mobile devices can be saved and how much signaling traffic can be reduced if PCP is deployed in mobile networks. To verify the implementation of the PCP server in SDN networks, NAT was implemented in the network.

PCP can be used by a host to receive external-to-internal address and port mapping information from middleboxes (firewall, NAT gateway), including the mapping lifetime. Once the host knows about the lifetime of its mapping, it is able to optimize the interval of sending keepalives to the destination host.

The potential of keepalive optimization proved to be significant in mobile networks, specifically the WCDMA networks. Given the measured values from sources [5] and [30], a suitable keepalive interval for WCDMA networks proves to be 1800 seconds. Given the keepalive interval of 400-600 seconds, mobile devices running an always-on application can save roughly up to 52 minutes of battery life per hour when the devices are in stand-by mode and the application is only sending keepalives to the network. Moreover, increasing the keepalive interval to 1800 seconds can reduce the number of signaling messages by thousands in one hour, only when considering one mobile device running one always-on application.

SDN, while flexible, requires effort to implement basic packet processing, such as ARP message processing, packet switching or routing. One may not realize this fact until the phase of the network application implementation. In traditional networks, these functions are provided automatically by switches and routers.

Conclusion

This diploma thesis is concerned with the implementation of the Port Control Protocol (PCP) over software defined networks (SDN). PCP allows end-user applications, such as instant messaging or VoIP, to receive mapping information directly from middleboxes (such as a NAT gateway) in order to properly traverse through the middleboxes without workarounds and to reduce application keepalives.

The SDN network solution separates the control plane and the data plane of a middlebox with a PCP server to an OpenFlow controller and forwarders, respectively, which allows for greater flexibility and vendor device compatibility of the network and avoids the need to install PCP server in each middlebox.

In today's networks, especially considering the mobile networks, the keepalive intervals may still be largely unoptimized. Reducing application keepalives can be considered a software-based method to reduce network traffic overhead and to reduce battery power consumption of mobile devices. Increasing the keepalive interval to as high as 1800 seconds can considerably reduce the amount of signaling traffic, especially considering that millions of mobile devices may be connected to the network at the same time.

The diploma thesis can be further expanded upon in the future by addressing the related topics described below.

The solution for the NAT functionality designed and implemented in this thesis can be further expanded upon by providing support for IPv6, more upper-layer protocols, such as ICMP, IPsec ESP, SCTP or DCCP, or the automatic deletion of TCP sessions by tracking the TCP RST and FIN segments.

NAT functionality in the architecture can be abstracted away – i.e. a generic interface for middleboxes can be defined so that other types of middleboxes, such as firewalls, can be implemented.

Further improvements in the architecture can be made by addressing the scalability of PCP in SDN networks, adding support for multihoming or addressing the security of PCP.

Finally, the keepalive reduction can be evaluated in different types of widely used mobile networks, such as GPRS or LTE.

References

- [1] BRIM, S.W., CARPENTER, B.E.: *Middleboxes: Taxonomy and Issues*. RFC 3234. 2002.
- [2] Anatomy: A Look Inside Network Address Translators - The Internet Protocol Journal - Volume 7, Number 3. Cisco [online]. [Accessed 11 December 2014]. Available from: http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_7-3/anatomy.html.
- [3] KROCHMAL, M., CHESHIRE, S.: *NAT Port Mapping Protocol (NAT-PMP)*. RFC 6886. 2013.
- [4] Understanding Carrier Grade NAT. *Network World* [online]. 4 September 2009. [Accessed 11 December 2014]. Available from: <http://www.networkworld.com/article/2237054/cisco-subnet/understanding-carrier-grade-nat.html>.
- [5] HAVERINEN, H. et al.: Energy Consumption of Always-On Applications in WCDMA Networks. In: *Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th*. April 2007. p. 964–968.
- [6] BRADEN, R.: *Requirements for Internet Hosts - Communication Layers*. RFC 1122. 1989.
- [7] TCP keepalive overview. [online]. [Accessed 14 March 2015]. Available from: <http://tldp.org/HOWTO/TCP-Keepalive-HOWTO/overview.html>.
- [8] POSTEL, J.: *Transmission Control Protocol*. RFC 793. 1981.
- [9] JENNINGS, C., AUDET, F.: *Network Address Translation (NAT) Behavioral Requirements for Unicast UDP*. RFC 4787. 2007.
- [10] SRISURESH, P. et al.: *NAT Behavioral Requirements for TCP*. RFC 5382. 2008.
- [11] HUTTUNEN, A. et al.: *UDP Encapsulation of IPsec ESP Packets*. RFC 3948. 2005.
- [12] WING, D. et al.: Session Traversal Utilities for NAT (STUN). [online]. October 2008. [Accessed 11 December 2014]. Available from: <http://tools.ietf.org/html/rfc5389>.
- [13] MATTHEWS, P. et al.: *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)*. RFC 5766. 2010.
- [14] ROSENBERG, J.: *Interactive Connectivity Establishment (ICE): A Methodology for Network Address Translator (NAT) Traversal for Offer/Answer Protocols*. RFC 5245. 2010.
- [15] SRISURESH, P. et al.: *Middlebox communication architecture and framework*. RFC 3303. 2002.
- [16] QUITTEK, J. et al.: *NEC's Simple Middlebox Configuration (SIMCO) Protocol Version 3.0*. RFC 4540. 2006.
- [17] TSCHOFENIG, H. et al.: *A Survey of Protocols to Control Network Address Translators and Firewalls*. Internet-Draft draft-eggert-middlebox-control-survey-01. 2007.
- [18] BOUCADAIR, M. et al.: *Port Control Protocol (PCP)*. RFC 6887. 2013.
- [19] Port Control Protocol - The Internet Protocol Journal, Volume 14, No. 4. Cisco [online]. [Accessed 16 October 2013]. Available from: http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_14-4/144_pcp.html.

- [20] WING, D.: Port Control Protocol. *The Internet Protocol Journal* [online]. Vol. 14. [Accessed 29 November 2013]. Available from: http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_14-4/ipj_14-4.pdf.
- [21] Port Control Protocol Overview - Technical Documentation - Support - Juniper Networks. [online]. [Accessed 22 October 2013]. Available from: http://www.juniper.net/techpubs/en_US/junos/topics/concept/nat-port-control-protocol.html.
- [22] REDDY, T. et al.: *Port Control Protocol (PCP) Server Selection*. RFC 7488. 2015.
- [23] BOUCADAIR, M. et al.: *DHCP Options for the Port Control Protocol (PCP)*. RFC 7291. 2014.
- [24] PENNO, R. et al.: *Port Control Protocol (PCP) Anycast Addresses*. Internet-Draft draft-ietf-pcp-anycast-04. 2015.
- [25] REDDY, T. et al.: *PCP Firewall Control in Managed Networks*. Internet-Draft draft-reddy-pcp-sdn-firewall-00. 2014.
- [26] SAUTER, M.: *From GSM to LTE-Advanced: An Introduction to Mobile Networks and Mobile Broadband*. Revised Second Edition edition. Chichester, West Sussex, United Kingdom: Wiley, 2014. ISBN 9781118861950.
- [27] 3RD GENERATION PARTNERSHIP PROJECT (3GPP): *Universal Mobile Telecommunications System (UMTS); Radio Resource Control (RRC); Protocol Specification*. 3GPP Technical Specification 25.331. 2015.
- [28] HOLMA, H., TOSKALA, A.: *HSDPA/HSUPA for UMTS: High Speed Radio Access for Mobile Communications* [online]. John Wiley & Sons, 2006. [Accessed 7 April 2015]. ISBN 978-0-470-01884-2. Available from: <http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470018844.html>.
- [29] PERALA, P.H.J. et al.: Theory and Practice of RRC State Transitions in UMTS Networks. In: *2009 IEEE GLOBECOM Workshops*. November 2009. p. 1–6.
- [30] SIGNALS RESEARCH GROUP, LCC: *Smartphones and a 3G network, reducing the impact of smartphone-generated signaling traffic while increasing the battery life of the phone through the use of network optimization techniques*. 2010.
- [31] HOLMA, H., TOSKALA, A.: *WCDMA for UMTS: HSPA Evolution and LTE*. 5 edition. Hoboken: Wiley, 2010. ISBN 9780470686461.
- [32] NADEAU, T.D., GRAY, K.: *SDN: Software Defined Networks*. O'Reilly Media, 2013. ISBN 978-1-4493-4230-2.
- [33] OPEN NETWORKING FOUNDATION: *Software-Defined Networking: The New Norm for Networks*. [online]. [Accessed 30 November 2013]. Available from: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf>.
- [34] OPEN NETWORKING FOUNDATION: *SDN Architecture Overview 1.1* [online]. [Accessed 1 May 2015]. Available from: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN-ARCH-Overview-1.1-11112014.02.pdf.
- [35] ENNS, R. et al.: NETCONF Configuration Protocol. [online]. [Accessed 11 December 2014].

Available from: <https://tools.ietf.org/html/rfc6241>.

[36] OPEN NETWORKING FOUNDATION: *OpenFlow Switch Specification, version 1.3.0* [online]. [Accessed 5 December 2014]. Available from: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf>.

[37] *OpenFlow Switch Specification, version 1.5.1* [online]. [Accessed 1 May 2015]. Available from: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf>.

[38] REDDY, T. et al.: Port Control Protocol (PCP) Authentication Mechanism. [online]. [Accessed 10 December 2014]. Available from: <http://tools.ietf.org/html/draft-ietf-pcp-authentication-06>.

Appendices

A. Attached DVD Contents

The attached DVD contains the following files:

- `Burda_PCP_in_SDN_Diploma_Thesis.pdf` – this document in PDF format,
- `Burda_PCP_in_SDN_Diploma_Thesis.odt` – this document in ODT format,
- `ofsoftswitch13` – directory containing the source code for *OFSoftswitch13*
- `pcp_sdn` – directory containing the source code of the solution and Unix shell scripts to execute the test scenario,
- `pcp-client` – directory containing the source code for the *PCP client library*
- `ryu` – directory containing the source code for the *ryu* controller framework,
- `unifycore-scripts` – directory containing install scripts from the *UnifyCore* project to simplify installation
- `vm` – directory containing a virtual machine with a test topology

B. Installation

From the CD, copy the `pcp_sdn_source` and `scripts` directories to a directory named `pcp_sdn` anywhere on the disk.

Download and install *VirtualBox*³⁵.

Download the PCP client library³⁶ and extract the contents to `pcp_sdn/pcp-client` directory.

Download *Ubuntu Server* 14.04.1, 64-bit³⁷.

Download *UnifyCore* install scripts³⁸ and extract the contents to `pcp_sdn/unifycore` directory.

Run *VirtualBox* and create a virtual machine from *Ubuntu Server* and name it *PCP_SDN*.

Configure the *PCP_SDN* virtual machine as follows:

- use the bridged adapter and set the MAC address to the MAC address of the host computer (this is necessary in order for the virtual machine to install packages from the Internet),
- add `pcp_sdn` as a shared directory.

Run the *PCP_SDN* virtual machine and install *Ubuntu Server*. Remember the log-in credentials to the virtual machine.

Install *VirtualBox* guest additions to be able to access the `pcp_sdn` directory from the virtual machine³⁹.

Mount the `pcp_sdn` directory:

```
mount -t vboxsf -o defaults pcp_sdn [home directory]/pcp_sdn
```

To run the previous command automatically on startup, open the `/etc/rc.local` file and insert the following lines:

```
mount -t vboxsf -o defaults pcp_sdn [home directory]/pcp_sdn
exit 0
```

Ensure that the `/etc/rc.local` file is executable:

```
chown +x /etc/rc.local
```

Install *UnifyCore*-related packages and files:

```
unifycore/support/install_core.sh
```

Install `nmap` (later to be used to test the communication between hosts):

```
sudo apt-get install iperf nmap
```

Power off the virtual machine, set the adapter to host-only adapter and start the virtual machine

35 Available at: <https://www.virtualbox.org/wiki/Downloads>

36 Available at: <https://github.com/libpcp/pcp>

37 Available at: <http://www.ubuntu.com/download/server>

38 Available at: <https://github.com/unifycore/unifycore>

39 Installation instructions are available at: <https://www.virtualbox.org/manual/ch04.html>

again.

Compiling the PCP client

Change working directory to `pcp-client`:

```
cd ~/pcp_sdn/pcp-client
```

Compile the PCP client:

```
./autogen.sh
```

```
./configure
```

```
make
```

```
sudo make install
```

C. Using the Software

This section describes how to use the software step by step.

The solution allows two hosts between a NAT gateway in a test network to communicate with each other. The hosts cannot communicate unless a mapping entry in the PCP server is created and the corresponding flow entries (translating IP addresses and ports) are installed on the forwarder in the test network.

Preparing the Environment

1. Create three terminal instances and connect to the virtual machine in each instance. These terminal instances are used to display output from each command necessary to run the network (`ryu-manager`, `ofdatapath` and `ofprotocol`).

On Windows, one may use *putty*⁴⁰ to connect to the virtual machine. Use the IP address of the host adapter and the log-in credentials (name and password) to access the virtual machine.

2. In each of these terminal instances, run

```
pcp_sdn/scripts/read_command_output.sh <command>
```

where `<command>` is `ryu-manager`, `ofdatapath`, and `ofprotocol`, respectively.

This and the previous step are not optional, otherwise the commands executed in the `pcp_sdn_test_topology.sh` script will not be operational. The reason is that the output from the commands is redirected to separate named pipes. If no processes read from the named pipes, the commands are blocked.

3. Create another terminal instance and run the script that creates the test topology and initializes the forwarder, the controller and the network application:

```
pcp_sdn/scripts/pcp_sdn_test_topology.sh
```

4. To observe the messages sent in the network, create two additional terminal instances and capture output from each host:

```
sudo ip netns exec 'host1' tcpdump -i 'host1_fw' -w  
~/pcp_sdn/test_pcp_sdn_host1_fw.pcap  
sudo ip netns exec 'host2' tcpdump -i 'host2_fw' -w  
~/share/test_pcp_sdn_host2_fw.pcap
```

Running the Test Scenario

Host 1 to Host 2

1. Generate a PCP request for a TCP connection to the PCP server:

```
sudo ip netns exec 'host1' pcp -i <host1 IP address>:<port> -l 3600  
-t -s <PCP server IP address>
```

The network application currently accepts any PCP server address as the forwarder in the

40 Available at: <http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>

network does not consider the IP address when forwarding PCP messages.

The `-i` option specifies the internal IP address (in this case, `host1` IP address) and the internal port (the `<port>` argument).

The PCP server then generates a PCP response and assigns an external IP address and port from the NAT pool defined in the network application. The output in Figure 1 shows that the PCP client received the external IP address and port highlighted in red. In this example, the PCP server address was set to `172.16.0.1`.

```
khalim@unifycore:~$ sudo ip netns exec 'host1' pcp -i 172.16.0.100:5555 -l 30 -s 172.16.0.1
[sudo] password for khalim:
 0s 000ms 000us INFO    : Found gateway ::ffff:172.16.0.1. Added as possible PCP server.
 0s 002ms 065us INFO    : Added PCP server ::ffff:172.16.0.1
 0s 002ms 510us INFO    : Added new flow(PCP server: ::ffff:172.16.0.1; Int. address: [::ffff:172.16.0.100]:5555; Dest. addr: [::]:0; Key bucket: 30)
 0s 002ms 904us INFO    : Initialized wait for result of flow: 30, wait timeout 1000 ms
 0s 003ms 411us INFO    : Pinging PCP server at address ::ffff:172.16.0.1
 0s 010ms 829us INFO    : Sent PCP MSG (flow bucket:30)
 0s 103ms 847us INFO    : Received PCP packet from server at ::ffff:172.16.0.1, size 60, result_code 0, epoch 29
 0s 104ms 026us INFO    : Found matching flow 30 to received PCP message.

Flow signaling succeeded.
PCP Server IP      Prot Int. IP      port  Dst. IP      port
t  Ext. IP          port Res State Ends
::ffff:172.16.0.1 TCP  ::ffff:172.16.0.100 5555  ::
0  ::ffff:200.0.0.2  49152  0 succ Wed May 13 14:49:08 2015

 0s 104ms 392us INFO    : PCP server ::ffff:172.16.0.1 terminated.
khalim@unifycore:~$
```

Figure 1: PCP Request processed by the PCP server in the controller

2. On `host2` (the external host), run the `nping` server:

```
sudo ip netns exec 'host2' nping --echo-server 'test' -v4
```

3. On `host1` (the internal host), send TCP segments to `host2`:

```
sudo ip netns exec 'host1' nping --echo-client 'test' --tcp
--source-port <port> -v4 <host2 IP address>
```

The `<port>` argument is the internal port specified when generating the PCP request from `host1`.

`nping` first establishes a management TCP connection between the client and the server and then establishes another connection to send “ping” segments.

The packet trace for both `host1` and `host2` are shown in Figures 2 and 3, respectively. As seen from the traces, the IP addresses and ports in the highlighted packets are properly translated by the forwarder in the network. For example, `host1` sends the first packet with its internal IP address and port as the source IP address and port, while `host2` receives the packet with the corresponding external source IP address and port. Likewise, when `host2` replies to `host1`, the external (destination) address and port are properly translated to their

internal counterparts.

6	0.741827	172.16.0.100	200.0.0.200	TCP	210	5555-9929 [PSH, ACK] Seq=1 Ack=97 Win=29056 Len=0
7	0.757001	200.0.0.200	172.16.0.100	TCP	66	9929-5555 [ACK] Seq=97 Ack=145 win=30080 Len=0
8	0.762217	200.0.0.200	172.16.0.100	TCP	178	9929-5555 [PSH, ACK] Seq=97 Ack=145 win=30080 Len=0
9	0.763051	172.16.0.100	200.0.0.200	TCP	226	5555-9929 [PSH, ACK] Seq=145 Ack=209 win=29056 Len=0
10	0.772995	200.0.0.200	172.16.0.100	TCP	114	9929-5555 [PSH, ACK] Seq=209 Ack=305 win=31104 Len=0
11	0.779427	172.16.0.100	200.0.0.200	TCP	54	5555-80 [SYN] Seq=0 win=1480 Len=0
12	0.783018	200.0.0.200	172.16.0.100	TCP	54	80-5555 [RST, ACK] Seq=1 Ack=1 win=0 Len=0
13	0.811699	172.16.0.100	200.0.0.200	TCP	66	5555-9929 [ACK] Seq=305 Ack=257 win=29056 Len=0
14	1.786147	172.16.0.100	200.0.0.200	TCP	54	[TCP spurious Retransmission] 5555-80 [SYN] Seq=0 win=0 Len=0
15	1.798935	200.0.0.200	172.16.0.100	TCP	54	80-5555 [RST, ACK] Seq=1 Ack=1 win=0 Len=0
16	1.805679	200.0.0.200	172.16.0.100	TCP	162	9929-5555 [PSH, ACK] Seq=257 Ack=305 win=31104 Len=0
17	1.805696	172.16.0.100	200.0.0.200	TCP	66	5555-9929 [ACK] Seq=305 Ack=353 win=29056 Len=0
18	2.789585	172.16.0.100	200.0.0.200	TCP	54	[TCP spurious Retransmission] 5555-80 [SYN] Seq=0 win=0 Len=0
19	2.805472	200.0.0.200	172.16.0.100	TCP	54	80-5555 [RST, ACK] Seq=1 Ack=1 win=0 Len=0
20	2.811737	200.0.0.200	172.16.0.100	TCP	162	9929-5555 [PSH, ACK] Seq=353 Ack=305 win=31104 Len=0

Figure 2: Packet trace from host 1

6	0.733261	200.0.0.2	200.0.0.200	TCP	210	49152-9929	[PSH, ACK] Seq=1 Ack=97 win=29056
7	0.733271	200.0.0.200	200.0.0.2	TCP	66	9929-49152	[ACK] Seq=97 Ack=145 win=30080 Len=0
8	0.740568	200.0.0.200	200.0.0.2	TCP	178	9929-49152	[PSH, ACK] Seq=97 Ack=145 win=30080 Len=0
9	0.754073	200.0.0.2	200.0.0.200	TCP	226	49152-9929	[PSH, ACK] Seq=145 Ack=209 win=29056
10	0.754507	200.0.0.200	200.0.0.2	TCP	114	9929-49152	[PSH, ACK] Seq=209 Ack=305 win=311
11	0.768735	200.0.0.2	200.0.0.200	TCP	54	49152-80	[SYN] Seq=0 win=1480 Len=0
12	0.768801	200.0.0.200	200.0.0.2	TCP	54	80-49152	[RST, ACK] Seq=1 Ack=1 win=0 Len=0
13	0.798471	200.0.0.2	200.0.0.200	TCP	66	49152-9929	[ACK] Seq=305 Ack=257 win=29056 Len=0
14	1.778032	200.0.0.2	200.0.0.200	TCP	54	[TCP Spurious Retransmission] 49152-80	[SYN]
15	1.778059	200.0.0.200	200.0.0.2	TCP	54	80-49152	[RST, ACK] Seq=1 Ack=1 win=0 Len=0
16	1.778882	200.0.0.200	200.0.0.2	TCP	162	9929-49152	[PSH, ACK] Seq=237 Ack=305 win=311
17	1.799183	200.0.0.2	200.0.0.200	TCP	66	49152-9929	[ACK] Seq=305 Ack=353 win=29056 Len=0
18	2.783900	200.0.0.2	200.0.0.200	TCP	54	[TCP Spurious Retransmission] 49152-80	[SYN]
19	2.783926	200.0.0.200	200.0.0.2	TCP	54	80-49152	[RST, ACK] Seq=1 Ack=1 win=0 Len=0
20	2.784065	200.0.0.200	200.0.0.2	TCP	162	9929-49152	[PSH, ACK] Seq=353 Ack=305 win=311

Figure 3: Packet trace from host 2

4. To review flow entries installed on the forwarder, run the following command:

```
sudo dpctl 'unix:/tmp/fw.socket' stats-flow | sed 's/, \n\n({table=})/, \n\n1/g'
```

The output contains information about each flow entry on a separate line for easier reading.

Host 2 to Host 1

This scenario can be repeated by running the `nping` server on `host1` and sending “ping” segments from `host2`. In this case, the management port on which `nping` listens to incoming connections is now hidden behind NAT. In order to facilitate communication between the hosts, the management port must be translated as well. Therefore, `host1` must send another PCP request to the PCP server to translate the management port.

It is assumed that the steps in the previous section were already performed.

1. Generate a PCP request for the management port for `nping`:

```
sudo ip netns exec 'host1' pcp -i <host1 IP address>:<management  
port> -l 3600 -t -s 172.16.0.254
```

The management port can be chosen arbitrarily.

2. On `host1`, run the `nping` server:

```
sudo ip netns exec 'host1' nping --echo-server 'test' -v4 -echo-  
port <management port>
```

3. On `host2`, send TCP segments to `host1`:

```
sudo ip netns exec 'host2' nping --echo-client 'test' --tcp
```



```
--source-port <port> --echo-port <management port> -v4 <host2 IP address>
```

4. Observe the `nping` output or the packet traces to see that the packets are properly translated by the forwarder.

Additional Commands for Manipulating Mapping Entries

Specifying explicit external IP address and port

To specify an explicit external IP address and port when creating a mapping via PCP, run

```
sudo ip netns exec 'host1' pcp -i <host1 IP address>:<existing port>
-l <lifetime> -t -s <PCP server IP address> -e <external
IP>:<external port>
```

If the external IP address lies outside the NAT pool defined in the controller, the PCP server will assign a valid IP address from the pool. The same behavior applies to the external port.

If the host already has a mapping assigned, the PCP server merely returns the existing external IP address and port assigned to the host.

Modifying mapping lifetime

To modify the lifetime of an existing mapping, run

```
sudo ip netns exec 'host1' pcp -i <host1 IP address>:<existing port>
-l <new lifetime> -t -s <PCP server IP address>
```

If the lifetime value is small, e.g. 5 seconds, one can observe from the controller output that the mapping expired after 5 seconds. Examining flow entries via `dpctl` also proves that the corresponding flow entries were removed from the forwarder.

Deleting mapping by PCP client

To delete an existing mapping explicitly by the PCP client, run

```
sudo ip netns exec 'host1' pcp -i <host1 IP address>:<existing port>
-l 0 -t -s <PCP server IP address>
```

Creating PEER requests

Previous examples of the `pcp` command generated a PCP MAP request to the PCP server. To generate a PEER request, specify the remote peer (`host2`) IP address and port on which the connection is going to be established:

```
sudo ip netns exec 'host1' pcp -i <host1 IP address>:<existing port>
-l 0 -t -s <PCP server IP address> -p <host2 IP address>:<port>
```

In the current implementation, the PCP server does not distinguish between MAP and PEER requests.

Cleaning up after Running the Test Scenario

1. Stop the execution of following programs or scripts by pressing `ctrl+C` in the corresponding terminal windows:

- `tcpdump` for each host,
 - `read_command_output.sh` for each command,
 - `pcp_sdn_test_topology.sh`.
2. Close all terminal windows connected to the virtual machine.
 3. Log out of the virtual machine.
 4. Power off the virtual machine.

D. Plan of Work

Table 1 contains the original plan of work for the diploma thesis.

Table 1: Plan of work for the diploma thesis

<i>Month (year)</i>	<i>Description</i>
October (2013)	Analysis of PCP and the problems it can solve
November	
December	Complete the document for DT 0 submission
January	Analysis of PCP, user applications, middleboxes, NAT traversal, keepalives, available SDN controllers and forwarders
February	
March	
April	
May	Complete the document for DT I submission
June	Analysis summary, requirements specification
July	
August	
September	Choosing the controller and forwarder for implementation, requirements specification
October (2014)	Prototype implementation, high-level design and architecture of the solution
November	
December	Finish implementing the prototype, complete the document for DT II submission
January	Detailed design, implementation, verification
February	
March	Complete the implementation, verify the implementation, evaluate battery life saved using PCP
April	
May	Complete and submit the final document, prepare for the presentation in June
June (2015)	Presentation preparation

Table 2 contains the assessment of the plan per each unit of work.

Table 2: Plan of work – assessment

<i>Unit of work</i>	<i>Fulfillment</i>
Analysis of middleboxes	Partially done
Analysis of user applications and keepalives	Partially done
Analysis of NAT traversal methods	Partially done
Analysis of PCP	Done
Analysis of WCDMA networks	Partially done – missing analysis of signaling

	traffic impact on RRC
Design	Done
Implementation – PCP server	Done
Implementation – NAT	Done
Verification	Done
Evaluation	Partially done

The analytical part of the thesis, especially middleboxes, user applications and NAT traversal methods, was not provided with sufficient details due to time constraints (additional school subjects and post-Imagine-Cup work and presentations taking most of the time).

The implementation is finished as per the constraints of the requirements and the design of the solution. Of course, many enhancements, be it small improvements or new features, may be incorporated into the solution at some point.

Resumé

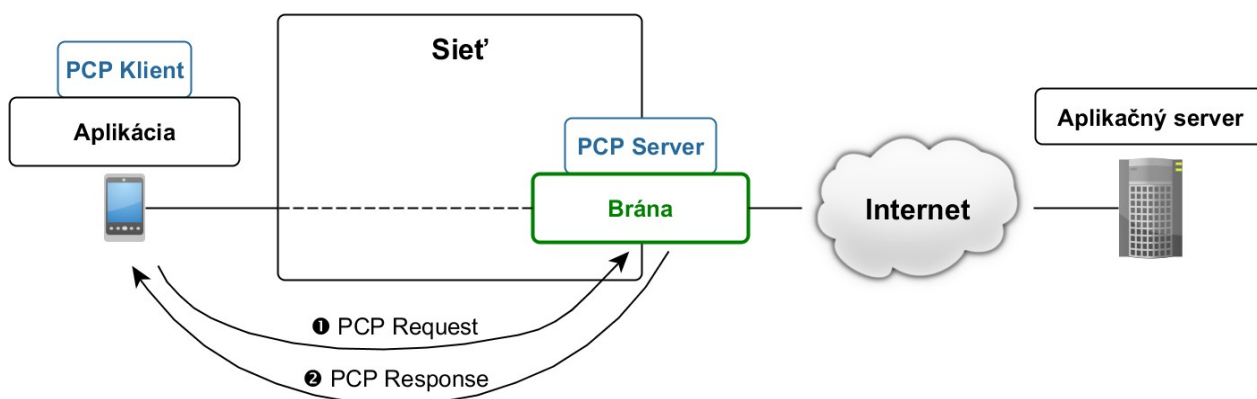
1. Úvod

Tento dokument obsahuje resumé diplomovej práce na tému Protokol PCP v softvérovo definovaných sieťach.

2. Analýza

Používateľské aplikácie, ako napr. VoIP alebo rýchle správy, majú problémy s komunikáciou cez tzv. zariadenia *middlebox* (ďalej len „brána“), napr. brána NAT alebo bezpečnostná brána. Brány obsahujú informácie o mapovaní, ako napr. interná IP adresa, interný port, externá IP adresa, externý port a časovač. Ak časovač vyprší (t.j. žiaden paket nie je vyslaný počas trvania časovača), brána informácie zo svojej internej tabuľky zmaže a tým znemožní zariadeniam komunikovať. Aby aplikácie tomuto javu zabránili, vysielajú správy na udržiavanie spojenia (tzv. správy *keepalive*). Keďže aplikácie hodnotu časovača nepoznajú, snažia sa vysielat' správy *keepalive* vo veľmi krátkych intervaloch, čo zaťažuje sieťové prostriedky, najmä v mobilných sieťach, a znižujú životnosť batérie mobilných zariadení.

Relatívne nový sieťový protokol Port Control Protocol (PCP) [1] umožňuje koncovému zariadeniu riadne komunikovať cez brány tak, že mu PCP poskytne informácie o mapovaní priamo z brán, ako je to znázornené na obr. 2.1. Týmto sa koncové zariadenie dozvie o časovači mapovania a dokáže tak optimalizovať vysielanie správ *keepalive*.



Obr. 2.1: Využitie protokolu PCP v sieťach

Pri vyžiadaní mapovacích informácií pre aplikáciu, PCP klient najprv vyšle správu *PCP Request*. PCP server spracuje požiadavku, nariadi bráne, aby tomuto spojeniu umožnila komunikovať a vyšle naspäť správu *PCP response* s mapovacími informáciami.

V porovnaní s podobnými protokolmi, resp. metódami (STUN, TURN, alebo UPnP-IGD) na komunikáciu cez brány, je protokol PCP pomerne jednoduchý a rýchly (spočíva vo výmene iba dvoch správ) a umožňuje zariadeniam optimalizovať interval vysielania správ *keepalive*. Potenciálnou nevýhodou je, že na každej bráne musí byť nainštalovaný PCP server, čo nemusí byť vyhovujúce pre výrobcov zariadení.

Mobilné siete, vzhľadom na zabezpečenie akceptovateľnej kvality v bezdrôtovej komunikácii,

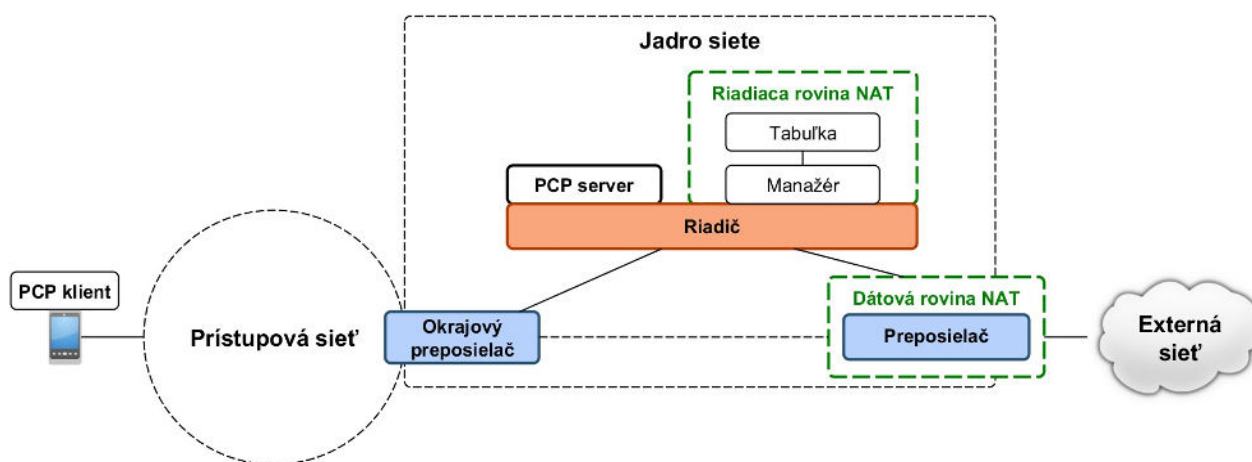
generujú veľké množstvo signalizačných správ pre každú užitočnú správu vyslanú z mobilného zariadenia. Veľmi krátke intervaly správ keepalive spôsobia generovanie veľkého množstva signalizačných správ v sieti, čím sa zahľucuje sieť, a zároveň skracujú životnosť batérie mobilných zariadení. Zvýšením intervalu posielania správ keepalive je možné ušetriť veľké množstvo sieťových prostriedkov a batériu zariadení. Zvýšenie intervalu keepalive je možné dosiahnuť práve pomocou protokolu PCP.

Softvérovo definované siete (SDN) [2] je novodobý koncept v počítačových sieťach, ktorý umožňuje naprogramovať správanie sa siete, čím sa zvýši napr. jej flexibilita a dlhodobá udržateľnosť. Implementácia PCP servera v sieťach SDN umožňuje umiestniť PCP server na riadič (*controller*), čím je brána odbremenená od PCP servera a zabezpečuje sa tak vyššia kompatibilita medzi výrobcami rôznych brán.

Cieľom diplomovej práce je implementovať protokol PCP v sieťach SDN, určiť množstvo signalizácie, ktoré sa dá zredukovať pomocou protokolu PCP a kvantifikovať zvýšenie výdrže batérie pomocou protokolu PCP.

3. Návrh

Obr. 3.1 zobrazuje architektúru navrhovaného riešenia. Súčasťou riešenia je implementácia brány NAT v sieťach SDN, aby bolo možné overiť implementáciu PCP servera. Zároveň je rozdelenie brány NAT na riadiacu a dátovú rovinu príležitosťou optimalizovať siete. PCP server a riadiaca rovina NAT spolu tvoria sieťovú aplikáciu nad riadičom.

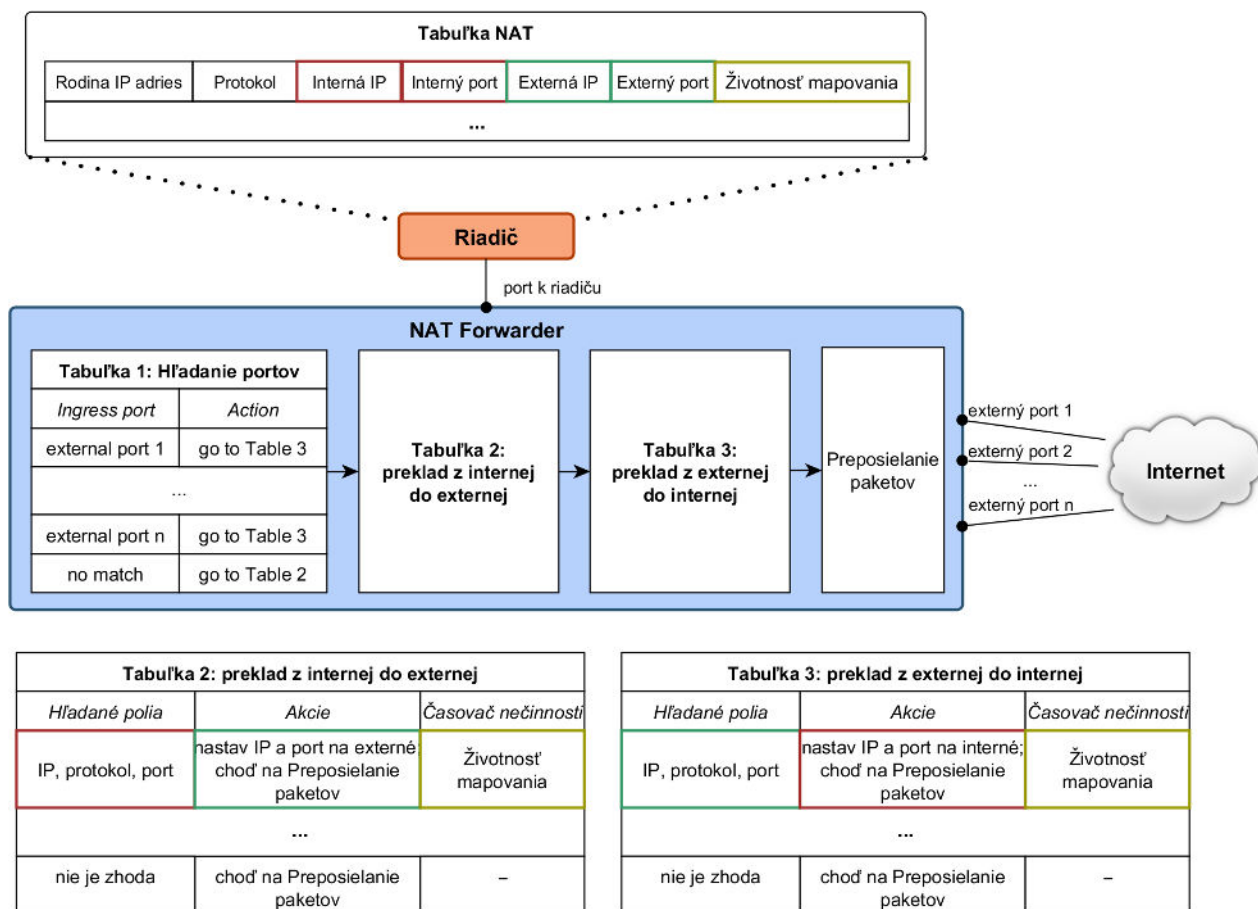


Obr. 3.1: Architektúra navrhovaného riešenia

Riadič a preposielače komunikujú sieťovým protokolom OpenFlow. Ako PCP klient je využitá voľne dostupná knižnica, ktorá zároveň obsahuje príkaz na generovanie PCP správ (t.j. PCP klient). Okrajový preposielač presmerováva PCP správy medzi PCP klientom a PCP serverom. Preposielač, ktorý pokrýva dátovú rovinu NAT (ďalej len „preposielač NAT“), obsahuje pravidlá, ktoré prepisujú IP adresy a porty z externých na interné a naopak. PCP server spracúva správy PCP request a nariadi riadiacej rovine NAT, aby vytvorila mapovanie pre PCP klienta.

Riadiaca rovina NAT obsahuje manažéra, ktorý inštaluje pravidlá do preposielača NAT a tabuľku,

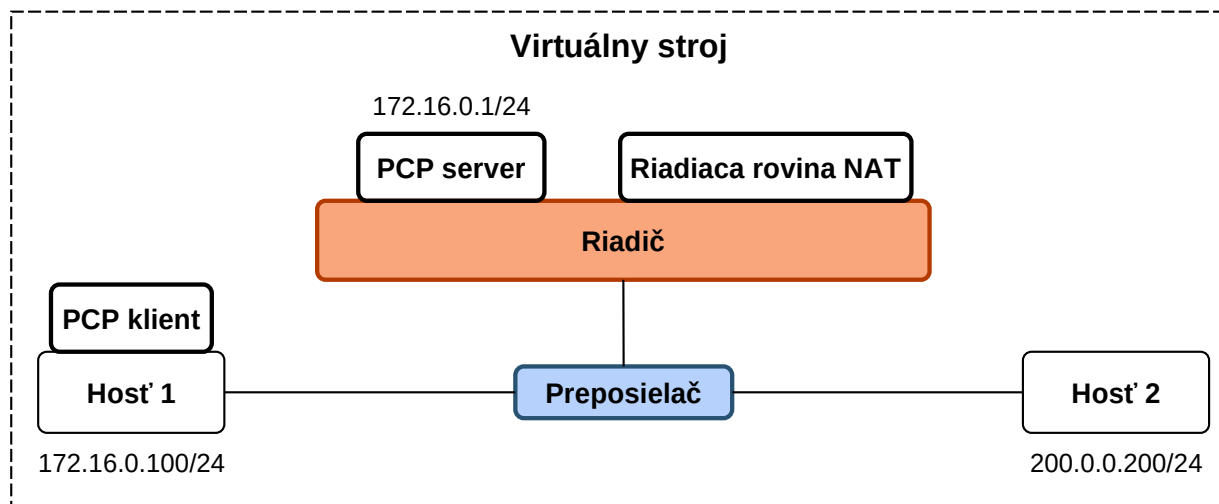
v ktorej sa nachádzajú mapovania priradené jednotlivým PCP klientom. Detailnejšie fungovanie riadiacej a dátovej roviny NAT je znázornené na obr. .



Obr. 3.2: Riadiaca a dátová rovina NAT

4. Implementácia

Na overenie implementácie je implementovaná sieťová topológia znázornená na obr. 4.1.



Obr. 4.1: Testovacia topológia na overenie

V rámci implementácie boli využité nasledovné nástroje: pracovné prostredie na vývoj riadiča *ryu*¹, preposielač *ofsoftswitch* 1.3², knižnica pre PCP klient³, sada príkazov *nmap*⁴. Na overovanie implementácie interný Host 1 vygeneruje správu *PCP request* a prijme správu *PCP response*. Následne Host 2 spustí server *nping* a Host 1 vyšle testovacie segmenty TCP Hostovi 2. Ak komunikácia prebieha (t.j. Host 1 dostane odpovede od Hosta 2), implementácia je úspešne overená.

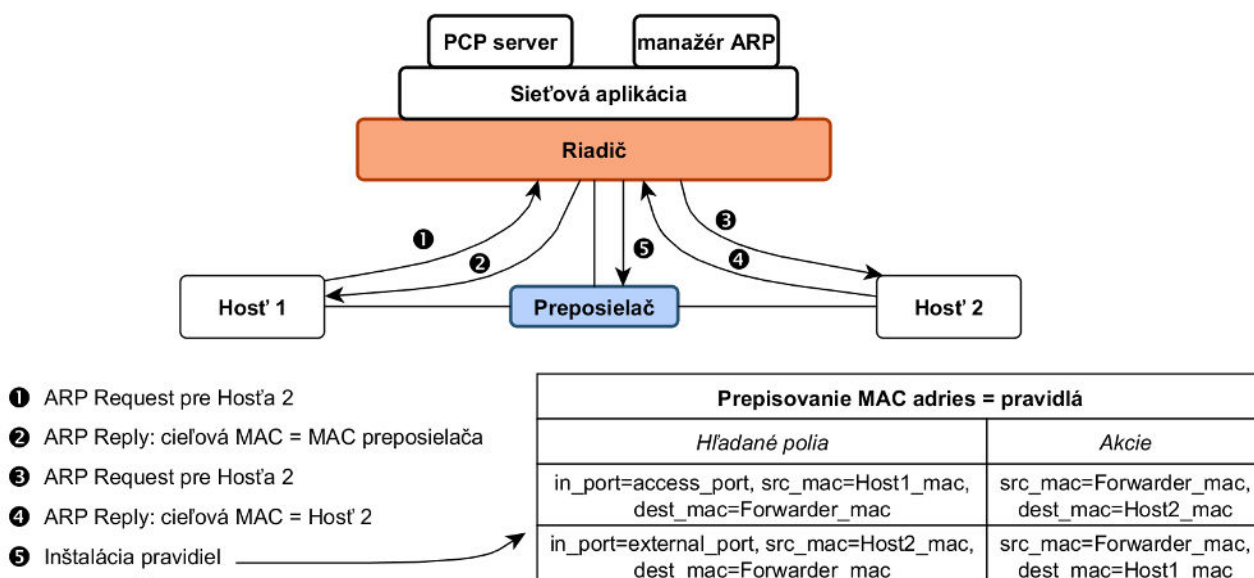
Pri implementácii bolo dodatočne zistené, že je potrebné implementovať spracovanie a riadenie správ ARP. V tradičných sieťach je to samozrejماً funkcionálna, no v sieťach SDN je potrebné túto funkciu implementovať nanovo (prípadne nájsť existujúcu sieťovú aplikáciu, ak existuje). Spracovanie ARP správ je implementované podľa obr. 4.2.

¹ <https://github.com/osrg/ryu>

² <https://github.com/CPqD/ofsoftswitch13>

³ <https://github.com/libpcp/pcp>

⁴ <https://nmap.org/>



Obr. 4.2: Implementácia ARP v riešení

5. Vyhodnotenie

V mobilných sieťach 3G s prístupovou technológiou WCDMA majú mobilné zariadenia stavy pripojenia *Radio Resource Control* (RRC, ďalej len „stavy pripojenia“) podľa množstva prenášaných dát. Príliš časté vysielanie správ keepalive spôsobuje, že mobilné zariadenia spotrebúvajú nezanedbateľné množstvo výkonu, keďže sa častejšie nachádzajú v stavoch pripojenia s vyššou spotrebou.

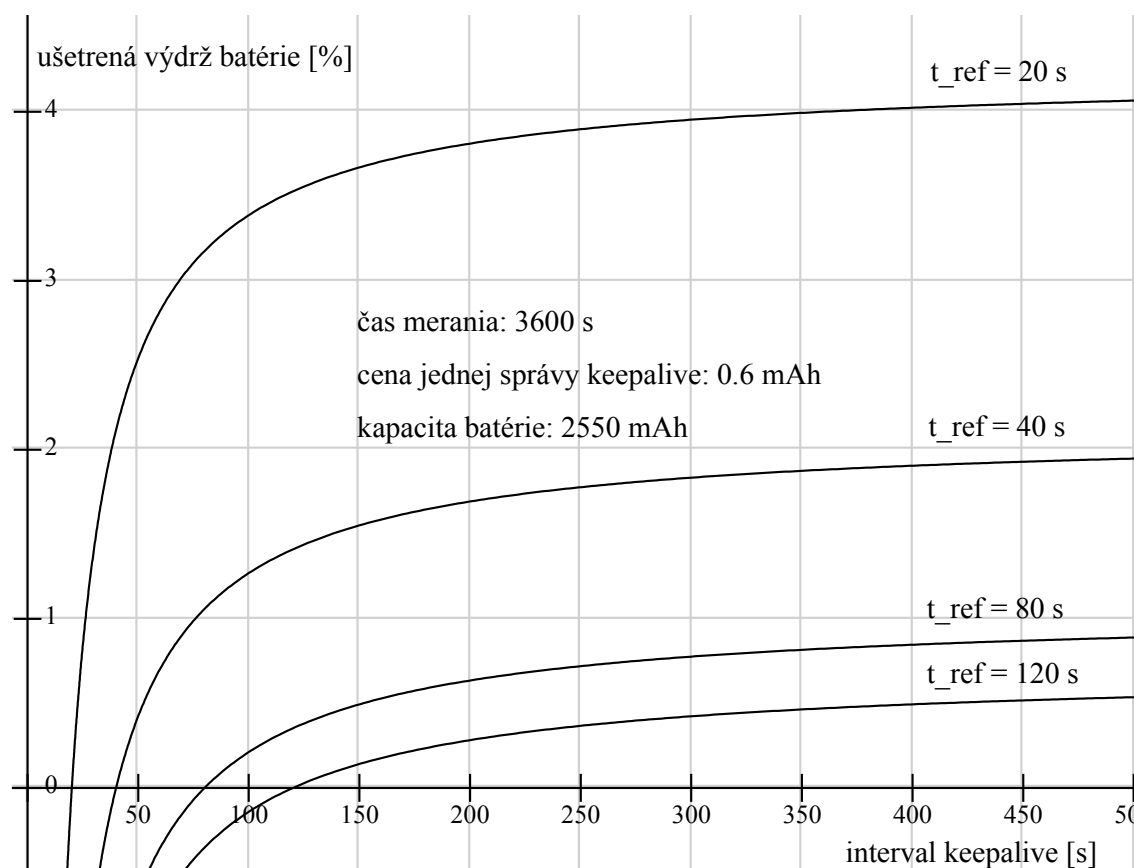
Prechody medzi stavmi pripojenia je potrebné oznamovať sieti. Časté medzistavové prechody generujú nezanedbateľné množstvo signalizačných správ, dôsledkom čoho je zvýšená záťaž v sieti a zhoršená odozva.

Jedným zo spôsobov, ako obmedziť vysielanie správ keepalive, je zvýšiť interval ich vysielania. Zvýšením intervalu na 400-600 sekúnd je možné dosiahnuť celkom značné zníženie spotreby batérie, ako je to znázornené na obr. 5.1 a uvedené v tab. 5.1. t_{ref} označuje pôvodný interval keepalive, t značí nový interval keepalive, T je čas merania, počas ktorého boli vysielané iba správy keepalive, a $cena$ je spotreba batérie na jednu správu keepalive v mAh. Údaje o spotrebe boli prevzaté zo zdroja [3].

Tab. 5.1: Zníženie spotreby batérie mobilného zariadenia pripojeného k sieti WCDMA pri daných kapacitách batérie a referenčných hodnotách

Referenčné hodnoty: $t_{ref} = 20$ s, $t = 400$ s, $T = 3600$ s, cena: 0.15-0.6 mAh

Kapacita batérie	Zníženie spotreby batérie
300 mAh (chytřé hodinky <i>Samsung Gear S⁵</i>)	8.5-34.2%
2550 mAh (telefón <i>Samsung Galaxy S6⁶</i>)	1-4%
7340 mAh (tablet <i>iPad Air 2⁷</i>)	0.35-1.4%



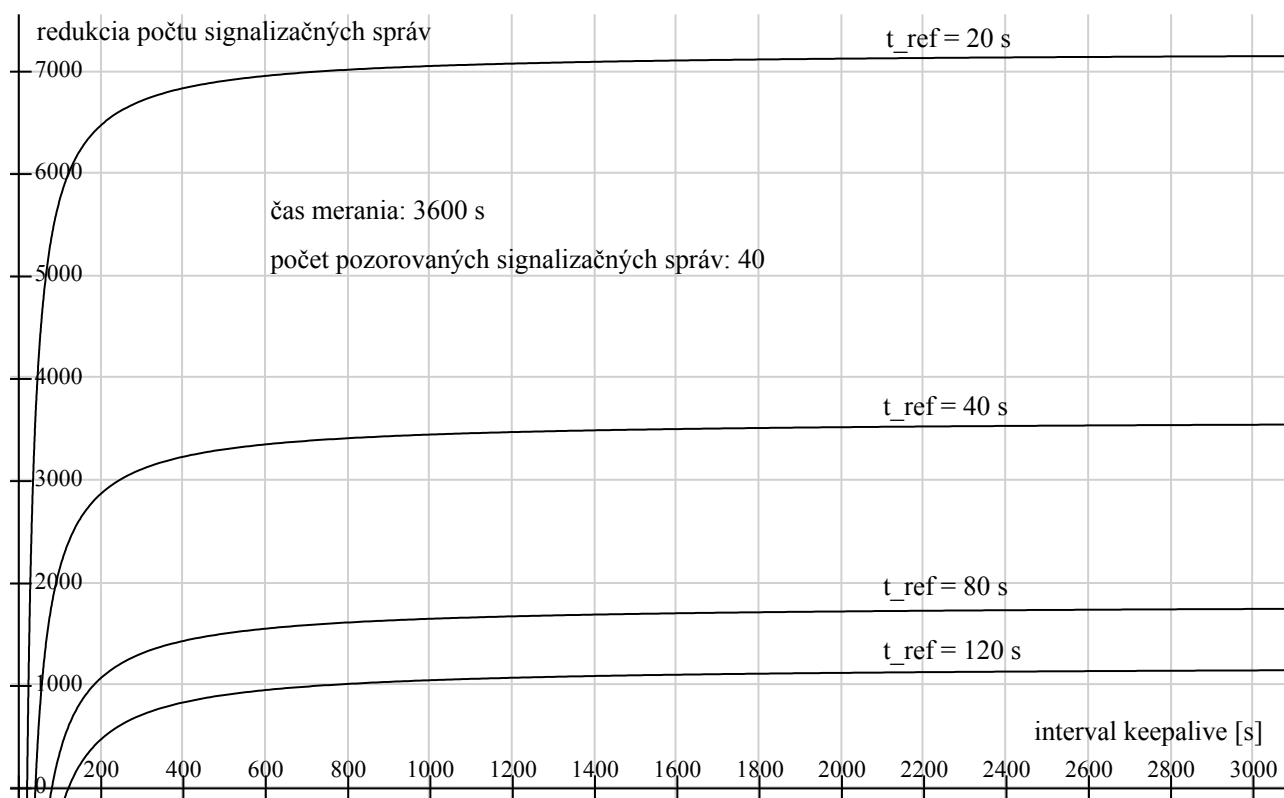
Obr. 5.1: Zníženie spotreby batérie mobilných zariadení v sieťach 3G WCDMA s referenčnými hodnotami

Zvýšením intervalu keepalive na približne 1800 sekúnd je možné dosiahnuť značné zníženie počtu signalizačných správ, ako je to znázornené na obr. 5.2 a v tab. 5.2. Zníženie počtu správ bolo vyjadrené iba pre jednu mobilnú aplikáciu jedného mobilného zariadenia. Pre milióny mobilných zariadení pripojených do siete naraz to môže predstavovať výrazné zníženie záťaže siete.

5 <https://www.samsung.com/uk/consumer/mobile-devices/wearables/gear/SM-R7500ZKABTU>

6 <http://arstechnica.com/gadgets/2015/04/samsung-galaxy-s6-review-its-whats-on-the-outside-that-counts/>

7 <http://arstechnica.com/apple/2014/10/the-ipad-air-2-a-host-of-hidden-upgrades-in-one-skinny-package/>



Obr. 5.2: Redukcia počtu signalizačných správ v sieťach 3G WCDMA s referenčnými hodnotami

Tab. 5.2: Redukcia počtu signalizačných správ vzhľadom na počet signalizačných správ na jednu správu keepalive a referenčné hodnoty

Referenčné hodnoty: $t = 1800$ s, $T = 3600$ s

Počet signalizačných správ na jednu správu keepalive	Pôvodný interval keepalive	Redukcia počtu signalizačných správ	Pôvodný interval keepalive	Redukcia počtu signalizačných správ
40 (pozorované)	20 s	7120	120 s	1120
50 (pozorované)	20 s	8900	120 s	1400
20 (nepozorované)	20 s	3560	120 s	560

Na základe výsledkov je možné skonštatovať, že vhodný interval keepalive pre siete WCDMA je približne 1800 sekúnd. Životnosť mapovania, ktoré by mal protokol PCP nastaviť, by mala byť o istú hodnotu vyššia ako žiaduci interval keepalive – približne aspoň 2000 sekúnd. Pre mapovania, ktoré sú vytvorené správou PCP MAP, by životnosť mapovania mala byť nastavená na dvojnásobok intervalu keepalive, t.j. 3600 sekúnd.

Zhrnutie

Protokol PCP dokáže riešiť problémy aplikácii s prechodom cez brány NAT alebo bezpečnostné brány a zároveň umožňuje aplikáciám získať, resp. si explicitne vyžiadať životnosť mapovania na bránach, vďaka čomu sú schopné optimalizovať interval vysielania správ keepalive.

V súčasných mobilných sieťach môžu mať používateľské aplikácie stále pomerne nízky interval keepalive. Zníženie počtu správ keepalive môže byť považované za softvérovú metódu redukcie nežiaducej réžie v sieti a zníženie spotreby batérie na mobilných zariadeniach. Zvýšením intervalu je možné docíliť značné zníženie signalizačnej réžie, obzvlášť, ak sú k sieti naraz pripojené milióny mobilných zariadení.

Implementácia PCP v SDN zabezpečuje vyššiu flexibilitu a kompatibilitu medzi zariadeniami od rôznych výrobcov. Vzhľadom na súčasnú implementáciu je možné rozšíriť architektúru o nové typy brán, integrovať bezpečnosť PCP, alebo implementovať tzv. viacdomovosť v sieti (angl. *multihoming*).

Literatúra

[1] BOUCADAIR, M. et al. 6887: *Port Control Protocol (PCP)* [online]. RFC. 2013. [cit. 2013-16-10]. Dostupné na Internete: <http://tools.ietf.org/html/rfc6887>.

[2] NADEAU, T.D., GRAY, K. *SDN: Software Defined Networks*. O'Reilly Media, 2013. ISBN 978-1-4493-4230-2.

[3] HAVERINEN, H. et al. Energy Consumption of Always-On Applications in WCDMA Networks. In : *Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th*. April 2007. p. 964–968.