INTERNET-DRAFT
"Internet Protocol Five Fields - Mobile TCP",
Alexey Eromenko, 2015-12-10,
<draft-eromenko-ipff-mops-00.txt>
expiration date: 2016-06-10

Intended status: Standards Track

A.Eromenko
December 2015

Mobile Protocol Stack
for Internet Protocol "Five Fields"

Specification draft

Abstract

This document describes a mechanism, that can be used to live-migrate opened TCP (and other) sessions between a server and a mobile node, even after mobile node moved to a different subnet or disconnected. It also describes limitations and includes NAT traversal scenarios and guidelines for "middlebox" vendors.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 9, 2014.

Copyright Notice

Table of Contents

1. Introduction.....
2. Comparison of Mobile Protocol Stack to Mobile IP.....
2.1 Comparison of Mobile TCP to Multipath-TCP.....
3. Socket creation.....
4. New ICMP Commands.....
4.1 ICMP Migrate.....
4.2 ICMP Anycast Server acknowledgement.....
4.3 ICMP NAT acknowledgement.....
5. Limitations.....
6. Requirements from middleboxes.....
7. Requirements from operating systems.....
8. Security considerations.....
Appendix A: Simple Mobile TCP scenario.....
Appendix B: Mobile TCP Double NAT traversal scenario.....
Appendix C: Future ideas.....
Author Contacts.....

1. Introduction

"basically TCP/UDP session migration... of mobile nodes between
subnets, without losing connectivity"

Mobile Protocol Stack (MOPS) is a solution to the Mobile IP problem.
Mobile IP problem derived from IP itself.
Traditional IP has chained the "location" and the "identity"
of a device by the traditional socket tuple
(s.port, d.port, s.address, d.address), and routing location.

Historically, it was never possible in the original TCP/IP
design to live-migrate a session or socket.
An old socket had to be closed, and a new one had to be opened,
and a new session had to be established.

This is not a problem for desktop computers, as they always
reside in the same particular LAN, and the same subnet.
But it becomes a problem, if you want to move "mobile nodes" from
one subnet to the next, without losing connectivity.

This whole idea comes from "Live Migration" in the virtualization
field, where you can migrate a living virtual machine, and it
keeps working. MOPS, in fact, will allow me to live-migrate

between Ethernet, 4G cellular and WiFi at home, and keep working.

2. Comparison of Mobile Protocol Stack to Mobile IP

Traditional Mobile IP solution is way too complex and sub-optimal.
For example Mobile IPv6, RFC-6275 spec takes over 400 kilobytes !
Mobile IPv4, RFC-5944 is similar in spirit.

Mobile IP requires setting up a Home Agent (Router) and Foreign Agent (Router), making tunnel between them, having access to both, configuring both, and it is way too complex...
In the end packets are travelling slowly over the network, via sub-optimal routes, slowing down traffic.
Needless to say, that if you *do not* control said routers, there is *no way* to make Mobile IP work.

MOPS is a fairly elegant solution, that covers most of the real-world use-cases, that gives optimal routing, doesn't require access to any routers, let alone configuration what-so-ever !!!
But it requires changes to the client & server operating systems in question.

Mobile Protocol Stack (MOPS) is sub-divided into "Mobile TCP" and "Mobile UDP" parts.

This is a clever trick achieved through Internet Control Message Protocol extensions and special sockets and better integration between layers.

2.1 Comparison of Mobile TCP to Multipath-TCP

A new standard recently popped up, Multipath TCP.
This standard allows you to add or remove subflows (i.e. new TCP sockets), allowing for migration between subnets as-long-as-one flow remains intact. I.e. you must have 3G enabled and migrate from WiFi subnet A to WiFi B.
It will not be able to migrate after disconnecting all subflows.

Mobile TCP, by contrast, allows re-connecting even after full disconnect, as long as TCP session did not timeout.

3. Socket creation

Sockets created with "MOBILE_ALLOWED" flag can later be remapped to a different IP address and/or port.
When creating a socket, the application SHOULD optionally specify

"MOBILE_ALLOWED" flag to the TCP or UDP protocols.

For UDP the new flag, if not specified, SHOULD be disabled by default. This is due to the lack of built-in ID, such as TCP Sequence number.

TCP, on the other hand, SHOULD have "MOBILE_ALLOWED" flag enabled by default, if not set by app.
This prevents the need to rewrite legacy applications, and gives them mobile capability provided by an Operating System upgrade.

When socket is created with "MOBILE_ALLOWED" flag, it is RECOMMENDED to increase timeout timers for such session.

4. New ICMP Commands

4.1. ICMP Migrate (version 1)

Description

This command instructs a remote server to remap the binding of an already opened socket, changing the tuple, allowing the server to send responses to a different IP address and port.

Historically, it was never possible in the original TCP/IP.
An old socket had to be closed, and a new one had to be created.

The destination port and IP do not change, as they belong to the server.

If used improperly, this command can be a security-hazard, allowing remote hackers to hijack running sessions.
For this reason, two defensive mechanisms were developed:
TCP Sequence number and application-supplied Unique ID.

TCP Sequence number provides a basic protection against session hijacking, without the need to modify existing applications, as a TCP stack provides built-in protection from remote hackers.
It doesn't protet from Man-in-the-middle (MITM) attacks however.

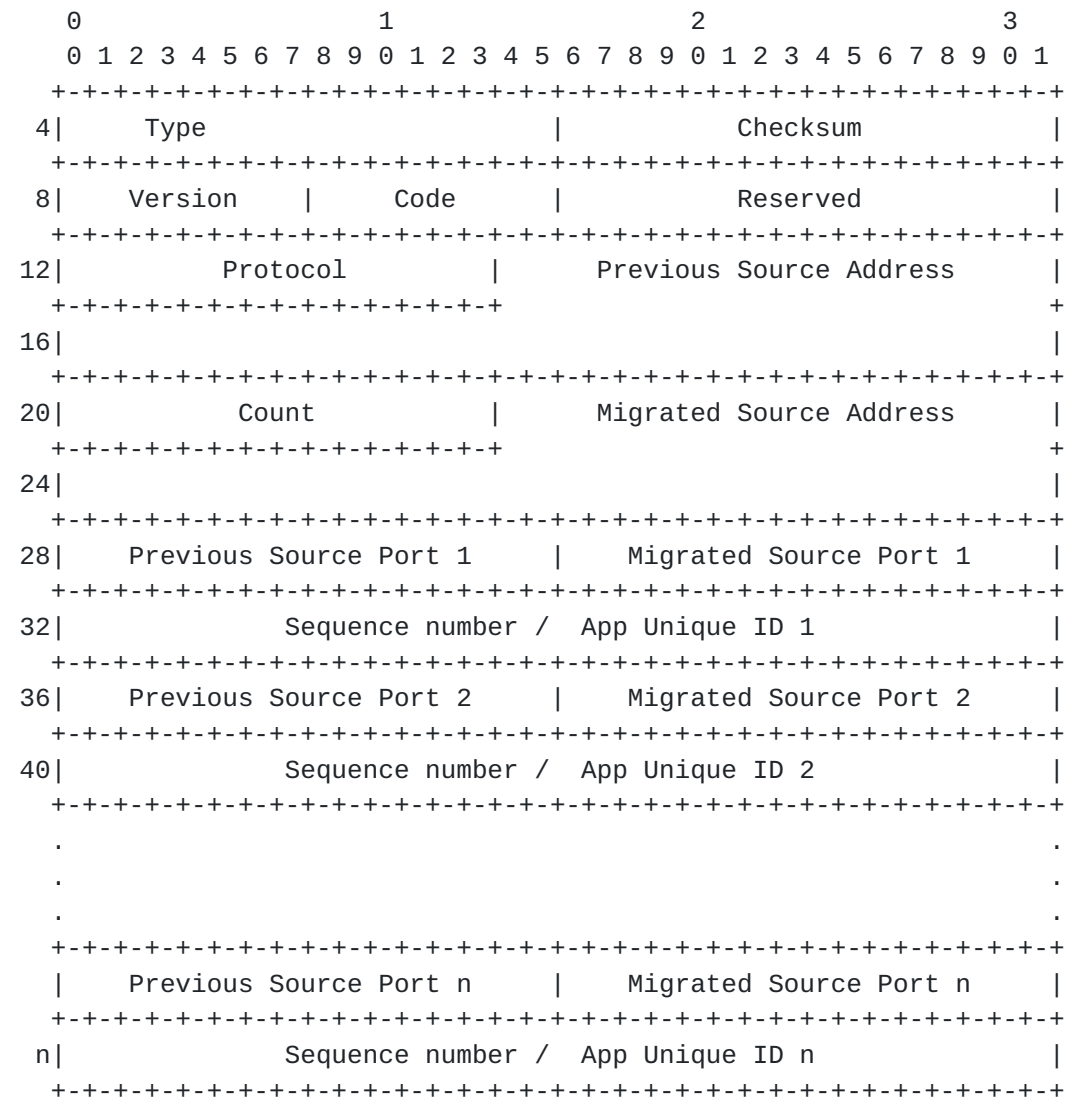
Unique ID provides additional protection against Deniel-of-Service attacks, and other session hijacking attempts by evil hackers, and it optionally can use encryption to make session hijacking unbreakable, including MITM hackers.

If an application requires "Unique ID", it must be supplied instead of TCP Sequence number.

For example, a simple Unique ID can be a "Player ID" for games, or

a "Video stream ID" for video-on-demand.
Effective for basic defences, but not strong.

For encrypted applications, it can be a one-time-generated
hash of a newly generated shared secret key.
After one successful migration, this hash (and key) must be
re-generated, by a predefined deterministic algorithm.
This would be much stronger defence.



(bytes)

ICMP Fields:

Type (16-bit)

100 (?) some number assigned by IANA

Checksum (16-bit)

   The checksum must be computed according to ICMP spec.

Version (8-bit)

   1

Code (8-bit)

   0 = migrate

   1 = migrate atomic (rollback, if one port migration fails)

   2 = migration success!

   3 = migration failed.

   Code 0 or 1 is a request by the mobile node client.
   Code 2 or 3 is a reply by server. Server must send all the details
   of the migrate command, with all the fields intact, as received.

Reserved (16-bit) field

   Should be set to zero by transmitter, ignored by receiver.

Protocol (14-bit)

   Upper-level Protocol, such as TCP or UDP. Same as in IP header.

   If you want to mass-migrate applications using several protocols,
   like both TCP and UDP, you will need to issue multiple
   Mass-migrate commands.

Previous Source Address (50-bit)

   The source IP address, before migration.
   Must use Translated Source IP, if behind NAT.

Count (14-bit)

   Amount of ports to be migrated (1...n)

Migrated Source Address (50-bit)

   The new source IP address, where Mobile Node (MN) would
   like to receive replies of a continued session.

Previous Source Port (16-bit; n fields)

   The source TCP port of a socket, before migration.
   Must use Translated Source port, if behind NAT.

Migrated Source Port (16-bit; n fields)

The new source TCP port of a socket, where Mobile Node (MN) would like to receive replies of a continued session.

Sequence number / Unique ID (32-bit; n fields)

Sequence number:

For TCP protocol, it would be the TCP sequence number. The mobile node client must get it from its own TCP stack, as-if preparing the next TCP packet for transmission.

Unique ID:

It must be an application-supplied value, matching on both ends. While TCP sequence number provides a basic protection, against session hijacking, a Unique ID improves security.

If TCP application on the server-side provides "Unique ID" requirements, it *MUST* be supplied, instead of TCP seq. number.

If TCP application does not define "Unique ID", then Mobile Node must provide TCP sequence number.

UDP:

Because UDP protocol lacks a "sequence number", the only way to migrate it, is if server side application asks for a Unique ID, and mobile node matches it.

Other transport-layer protocols MAY specify their own Sequence identifier or equivalent.

Technical details:

Replies:

On first attempt, either "ICMP migration failed." or an "ICMP migration success!" should be returned to client. A reply must have the same parameters, with all fields intact, as a request, except Code.

If at least one port migration failed, it should inform the mobile node as "ICMP migration failed.", but returning only the ports, that failed to migrate.

Typically, request code 0, ports that have migrated successfully should stay migrated (i.e. do not roll-back). But if request was send with code 1 "migrate as one atomic operation", all ports must be rolled-back, if one of the ports failed to migrate (due to Unique ID match failure, TCP sequence

number mismatch, port timeout, or otherwise).

Retransmission by client:

If no reply was received in the next few seconds (I recommend 2 sec), the mobile node should re-send "ICMP migrate" up to 3 times. After this declare old socket as "dead", and report error to the application.(like it would after receiving TCP reset)

Timing & Retransmission by server:

If "ICMP migrate" command received with the same previous source IP address + source port too often, it should be silently discarded by server, as it indicates either a hacker attempt, or a flapping link. Recommended is 5 seconds interval before accepting new attempt. After timeout expires, server can accept new commands and send new ICMP migration reply.

4.2 ICMP Anycast Server acknowledgement: (version 1)

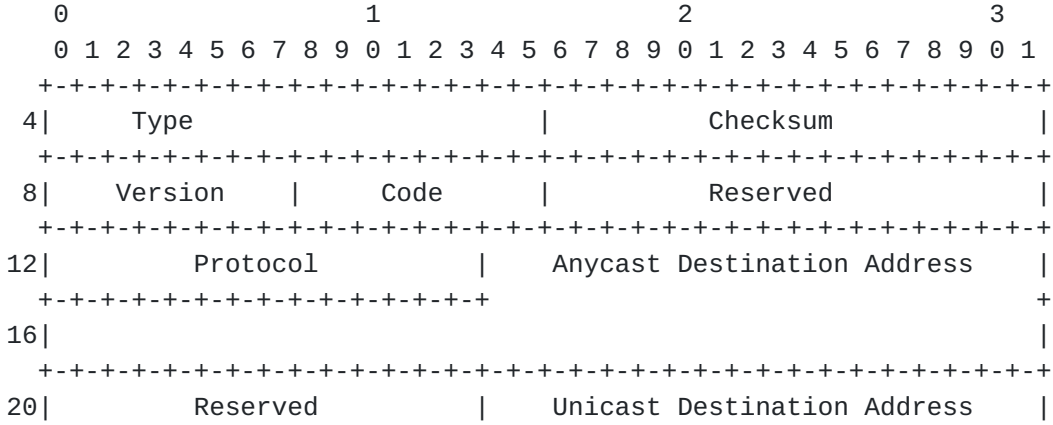
Description

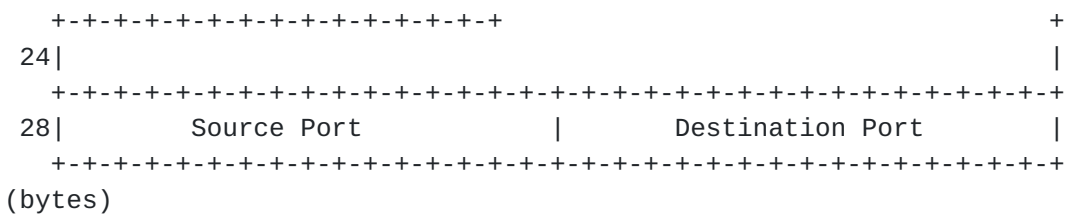
This command informs a mobile node client, how-to re-connect to this particular Anycast server, should TCP connection fail.

Only an Anycast server (or its middlebox) can send this message to the Mobile Node, when it got a new incoming TCP session SYN with "MOBILE_CAPABLE" flag.

Historically Anycast servers were used, as a connectionless system with UDP, for quick queries such as DNS, or very short TCP requests, because the path could change mid-way.

Using Mobile Protocol Stack, it is possible to reconnect again, and run much more complex, connection-oriented services on Anycast servers. This requires that Anycast servers have unicast addresses.





ICMP Fields:

Type (16-bit)

101 (?) some number assigned by IANA

Checksum (16-bit)

The checksum must be computed according to ICMP spec.

Version (8-bit)

1

Code (8-bit)

0

Reserved fields

Initialized to zero on transmission, ignored by receiver.

Protocol (14-bit)

Upper-level Protocol, such as TCP or UDP. Same as in IP header.

Anycast Destination Address (50 bits)

The IP address of the server, to which MN is currently connected.

Unicast Destination Address (50 bits)

The unique Unicast IP address of the server, to which Mobile Node can re-connect.

Source port (16 bits)

Protocol source port (of mobile node) of an ongoing session.

Destination port (16 bits)

Protocol destination port (of anycast server) of an ongoing session.

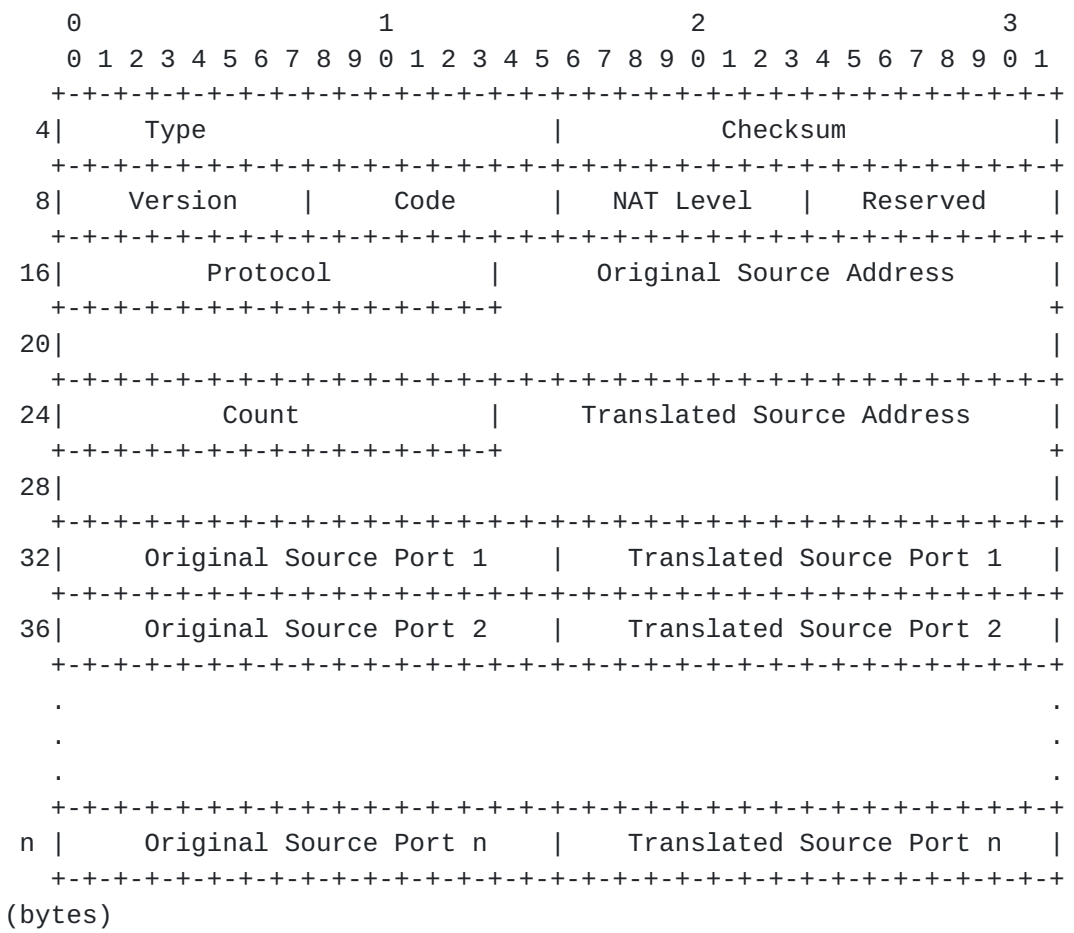
4.3 ICMP NAT acknowledgement: (version 1)

Description

This command solves NAT traversal problem by instructing
a mobile node client, that it's external
or translated addresses are different from internal or original.
This information will be required down the road to construct
a proper ICMP migrate command parameters.

Only a NAT Router can send this message to the client, when
it intercepts a new TCP session (with TCP SYN flag), or when
it intercepts an ICMP migrate command.

+++TODO: (and MOBILE_CAPABLE TCP flag ?)



ICMP Fields:

Type (16-bit)

102 (?) some number assigned by IANA

Checksum (16-bit)

   The checksum must be computed according to ICMP spec.

Version (8-bit)

   1

Code (8-bit)

   0

NAT Level (8-bit)

   Typically 1, for one-level of NAT routing.

   The NAT Router must decide if it is the closest level to client,
   it would send level = 1, but if it received "NAT acknowledgement"
   from another router with matching parameters, like this:

   my NAT table:
   NAT level=1
   original.IP address: A
   original.port: B
   translated.IP : C
   translated.port : D

   Received ICMP NAT acknowledgement from 2nd router:
   NAT level=1
   original.IP address: C
   original.port: D
   translated.IP : E
   translated.port : F

   NAT Router sees that your "translated IP + port" fields match
   a NAT ack from a 2nd router, in it's "original IP address + port"
   fields, NAT Router knows there is one more level to go.

   It should increment the NAT level by +1, and send to ICMP NAT
   acknowledgement client.
   And add this info to own NAT table, as 2nd level, as more levels
   may exist.

   Up to 255 levels are supported.
   See "Appendix B" for example.

Reserved (8-bit)

   Initialized to zero on transmission, ignored by receiver.

Protocol (14-bit)

Upper-level Protocol, such as TCP or UDP. Same as in IP header.

Original Source address (50 bits)

Source IP address of the client, before Network Address Translation.

Count (14-bit)

The amount of ports, that were NAT-ted.
Typically 1.
When a new (TCP) session begins, it is only one port.

Multiple ports can be NAT-ted at once, only if NAT router intercepted "ICMP migration success" acknowledgement.

Translated Source address (50 bits)

Source IP address of the client, after Network Address Translation.

Original Source port (16 bits; n fields)

Upper-level Protocol port, before Network Address Translation.

Translated Source port (16 bits; n fields)

Upper-level Protocol port, after Network Address Translation.

5. Limitations:

Protocols, that open TCP ports in reverse direction (like FTP), from server to client, can have trouble. Limitation. So maybe Mobile IPFF will be needed for corner cases.

I believe, that Application Specific Extensions (ASEs) will need to be written for problematic protocols, just like with NAT. This allows for certain vendor differentiation.

Limitations: Protocols that are NAT-alien may fail for MOPS.
...for a similar reason: they exchange IP-level data as application-layer payload.

Other limitations includes Firewalls and middleboxes, that have TCP initial sequence number (ISN) randomization. This feature must be disabled.
This feature is incompatible with Mobile TCP.

In addition, all NATs (and load balancers) need to be Mobile TCP/MOPS-aware, at both source and destination networks.

The good news, is that because Mobile TCP is standardized together
with IPFF, old firewalls and NAT Routers must be thrown
out-the-window anyway.
(or get a major software upgrade...)

6. Requirements from middleboxes, firewalls and NAT routers:

support for ICMP NAT acknowledgement.
ability to intercept "ICMP redirect" command.
Never modify initial sequence number (ISN) for TCP sessions.

"ICMP migrate" command MUST be intercepted by NAT, and modified,
just like a new TCP session would.
"ICMP NAT acknowledgement" command MUST be implemented in IPFF
NAT Routers.

Also when "ICMP migrate" command is intercepted by a stateful
firewall, a new rule should appear on it's firewall table,
something like "TCP, ESTABLISHED", with proper socket.

7. Requirements from operating systems:

For MOPS to work, changes the Operating System is required.
Ability to change and remap opened sockets on-the-fly.
Opening sockets with "MOBILE_ALLOWED" flag.

Second, it must provide TCP sequence number to IP/ICMP layer,
both to send and to receive "ICMP migrate" command.

Third, it must allow to pass Application-level parameters,
such as Unique ID down to IP/ICMP layer.

8. Security considerations:

With at 1st look it looks very insecure to give anyone an option
to hijack a session from remote, but digging deeper we discover
the following:

If a hacker is outside the path (your subnet, or your server's
subnet or ISP), not in the middle, he has no realistic chance to
guess both the 50-bit source IP address, and 16-bit port and 32-bit
TCP sequence number, and/or Unique ID by application layer.

If he *is* a man-in-the-middle attacker (MITM), then he can
hijack TCP session by listening to SYN/ACK, and knowing the next
sequence number, spoofing source IP address, and if his computer
answers to packet before yours,
session will get hijacked with normal TCP (without Mobile TCP).
So in this case Mobile TCP is not worse than the standard TCP.

The good news, is that sensitive data transmission should be
encrypted, and in reality it is. Both SSL (HTTPS) and SSH are

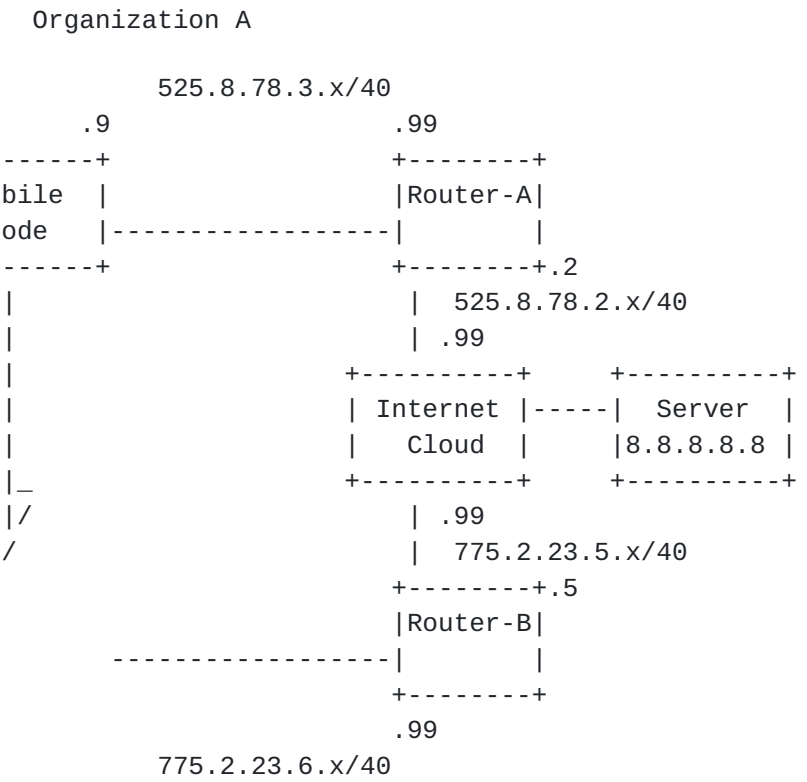
commonplace, and if he hijacks an encrypted session,
it will be just shut-down, as he does not possess the keys.

Mobile nodes should use Initial Sequence Number (ISN) randomization.

For best security, use encrypted protocols, and one-time
Unique ID. This way, hacker cannot hijack session, even if he is a
man-in-the-middle.

Appendix A: Simple Mobile TCP scenario: (without NAT or Firewall)

MN = Mobile Node. A typical smartphone, tablet or laptop web client.
Remote Server 8.8.8.8.8, is a typical HTTP server, listening at
TCP/80.

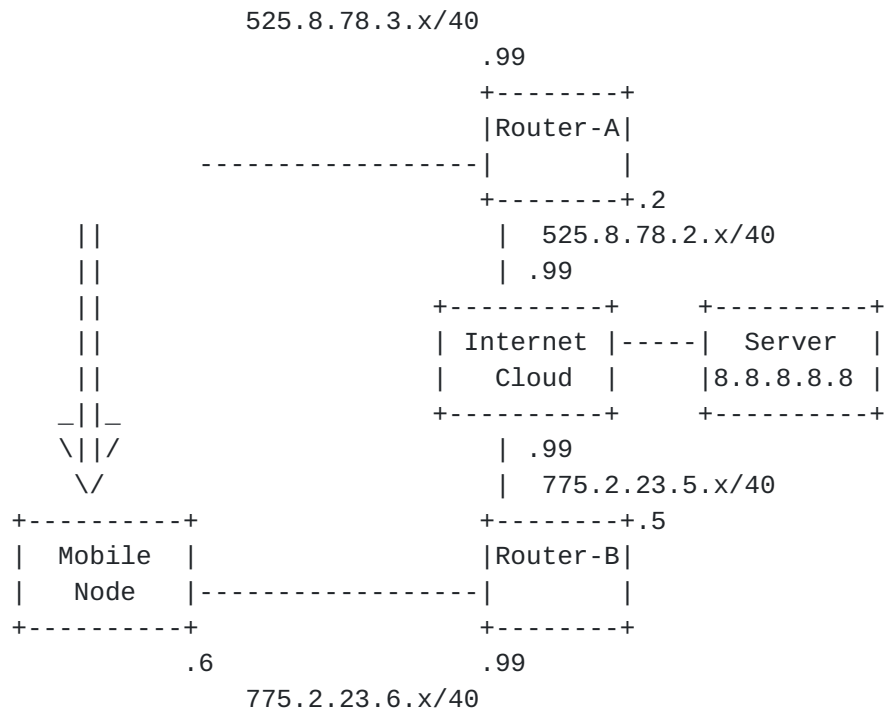


Mobile Node opened a socket to a remote server.

source.port = 1027, source.IP = 525.8.78.3.9,
destination.port = 80, destination.IP = 8.8.8.8.8

...a typical socket tuple. Nothing special.
Now Mobile Node decided to go to a different location, and joined
a different network:

Organization A



Organization B

Our Mobile Node got a new IP address via DHCP from Router-B;
775.2.23.6.6
And due to losing previous connecting, it must look it's own
sockets table... ohh we have a connection from "525.8.78.3.9",
which is no longer our address !
We must migrate this old connection to our new address !

Let's send an "ICMP migrate" command.

"ICMP migrate" requires several parameters, including our previous
IP+port, and our new IP+port:

Protocol = 6 (TCP)
TCP sequence number (must query its own OS) - a very long 32-bit
number.
previous source port = 1027
previous source IP = 525.8.78.3.9
migrated source port = 2098 (or whatever free port it has)
migrated source IP = 775.2.23.6.6

Server must compare against it's own opened sockets, and verify TCP
seq. number.

The socket now will be re-mapped (on both client and server):

source port = 1027, source IP = 525.8.78.3.9,
destination port = 80, destination IP = 8.8.8.8
to:

source port = 2098, source IP = 775.2.23.6.6,
destination port = 80, destination IP = 8.8.8.8

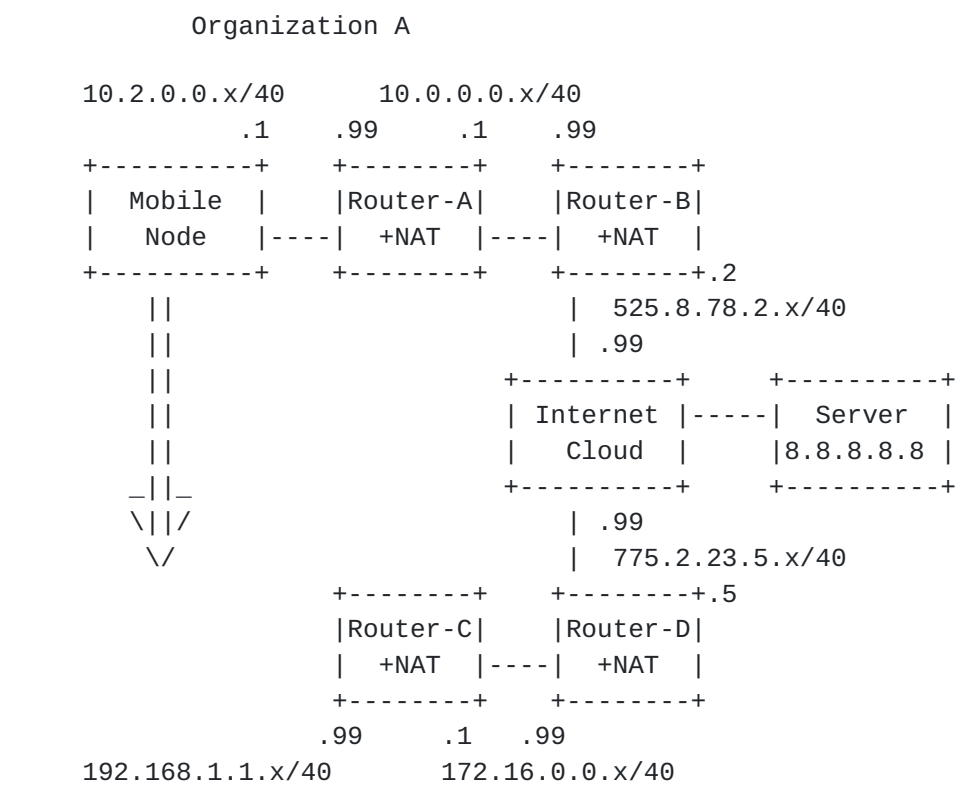
Server will send "ICMP migration success!"

And hoopla ! Connection will continue !

Appendix B: Mobile TCP Double NAT traversal scenario:

Source Network Address Translation (NAT) with Port Address Translation (PAT)

MN = Mobile Node. A typical smartphone, tablet or laptop web client. Remote Server 8.8.8.8.8, is a typical HTTP server, listening at TCP/80.



Mobile node will be migrated to a new network.
Mobile node now it got a new IP address, 10.2.0.0.1 via DHCP from Organization A.

NAT traversal table on Mobile Node: (it must remember it!)
(empty)

After Router-A got TCP packet, it creates a NAT traversal table:

NAT Level	1	2	3	n

```
    original source port =   |1027        |     |
    original source IP   =   |10.2.0.0.1  |     |
    translated source port = |5050        |     |
    translated source IP   = |10.0.0.0.1  |     |
```

NOTE: NAT traversal table and NAT table can be the same data structure or different.

...and sends "ICMP NAT acknowledgement" with the highest NAT level, in this case = 1. via Unicast to the address of "original source IP", in this case Mobile Node (MN).
Mobile Node, after receiving this NAT traversal table, will have it also.

After Router-B got TCP packet, it creates a NAT traversal table:

```
    NAT Level                |      1     |  2  |  3  |  n
                             |------------|     |
    original source port =   |5050        |     |
    original source IP   =   |10.0.0.0.1  |     |
    translated source port = |6050        |     |
    translated source IP   = |525.8.78.2.2|     |
```

...and sends "ICMP NAT acknowledgement" with the highest NAT level, in this case =1. via Unicast to the address of "original source IP", in this case Mobile Node (MN).
Router-A, after receiving this NAT traversal table, will add a new level to his table.

Router-A NAT traversal table:

```
    NAT Level                |      1     |      2     |  3  |  n
                             |------------|------------|     |
    original source port =   |1027        |5050        |     |
    original source IP   =   |10.2.0.0.1  |10.0.0.0.1  |     |
    translated source port = |5050        |6050        |     |
    translated source IP   = |10.0.0.0.1  |525.8.78.2.2|     |
```

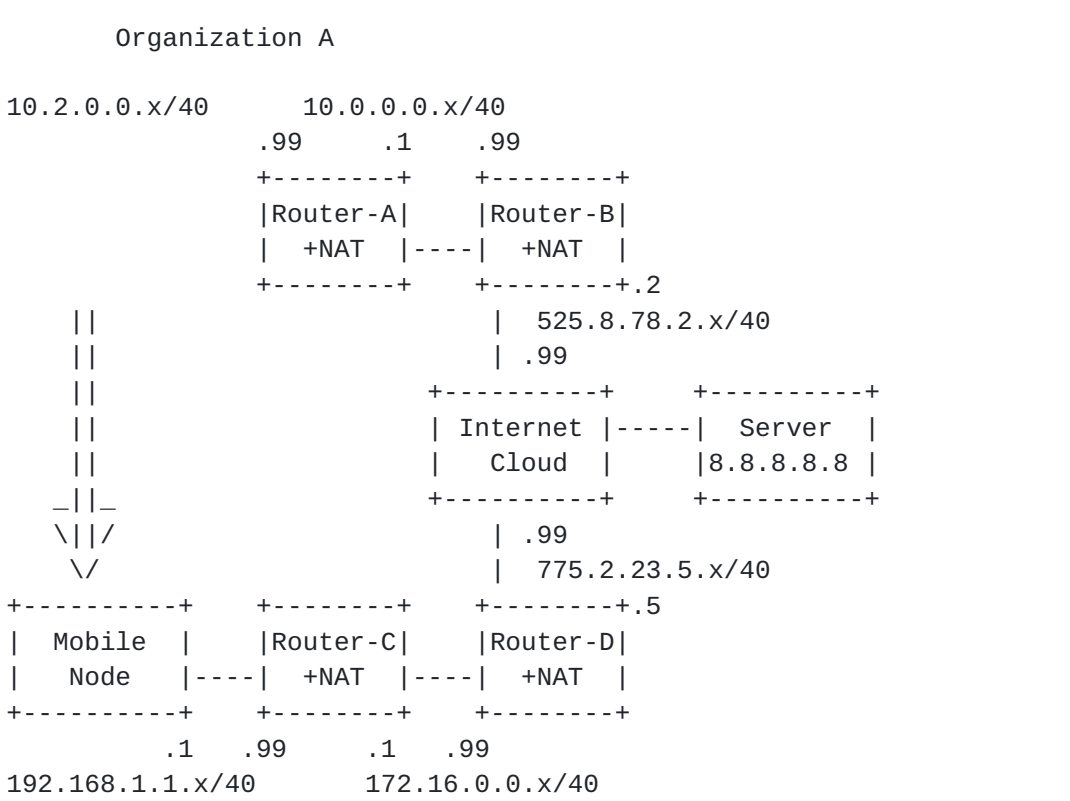
...and due to this update, Router-A sends "ICMP NAT acknowledgement" with the highest NAT level, in this case =1.
via Unicast to the address of "original source IP", in this case Mobile Node (MN).

packet looks like:
type: ICMP NAT acknowledgement (Unicast)

```
    NAT Level                |      2     |
                             |------------|
    original source port =   |5050        |
    original source IP   =   |10.0.0.0.1  |
    translated source port = |6050        |
    translated source IP   = |525.8.78.2.2|
```

Mobile Node now also got the same table as Router-A.
So at this stage Mobile Node already knows it's translated source
IP & port numbers, which it will use to send "ICMP migrate" command.

Mobile Node (MN) now is ready to leave its home subnet, and can
send "ICMP migrate" command.



In reality, this "NAT learning" will happen in a blink of an eye,
under 100 milliseconds. And it will be ready to migrate.

Mobile Node has physically migrated to Organization B, and got a new
 address from the local DHCP server, 192.168.1.1.1. (also NAT
Router-C), and lost it's previous address, namely "10.2.0.0.1".
At this stage, our Mobile Node must look at all packets sourcing
from it's previous address and issue sends "ICMP migrate" to the
Server (8.8.8.8),
with whom it had previously opened sockets (TCP).

protocol = TCP
TCP sequence number = XYZ
previous source port = 6050
previous source IP   = 525.8.78.2.2 (the translated IP address and
port it learned from "ICMP NAT acknowledgements").
migrated source port = 1027
migrated source IP   = 192.168.1.1.1

```
  (new IP address learned via DHCP)

... and at this very moment, client remaps it's socket:

source port = 1027, source IP = 10.2.0.0.1,
destination port = 80, destination IP = 8.8.8.8
to:
source port = 1027, source IP = 192.168.1.1.1,
destination port = 80, destination IP = 8.8.8.8

-----------------------------------------------------------------
Router-C, when receiving "ICMP migrate" command, must intercept it,
change it's "migrated" fields.

Now: Router-C MUST:
1. add a new entry to his own Network Address Translation (NAT)
   Table.
2. send "ICMP NAT ack" to "original source IP" from it's NAT table.
3. send the new "ICMP migrate" command to target destination
   (8.8.8.8)

1.+ 2. new NAT entry + NAT ack:
NAT Level                |      1      |
                         |-------------|
original source port =   |5050         |
original source IP   =   |192.168.1.1.1|
translated source port = |6050         |
translated source IP   = |525.8.78.2.2 |

protocol = TCP
TCP sequence number = XYZ (must not touch it)
previous source port = 6050
previous source IP   = 525.8.78.2.2
migrated source port = 2038 (or whatever random free port it has;
                             was 1027)
migrated source IP   = 172.16.0.0.1 (was 192.168.1.1.1)
-----------------------------------------------------------------
Router-D, when receiving "ICMP migrate" command, must change
"migrated" fields to this:

protocol = TCP
TCP sequence number = XYZ (must not touch it)
previous source port = 6050
previous source IP   = 525.8.78.2.2
migrated source port = 4128 (or whatever random free port it has;
                             was 2038)
migrated source IP   = 775.2.23.5.5 (was 172.16.0.0.1)

Now Router-D MUST:
1. add a new entry to his own Network Address Translation (NAT)
   Table.
```

2. send "ICMP NAT acknowledgement" to "original source IP" from
   it's NAT table.
3. send the new "ICMP migrate" command to target destination
   (8.8.8.8.8)

Server (8.8.8.8.8)

At this stage, the remote Server (8.8.8.8.8) got it's "ICMP migrate" command.
It must look if it has this socket opened, and compare the expected TCP sequence number. If there is a match it will rebind it's TCP socket from:

source.port = 6050, source.IP = 525.8.78.2.2,
destination.port = 80, destination.IP = 8.8.8.8.8
to:
source.port = 4128, source.IP = 775.2.23.5.5,
destination.port = 80, destination.IP = 8.8.8.8.8

Keeping the TCP session open ...and sending new data to this new subnet !!!

NOTE: this can happen, even if the client disconnected and reconnected later.

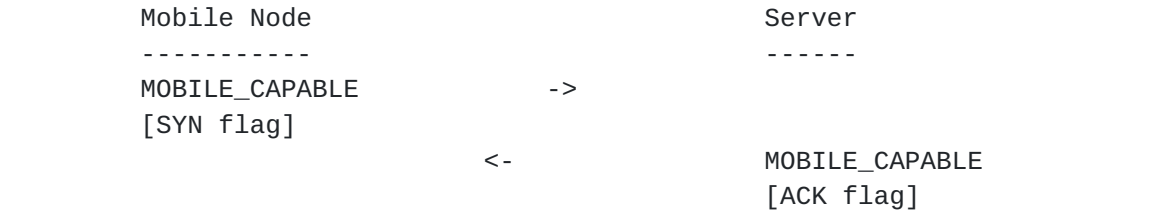
Server will send either:
-"ICMP migration success!" (with the "ICMP migrate" message parameters)
-or-
-"ICMP migration failed." (with the "ICMP migrate" message parameters)

NATs must intercept this command, and send a modified "ICMP migration success!/failed." acknowledgement.

Appendix C: Future ideas

x.y. Initiating a Mobile TCP Session

This is the same signaling as for initiating a normal TCP connection, but the SYN, SYN/ACK, and ACK packets also carry the MOBILE_CAPABLE option.



```
   ACK + MOBILE_CAPABLE      ->
   [SYN+ACK flags]
```

TCP option number = (?)

This will allow middleboxes to be aware of Mobile TCP session,
to increase their timeouts, which is important for NATs & Firewalls.

This will also allow NAT routers to send "ICMP NAT acknowledgements"
only if a TCP session has "MOBILE_CAPABLE" flag, reducing noise
traffic in corporate environment.

Author Contacts

Alexey Eromenko
Israel

Skype: Fenix_NBK_
EMail: al4321@gmail.com

INTERNET-DRAFT
Alexey
expiration date: 2016-06-10