(draft)

Socket (Session) Aware Change of IP – SACIP

Samo Pogačnik (samo_pogacnik@t-2.net),
Škofja Loka
September, 2008

Abstract:

This paper presents one possibility to preserve established network connections of unmodified applications, when an IP network address of an endpoint device changes. This functionality might enable the mobility of network users in a way, where a mobile user could cross borders of staticaly configured IP (sub)network areas just by accepting new IP address available from the entered IP network. At the same time established network connections would not be interrupted and no need for the network infrastructure routing reconfiguration on behalf of the mobile user's changed location, would have been necessary.

The basic idea of SACIP is to manipulate IP addresses written into IP packet headers dynamically as the IP address of a local or remote endpoint change. This manipulation must not break connection associated data of the connections established at the socket level. For this purpose extended socket parameters are being filled with the new IP address values as well as remote hosts are being notified about the change. A special notification protocol is necessary for this purpose and which mostly defines the security aspect of the solution. This presentation includes a description of the primitive implementation of SACIP functionality in Linux and its possible enhancements and use cases.

Table of content

1 Introduction....3
2 Functional limitations and requirements....4
3 SACIP implementation....5
3.1 IPv4 network socket extensions and their initialization....5
3.2 Usage of the extension fields....7
3.3 SACIP notification protocol....8
3.4 SACIP activation....9
4 Performances....10
5 Security 10
6 Conclusion....10
7 References....11

1 Introduction

The main motivation for this experiment, is the increasing need for the on-line mobility of user network devices (mobile phones, laptops, hand helds, ...), as well as IP becoming the “de facto” platform for any kind of communication. The aim of this work is to explore possibilities (as well as to implement at least minimal demonstration in Linux) for preserving network connections over the IP network, while one or both connection endpoints change their IP address.

There are several reasons for changing IP address of a mobile endpoint device. A user could be physically crossing borders of two or more IP subnets. And there could be cases where a user wants to switch between more concurrently accessible IP networks on the same or different network interfaces (a device might support more access technologies for instance). Beside those cases, it can happen, that a device (interface) gets new address within the same IP subnet as well (i.e. DHCP lease expiration). To achieve true mobility of IP endpoint devices, breaking connections would not allways be acceptable, if IP addresses change. Preserving network connections of a mobile user is a different topic than using the same IP address independently of the users connection point in the IP network (defined as Mobile IP – RFC 2002). Allthow theese are two different things, they might be complementary.

The idea for this experiment rises from the fact that a network using IP, which is unreliable and connectionless oriented protocol of the network layer, delivers packets from one endpoint to another, independently of the upper protocol layers (which define connections). *So, there must exist the possibility to preserve already established network connections, if a kind of notification protocol between communicating endpoints about the IP address changes, would have been implemented*. As a result, IP addresses of IP packets of live connections could adapt in sync with the IP address changes. Additionally, attention must be paied towards how layers above the network layer use IP addresses to preserve the connection.

This means that in case of established network connections based on connected `inet` sockets, three things need to be done:

1. Extension of the original IP address fields of `inet` socket structure by additional fields for local and remote IP addresses, referring to current and next active values.
2. The original local and remote IP address parameters to stay bound to the upper socket layer only. Their use in the IP layer is being replaced by currently active values from newly added fields. On IP address change only IP layer bound parameters of the extended socket get changed. This means that IP layer bound addresses are being used for the routing purposes of the outgoingpacket, being written into IP headers of outgoing packets as well as being used do find corresponding socket of the incoming packet. On the other hand original addressed contribute to TCP/UDP checksum calculation, for instance.
3. On IP address change, notification message about changed IP has to be sent to every destination pointed by established socket related to the changed IP address. On receip of the notification message, related local sockets must be updated with the received value as the new current remote IP address.

2 Functional limitations and requirements

To be able to set a frame around the implementation of this experiment, a few limitations and requirements regarding the functionality, have been specified:

1. No application modifications would have been made. That way only connections defined in the socket layer could be preserved. To be able to address connections defined above the socket layer in each application, additional notification mechanism towards application is needed.
2. The security issues as well as the reliability of the notification protocol are going to be overlooked for the sake of the simplicity of the implementation. Concerns about the definition of the notification protocol mostly relate to security aspects of transferring changed (and the original) IP addresses over the network. Otherwise, notification has to be as simple as possible, to reduce impact on latency critical connections (such as phone conversations). This requirement also contradicts the requirement for reliability of notification.
3. Only IP address changes on the same network interface, which is being used for communication, are being covered by this implementation. That way existing networking functionality (promote secondaries) and IP configuration tools could be used to test SACIP.
4. Only simple network configurations without NAT devices in the connection path are being covered. Today's IPv4 network consists of a complex mixture of public and private sub networks. To preserve connections through a combination of public and private sub networks, to many migration scenarios would have to covered straight from the beginning. Therefore, limiting this research to IP changes of an endpoint device only within a single private or just a public network, is necessary.
5. Only IPv4 would have been covered.
6. Lose of current IP address is not allowed before the notification message would have been sent. The effect of this requirement was that only new address has to be specified in the payload of the notification message. This is again a simplification for the sake of the implementation and also means that a user must always enter new subnet through an area where both subnets would be accessible.

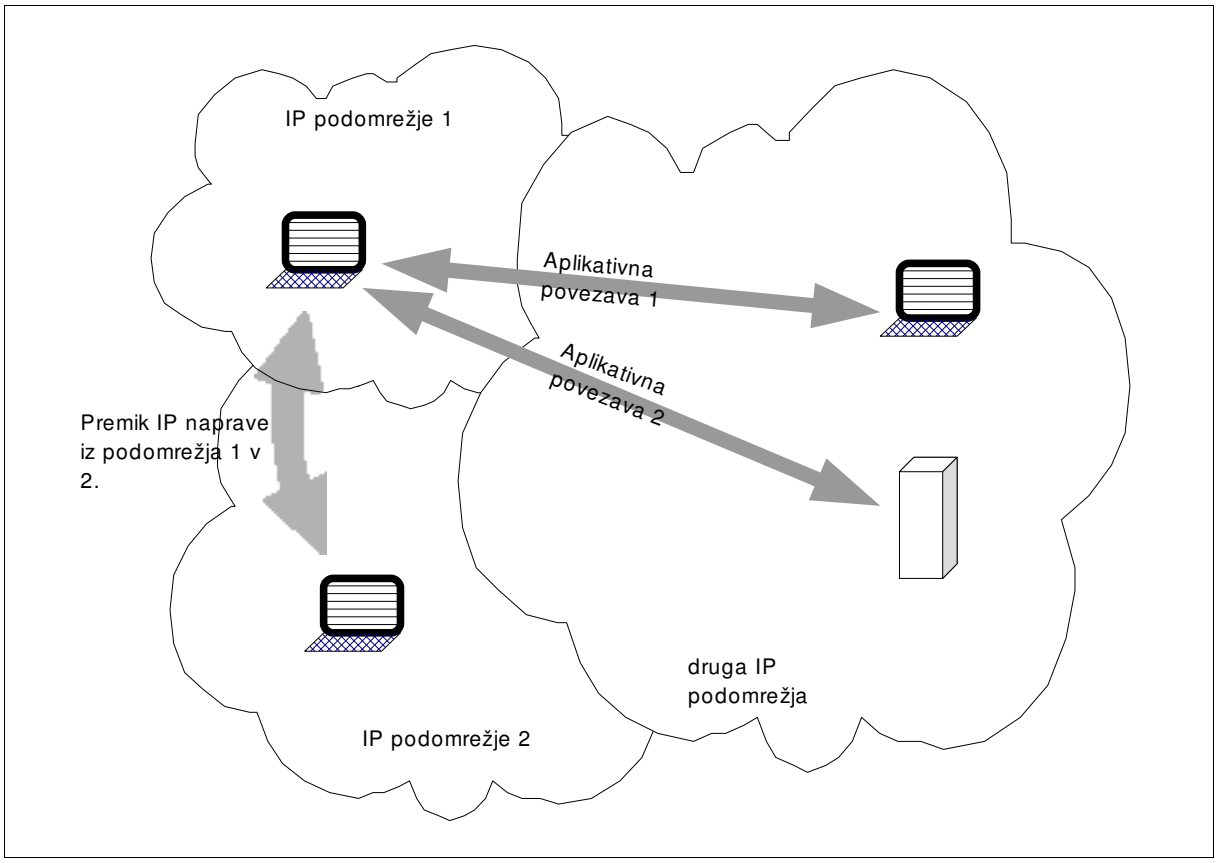

Figure 1 - Simple migration of a mobile IP endpoint device.

3 SACIP implementation

This section explains the implementation of the SACIP functionality within the 2.6.19 Linux kernel.

3.1 IPv4 network socket extensions and their initialization

The structure of the IPv4 network socket needs additional fields:

- a pair of additional source IP addresses
- a pair of additional destination IP address
- an index of the currently active source IP address within the added source address pair
- and an index of the currently active destination IP address within the added destination address pair

The implementation detail of a socket structure extension:

```
--- linux-2.6.19/include/net/inet_sock.h
+++ linux-2.6.19-sacip/include/net/inet_sock.h
@@ -112,6 +112,12 @@ struct inet_sock {
      /* Socket demultiplex comparisons on incoming packets. */
      __be32                  daddr;
      __be32                  rcv_saddr;
+#ifdef CONFIG_SACIP
+     __be32                  sac_daddr[2];
+     int                     sac_daddr_act;
+     __be32                  sac_rcv_saddr[2];
+     int                     sac_rcv_saddr_act;
+#endif
      __be16                  dport;
      __u16                   num;
      __be32                  saddr;
```

A pair of additional source and destination IP addresses have been specified for the following reasons:

1. Original IP addresses bound to the upper applicative layers stay untouched during the whole life of a connection as they were set at the connection setup. This way application does not notice the change in IP addresses used to carry connection data.
2. One additional IP address holds currently active source and destination IP address. Those addresses are being used within the network layer (IP layer).
3. The other additional IP addresses are being prepared for the new source or destination address during the IP change at one end of the connection.

All additional IP addresses get initialized to the same values as original IP address fields at connection setup or any other action that explicitly initialize original IP address fields. At the initialization indexes of the active addresses always point to the first address of a pair.

Extra helper functions are being prepared for the manipulation of theese extensions:

```
--- linux-2.6.19/include/net/inet_sock.h
+++ linux-2.6.19-sacip/include/net/inet_sock.h
@@ -150,6 +156,54 @@ static inline struct inet_sock *inet_sk(
      return (struct inet_sock *)sk;
 }

+#ifdef CONFIG_SACIP
+static inline __be32 sac_inet_daddr(const struct sock *sk)
+{
+     struct inet_sock *inet = inet_sk(sk);
+     return inet->sac_daddr[inet->sac_daddr_act];
+}
+
+static inline __be32 sac_inet_rcv_saddr(const struct sock *sk)
+{
+     struct inet_sock *inet = inet_sk(sk);
+     return inet->sac_rcv_saddr[inet->sac_rcv_saddr_act];
+}
+
+static __inline__ void sac_init_daddr(struct inet_sock *sk, __be32 daddr0,
__be32 daddr1, int act)
+{
+     sk->sac_daddr[0] = daddr0;
```

```
+     sk->sac_daddr[1] = daddr1;
+     sk->sac_daddr_act = act;
+}
+
+static __inline__ void sac_init_rcv_saddr(struct inet_sock *sk, __be32 saddr0,
__be32 saddr1, int act)
+{
+     sk->sac_rcv_saddr[0] = saddr0;
+     sk->sac_rcv_saddr[1] = saddr1;
+     sk->sac_rcv_saddr_act = act;
+}
+
+static __inline__ void sac_add_rcv_saddr(struct inet_sock *sk, __be32 saddr)
+{
+     sk->sac_rcv_saddr[(sk->sac_rcv_saddr_act + 1) % 2] = saddr;
+}
+
+static __inline__ void sac_act_rcv_saddr(struct inet_sock *sk)
+{
+     sk->sac_rcv_saddr_act = (sk->sac_rcv_saddr_act + 1) % 2;
+}
+
+static __inline__ void sac_add_daddr(struct inet_sock *sk, __be32 daddr)
+{
+     sk->sac_daddr[(sk->sac_daddr_act + 1) % 2] = daddr;
+}
+
+static __inline__ void sac_act_daddr(struct inet_sock *sk)
+{
+     sk->sac_daddr_act = (sk->sac_daddr_act + 1) % 2;
+}
+#endif
+
 static inline void __inet_sk_copy_descendant(struct sock *sk_to,
```

3.2 Usage of the extension fields

For the incoming packets, extension fields are being used as a glue between the network and transport layer to find sockets related to incoming packets, in the same way as original IP addresses did:

```
--- linux-2.6.19/include/net/inet_hashtables.h
+++ linux-2.6.19-sacip/include/net/inet_hashtables.h
@@ -325,6 +325,7 @@ typedef __u64 __bitwise __addrpair;
       (!((__sk)->sk_bound_dev_if) || ((__sk)->sk_bound_dev_if == (__dif))))
 #else /* 32-bit arch */
 #define INET_ADDR_COOKIE(__name, __saddr, __daddr)
+#ifndef CONFIG_SACIP
 #define INET_MATCH(__sk, __hash, __cookie, __saddr, __daddr, __ports, __dif)
      \
      (((__sk)->sk_hash == (__hash))                          &&     \
       (inet_sk(__sk)->daddr        == (__saddr))             &&     \
@@ -337,6 +338,20 @@ typedef __u64 __bitwise __addrpair;
       (inet_twsk(__sk)->tw_rcv_saddr     == (__daddr))           &&     \
       ((*((__portpair *)&(inet_twsk(__sk)->tw_dport))) == (__ports)) &&     \
       (!((__sk)->sk_bound_dev_if) || ((__sk)->sk_bound_dev_if == (__dif))))
+#else
+#define INET_MATCH(__sk, __hash, __cookie, __saddr, __daddr, __ports, __dif) \
```

```
+     (((__sk)->sk_hash == (__hash)) && \
+      (sac_inet_daddr(__sk) == (__saddr)) && \
+      (sac_inet_rcv_saddr(__sk) == (__daddr)) && \
+      ((*((__portpair *)&(inet_sk(__sk)->dport))) == (__ports)) && \
+      (!((__sk)->sk_bound_dev_if) || ((__sk)->sk_bound_dev_if == (__dif))))
+#define INET_TW_MATCH(__sk, __hash,__cookie, __saddr, __daddr, __ports,
__dif) \
+     (((__sk)->sk_hash == (__hash)) && \
+      (sac_inet_tw_daddr(__sk) == (__saddr)) && \
+      (sac_inet_tw_rcv_saddr(__sk) == (__saddr)) && \
+      ((*((__portpair *)&(inet_twsk(__sk)->tw_dport))) == (__ports)) && \
+      (!((__sk)->sk_bound_dev_if) || ((__sk)->sk_bound_dev_if == (__dif))))
+#endif
 #endif /* 64-bit arch */
```

They are being used within the network layer as well. Instead of the original fields, extended fields define source and destination IP addresses of outgoing packets:

```
--- linux-2.6.19/net/ipv4/ip_output.c
+++ linux-2.6.19-sacip/net/ipv4/ip_output.c
@@ -309,7 +309,11 @@ int ip_queue_xmit(struct sk_buff *skb, i
      __be32 daddr;

      /* Use correct destination address if we have options. */
+#ifndef CONFIG_SACIP
      daddr = inet->daddr;
+#else
+     daddr = inet->sac_daddr[inet->sac_daddr_act];
+#endif
      if(opt && opt->srr)
            daddr = opt->faddr;

@@ -317,7 +321,11 @@ int ip_queue_xmit(struct sk_buff *skb, i
            struct flowi fl = { .oif = sk->sk_bound_dev_if,
                                .nl_u = { .ip4_u =
                                        { .daddr = daddr,
+#ifndef CONFIG_SACIP
                                        .saddr = inet->saddr,
+#else
+                                       .saddr = inet->sac_rcv_saddr[inet->
sac_rcv_saddr_act],
+#endif
                                        .tos = RT_CONN_FLAGS(sk) } },
                                .proto = sk->sk_protocol,
                                .uli_u = { .ports =
```

3.3 SACIP notification protocol

As already mentioned, simplicity of the notification is the main goal of this specification (no acknowledgement messages, no message authentication). It is merely specified to test the SACIP concept.

Notification message has been sent for each socket found associated with the changed IP address of the endpoint device. It is defined as an additional ICMP message (temporary definition: type 20?). ICMP data field contains only new IP address to be used for the connections bound to the endpoint device sending the notification message. This way we can not afford to loose original address

before sending the notification, but connection is at least a little bit safer.

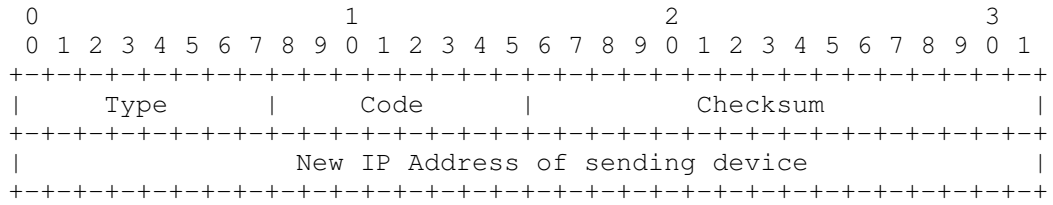

Figure 2 - SACIP notification ICMP message

The device which receives this notification message scans its sockets and reconfigures its socket extensions so that all packets sent for the affected connections use new destination IP address.

3.4 SACIP activation

No additional configuration tool has been implemented for this, so features of the *ip* tool (from *iproute2* package) had been used. First by adding a new address to existing interface and then by deletion of the first primary address, sacip functionality gets called. And if new address has been added as a secondary address, then the promote secondaries kernel feature has to be enabled as well to activate sacip functionality by deletion of the primary address.

After the deletion of the first primary address, all tcp established sockets as well as all udp and raw sockets are checked, if their current active source address correspond to old/deleted ip address. For each socket found an ICMP message containing only new primary address is sent to the currently active destination address of a socket. After that new source addres get assigned as currently active on a local endpoint.

On receipt of a sacip ICMP message, remote side also checks its sockets, if their current active destination address equals to the source address of the ip packet header carrying the sacip icmp message. Found sockets get assigned new ip address (received in the icmp message) as current active destination address, afterwards.

Afterwards communication continues using newly assigned ip address, but the application (telnet for instance) operates as if old address would still be assigned.

An excerpt of the activation sequence where a primary IP address gets deleted and additional addresses promoted:

```
--- linux-2.6.19/net/ipv4/devinet.c
+++ linux-2.6.19-sacip/net/ipv4/devinet.c
@@ -282,6 +288,20 @@ static void __inet_del_ifa(struct in_dev
                        break;
                }
        }
+#ifdef CONFIG_SACIP
+       if (promote) {
+/*samo - tst:            printk("__inet_del_ifa: 1: ifa1->ifa_local=0x%x,
promote->ifa_local=0x%x\n", ifa1->ifa_local, promote->ifa_local);*/
+               sac_add_rcv_saddr_tcp(ifa1->ifa_local, promote->ifa_local);
+               sac_add_rcv_saddr_udp(ifa1->ifa_local, promote->ifa_local);
```

```
+                 sac_add_rcv_saddr_raw(ifa1->ifa_local, promote->ifa_local);
+           } else
+                 if (prev_prom) {
+/*samo - tst:               printk("__inet_del_ifa: 2: ifa1->ifa_local=0x%x,
prev_prom->ifa_local=0x%x\n", ifa1->ifa_local, prev_prom->ifa_local);*/
+                       sac_add_rcv_saddr_tcp(ifa1->ifa_local, prev_prom->
ifa_local);
+                       sac_add_rcv_saddr_udp(ifa1->ifa_local, prev_prom->
ifa_local);
+                       sac_add_rcv_saddr_raw(ifa1->ifa_local, prev_prom->
ifa_local);
+                 }
+#endif
      }

      /* 2. Unlink it */
```

4 Performances

Regarding performances, special care has to be taken within the code handling every incoming or outgoing packet.

5 Security

Security of connections following this SACIP is poor, because a single spoofed SACIP notification message containing invalid new IP address breaks all connections originated from one IP address. If some secrets could be safely exchanged at the beginning of the connection (I propose IKE protocol from the IPSec), encryption of the notificaion message would be possible. Then this kind of connection intrusion would be much harder. Other security aspects of the connection should note get worse because of SACIP.

6 Conclusion

I am not sure how valuable this functionality could be, but *mobility* is the future (they say). So any research in this direction might pay-off.

And for the future, elimination of imposed limitations would be nice:

- application notification, which must not affect existing applications,
- secure notification,
- notification carying original and new IP address to be able to skip gaps without accesible network,
- extra tool for SACIP activation,
- SACIP activation over multiple network interfaces,
- how to include NAT devices into the game,
- Any comments are most welcome, also to make this document more understandable.

7 References

[1] RFC 791, Internet Protocol, 1981

[2] RFC 793, Transmission Control Protocol, 1981

[3] RFC 768, User Datagram Protocol, 1980

[4] RFC 792, Internet Control Message Protocol, 1981

[5] RFC 854, Telnet Protocol, 1983

[6] Internet sockets, http://en.wikipedia.org/wiki/Internet_socket