Implementation of SCTP Mobility in The Linux Kernel

Michio Honda
Keio University †
micchie@sfc.wide.ad.jp

Hideyuki Tokuda
Keio University/JST-CREST †
hxt@ht.sfc.keio.ac.jp

Abstract—In this work we implement mobility features of Stream Control Transmission Protocol (SCTP) in the Linux kernel. SCTP is a transport protocol that supports many features including native multi-homing and dynamic address reconfiguration in addition to TCP's features. The dynamic address reconfiguration allows the transport layer connection to be maintained when the host reconfigures its IP address. It thus supports host mobility between different network segments. When a host adds or deletes an IP address, the SCTP endpoint sends an ASCONF packet to the remote endpoint to tell the address reconfiguration. Auto-ASCONF allows the SCTP endpoint to negotiate ASCONFs automatically upon the address event without explicit operation of the application, which is important for mobility support without applications changed. We imported the basic Auto-ASCONF implementation into the net-next-2.6 tree of the Linux kernel.

Keywords: mobility, SCTP, implementation, Auto-ASCONF

I. INTRODUCTION

SCTP is a transport protocol that supports advanced features, such as native multi-homing, multi-streaming and message-oriented transmissions in addition to TCP’s features [1]. Dynamic address reconfiguration in the existing end-to-end connection is standardized as RFC5061 [2], which supports host mobility between different networks. When a host adds or deletes an IP address, it sends an ASCONF packet to the remote endpoint to arrange the address reconfiguration in the end-to-end SCTP association.

This ASCONF negotiation is invoked by two ways; application’s operation and automatic operation. The former is that the application binds another address manually to the socket via sctp_bindx() system call. However, this is inconvenient for applications, because they need to implement address event detection by themselves (e.g., via a routing socket) in addition to calling sctp_bindx(). In the latter way, ASCONF negotiation is done within the SCTP implementation, so-called Auto-ASCONF. With Auto-ASCONF applications do not need to be modified for mobility support.

However, there are many technical challenges to implement mobility features including Auto-ASCONF in the Linux SCTP implementation, such as in address event handling, socket operations, and congestion control. In address event handling, we have to care multiple notifications for one event, address scope, and actual address state (e.g., duplicate address detection).

In socket operations, Auto-ASCONF implementation must manage SCTP connections appropriately, deciding if it applies Auto-ASCONF operation depending on the socket option, the sysctl parameter, remote implementation and the bind address type. In congestion control, SCTP specification describes that congestion control is managed based on the destination address, rather than source-destination address pair. We have to change this behavior to perform congestion control properly. In this paper, we describe implementation of mobility feature of SCTP, which includes Auto-ASCONF and the other necessary functionalities. We recently merged these implementations to the net-next-2.6 kernel tree [3] that is constantly merged into the main-line tree.

II. ADDRESS EVENT HANDLING

When an IP address becomes available or unavailable, an address event is notified to the SCTP implementation. The straightforward implementation for Auto-ASCONF would transmit an ASCONF on every SCTP socket every time an address event occurs. However, there are two issues to handle the address event properly.

First, multiple notifications for an address event must be processed correctly. For example in DHCP, an address event that adds an IP address, one that deletes the same address, and one that adds this address again occur when an IP address is configured to the host. In order to avoid an ASCONF sent every time an address event happens, we implement a queue to accommodate address events and a timer to process address event on each socket.

Suppose an address event adding an IP address happened, it was enqueued, and the timer has been started. The period to expire is heuristically 500 ms by default. When another address event deleting this IP address happens before the timer expires, the SCTP implementation checks whether an opposite address event for the same address exists in the queue. If it is found as in this case, the queue offsets the latter address event against the existing opposite one. If not, the SCTP implementation simply enqueues the address event. When the timer expires, the SCTP implementation picks address events from the queue, and operates corresponding ASCONF packets for each socket.

Second, we must handle the IPv6 address during duplicate address detection (DAD) procedure. Due to the implemen-
tation of the address event notification, it notifies the SCTP implementation of the new address during DAD. Hence, we have to wait for sending the ASCONF until the DAD completes. In our implementation, we add the DAD address to the address event queue. When the timer for the address event queue expires, the SCTP implementation restarts the timer without processing the DAD address, if the DAD has not been completed yet. Hence, our implementation operates ASCONFs without trying to add the IPv6 address during DAD.

### III. Socket Operation

The SCTP implementation must apply the address event to the each existing socket properly. We provide two ways to enable Auto-ASCONF. First, it is enabled by a `sysctl` parameter `default_auto_asconf`. This is set by the system administrator, as it determines system-wide SCTP behavior. Second, it is enabled by a `setsockopt()` with a `SCTP_AUTO_ASCONF` parameter as defined in the Internet Draft [4]. This definitely affects the corresponding socket, which overwrites the `sysctl` parameter. On the other hand, [4] suggests that the `sysctl` parameter would be able to disable Auto-ASCONF regardless of the `setsockopt()` operation. However, we do not adopt this choice. We believe it makes sense that the application can locally enable and disable Auto-ASCONF. Depending on the future requirements, we will implement another `sysctl` parameter to disable Auto-ASCONF regardless of the `default_auto_asconf` `sysctl` parameter and `setsockopt()` with `SCTP_AUTO_ASCONF`.

In order to apply address events to existing SCTP sockets easily, we implement the Auto-ASCONF socket list. When the `default_auto_asconf` is true, the SCTP implementation adds the socket to this list at the initiation of the socket. It removes this socket from the Auto-ASCONF socket list if the application calls `setsockopt()` to disable Auto-ASCONF. When the `default_auto_asconf` is false, the SCTP implementation adds the socket to this list only when it calls `setsockopt()` to enable Auto-ASCONF. Hence, when the SCTP implementation applies the address event to existing sockets, it can simply walk through this list and operates ASCONF packets.

On the other hand, when the application binds specific address to the socket with the `bind()` system call rather than binds wildcard address (i.e., `INADDR_ANY` in IPv4), Auto-ASCONF cannot be enabled [4]. Hence, we filter out such sockets when we operate ASCONF packets through the Auto-ASCONF socket list. In addition, when the SCTP implementation applies an address event in the socket, it must check the address scope. For example, if the remote endpoint has the only global addresses, the local SCTP implementation cannot add private addresses to the association, thus it should not send ASCONF packets for such addresses.

### IV. Last Address Handling and Congestion Control

As described in [2], the SCTP endpoint cannot remove the last address in the association. This limitation is obvious, otherwise the association becomes the state that does not have any address. However, this situation easily occurs in the mobile. When the single-homed host moves from network A to network B, it will delete the address of network A (address A), and will configure an address of network B (address B). In this situation, the application wishes to remove the address A after the address B is configured. However, it is not achieved if we process the address event sequentially, because the deletion of the address A fails.

To delete the address A, we implement delayed processing of the last address deletion. On the address event deleting the address A, the SCTP implementation maintains this address in the control block. When the address B is configured, the SCTP implementation creates the ASCONF for the address B, and then piggybacks the deletion of the address A in this ASCONF packet. This is allowed in the specification as far as the message to delete the address A is placed after the message to add the address B.

Congestion control is also an issue to implement mobility features in SCTP. SCTP performs congestion control per destination address rather than source-destination address pair in its principle. However, it does not make sense to keep this criteria in mobility scenarios, because path conditions change before and after the host migration [5]. The wireless network where the mobile host is attached is bottleneck in most of the cases, thus path condition should be re-probed when using the new address.

From this reason we modify congestion control in the mobile node side. We reset congestion control parameters, such as the congestion window and the retransmission timeout (RTO) value when the host reconfigures the last address, as the data transmission path obviously changes.

### V. Conclusion and Future Work

In this paper, we presented mobility implementation of Linux SCTP that is recently merged in to net-next-2.6 tree. We are now submitting a patch for the fast handoff feature proposed in [5], which minimize the idle period of data transmissions after the connectivity revival.

### Acknowledgment

We especially thank Wei Yongjun, who thoroughly reviewed and commented on our patches submitted to the net-next-2.6 mailing list. We also thank Jamal Hadi-Salim, Yoshihiji Hideaki and David Miller who assisted our patch submission.

### References