OSPF Protocol 표준 및 구현의 대수 정형적 명세 및 정형적 검증

정회원 박 재 현*

Algebraic Formal Specification and Formal Validation of the Standard and an Implementation of the OSPF Protocol

Jae-Hyun Park* *Regular Members*

요 약

OSPF 프로토콜은 가장 보편적으로 사용되고 있는 Interior Gateway Routing Protocol 이다. 그러므로, 초고속 라우터의 동작의 신뢰성 확보를 위해, OSPF 프로토콜의 상호운용성과 안전성의 보장은 중요하다. 본 논문에서는 OSPF 라우팅 프로토콜의 표준 문서를 분석하여, 프로토콜 동작들을 대수 정형 방법을 사용하여 상세 설계 수준으로 명세하고, OSPF 프로토콜의 가용한 원천코드를 참조하여, 표준에 명확하게 명세되지 않은 좀더 상세한 동작까지 표현하도록 이 명세를 보완하였다. 그리고 명세된 프로토콜 상태 기계의 상호운용성과 안전성을 정형적으로 증명하였다. 우리는 먼저 제시한 정형적 명세가 OSPF 프로토콜의 표준 문서에 나타난 모든 상태들과 모든 천이 이벤트들을 표현하고 있어, 명세된 프로토콜 상태기계가 완전성을 가지고 있다는 사실을 보이고, 또한 상호운용성을 가짐을 증명하였다. 그리고 프로토콜의 동작의 안전성을 증명하기 위해, 생존성, 그리고 라이브락 부재를 정형적으로 증명하고, 교착상태 부재 역시 정형적으로 증명하였다. 결과적으로 OSPF 프로토콜이 일관성을 가진다는 사실을 보인다. 본 명세와 검증은 기존의 OSPF Version 2의 프로토콜 메커니즘을 그대로 계승한 OSPF Version 3에서도 유효하다.

Key Words : Protocol Validation, OSPF Routing Protocol, Formal Method, Label Transition System

ABSTRACT

The OSPF protocol is the most widely used Interior Gateway Routing Protocol. Therefore, for the reliability of behavior of gigabit swiching routers, it is essential to guarantee the interoperability and the safety of the OSPF protocol. In this paper, we analyze the standard document of the OSPF protocol, so that we provide a formal specification that specifies the protocol behaviors by detailed design level using the algebraic formal method. By referring available source codes of the OSPF protocol, we supplement the formal specification to express more detailed behaviors that is not specified definitely in the standard. We also formally verify the interoperability and the safety of the protocol state machine of the specification. By showing that the formal specification specify all of the states and the transition events that appear in the standard document of the OSPF protocol, we prove that the state machine has the completeness, and prove it has the interoperability. To prove that the specification of the protocol has the safety, we formally verify the reachability, the liveness, the livelock-free property, and the deadlock-free property. As a result, we prove the protocol has the consistency. The specification and the validation are also effective to the OSPF Version 3 that inherit the protocol mechanism of the OSPF Version 2.

Ⅰ. 서 론

인터넷이라고 하는, 고속 공중 데이터 망을 구성하기위해 사용되는 교환기인 초고속 라우터를 개발

* 중앙대학교 컴퓨터공학부 초고속통신 연구실(hyunie@cau.ac.kr)
논문번호:030087-0306, 접수일자: 2003년 X월 X일
※본 연구는 첨단정보기술 연구센터를 통하여 과학재단의 지원을 받았습니다.

하기 위해서, 우리는 OSPF 프로토콜을 포함한 라우터 시스템의 소프트웨어를 개발하여야 한다. OSPF 프로토콜은 Interior Gateway Routing Protocol로서 널리 사용되고 있고, 또한 개발 측면에서 볼 때 라우팅 시스템의 소프트웨어 중 가장 복잡하기 때문에, 이 OSPF 프로토콜이라 하는 경로 설정 프로토콜의 안정성과 상호 운용성을 확보하는 것이, 라우터를 상용화하기 위한 중요한 요건 이다.

라우팅 프로토콜의 정형적 검증을 위한 대표적인 연구들로는 다음과 같은 연구들이 있었다. OSPF 프로토콜 표준문서[1]에 명시된 상태기계의 FULL 상태에서, Link State Update Packet에 담아 보내는, Link State Advertisements의 동기화의 이론적 정당성을 연구한, 비동기 데이터 네트웍 상에서의 정보의 동기화 연구[2]와 Hello Protocol의 2WAY 상태와 INIT 상태[1]에서 Designated Router를 선출하는 알고리즘의 이론적 정당성을 보인 지도자 선출 연구[3]에서 OSPF 프로토콜의 동작의 기반이 되는 부속 프로토콜들은 정형적으로 증명되어 있다. ATM의 PNNI의 라우팅 프로토콜의 경우에도 위의 두가지 쟁점들을 모델 검사기(Model Checker)인 SPIN[4]을 사용해서 자동 검증한 연구[5]가 있다. 그리고 라우팅 프로토콜인 RIP의 수렴성을 자동 검증 도구들을 사용하여 검증한 연구[6]가 있다. 그러나 OSPF 프로토콜 표준 문서의 전체 명세에 해당하는, 프로토콜 상태 기계 인 이웃 상태 기계[1]를 정형적으로 검증한 연구는 없었다.

더구나, OSPF 프로토콜의 소프트웨어는 GNU General Public License로 원천 코드가 공개되어 있다. 따라서 상용 라우터의 시제품의 구현을 위해 필요한 OSPF 프로토콜 소프트웨어를 확보하기는 쉽다. 그러나 OSPF 프로토콜은 IETF에 의해 정의된 프로토콜 중 가장 복잡하고, 표준 문서의 프로토콜 상태 기계도 직관적으로 문장들로 기술되어 있어, 상태 기계를 이해하기 어렵고, 신뢰성에 문제가 발생할 경우, 그 해결이 어렵다. 따라서 정형적 방법을 사용한 검증이 필수적이라고 하겠다.

본 논문에서는 상용 라우터에 사용할 수 있는 OSPF 프로토콜이 가져야 하는 최소한 특성을 다음 두가지 사항으로 정리하고, 이들을 검증하였다.

- 표준의 일치: 기존의 시장을 선점한 제품과의 상호 연동성을 확보하기 위해, OSPF 프로토콜의 외부 프로토콜 동작들은 IETF의 RFC2328[1]을 따라야 한다.
- 신뢰성: OSPF 프로토콜의 소프트웨어 Carrier Class 제품의 신뢰성을 지녀야 한다.

본 논문에서는, 상기한 두가지 특성들을 보장하는, OSPF Protocol을 확보하기 위해서, OSPF 표준 문서에서 직관적으로 문장들로 기술되어 있는 프로토콜 상태 기계를 분석하고, 구현되어 있는 OSPF 프로토콜 소프트웨어의 원천 코드[11]를 참조하여, 상태 기계를 상세 설계 수준까지, 프로세스 대수[7,8,9,10]를 사용하여, 정형적으로 명세한다. 그리고, 이 명세를 분석하여, 이 정형적 명세가 표준 문서의 프로토콜 상태 기계와 완전히 일치함을 보여, 제시한 정형적 명세가 완전성 (Completeness)을 지님을 보이고, 상호 운용성을 가짐을 증명한다. 그리고 명세의 초기 상태들로부터 각 상태들로 도달 가능함을 의미하는 도달성 (Reachability)을 정형적으로 증명하고, 모든 상태들로부터 초기상태가 도달 가능하다는 것을 의미하는 생존성(Liveness)을 정형적으로 증명한다. 그리고 명세의 모든 프로토콜 동작들의 폐흐름 (Loop)들이 모두 생산적 폐흐름임을 보여, 라이브락 부재 (Livelock-Free) 특성을 증명하고, 교착상태 부재 (Deadlock-Free) 특성을 정형적으로 검증한다. 결과적으로 우리는 OSPF 프로토콜의 명세[1]와 상세 내부 상태들과 동작들의 명세를 위해 참조한 OSPF 소프트웨어의 원천 코드[11]의 안전성 (Safety)과 일관성 (Consistency)를 분석적으로 검증한다. 이들 작업들을 통해 OSPF 프로토콜의 동작을 좀더 깊이 이해하고, 더나아가 프로토콜의 신뢰성 보완 기술을 확보한다.

본 논문의 구성은 다음과 같다. 2절에서, 프로세스 대수를 사용하여, OSPF 프로토콜의 정형적 명세를 제시한다. 3절에서 정형적 명세를 분석하여, OSPF 프로토콜의 정형적 명세의 완전성, 도달성, 생존성, 라이브락 부재성, 교착상태 부재성, 그리고 안전성 및 일관성을 검증한다. 마지막으로 4절에서 결론을 맺는다.

Ⅱ. OSPF 프로토콜 구현의 대수적 정형 명세

본 절에서는 일종의 프로세스 대수 (Process Algebra)인 Labeled Transition System (LTS)[7,8,9,10]이라 불리우는 대수적 정형 기술 기법을 사용하여, OSPF 프로토콜 표준 문서에 직관적으로

명세되어 있는 프로토콜 상태 기계를 이해하기 쉽게 그리고 포괄적으로 명세 한다.

본 기법을 사용하면, 프로토콜을 상태들과 상태들 간의 프로토콜의 행위들에 따른 천이들로써 정의한다. LTS 기호 체계를 간략히 설명하면, 다음과 같다. 상태 방정식 A ≡ a . B 가 의미하는 것은, 상태 A의 프로세스가 프로토콜 행위 a 후에 상태 B의 프로세스로 천이한다는 것을 의미한다. 상태 방정식 A ≡ a . A + b . B는 상태 A가 행위 a 후에 상태 A로 되거나, 행위 b 뒤에 상태 B로 되는 것을 표현한다. 다음에서는, LTS를 사용하여, OSPF 표준 문서에 직관적으로 기술되어 있는 상태 기계를, 구현되어 있는 OSPF 프로토콜 소프트웨어의 원천 코드[11]를 참조하여, 상세 설계 수준까지 정형적으로 명세한다.

정 의 1 LTS를 사용하여 OSPF 프로토콜을, 구현하는 함수 수준의 상세 동작들까지, 대수 정형적으로 명세하면 다음과 같다.

```
DOWN ≡ HelloInterval_expiry . send(hello_pkt) .
    reset_timer_for_HelloInterval . ATTEMPT +
    PollInterval_expiry . send(hello_pkt) .
    reset_timer_for_PollInterval. ATTEMPT +
    recv(hello_pkt) . reset_timer_for_HelloInterval .
    reset_timer_for_PollInterval .
    reset_RouterDead_timer . (valid_hello_pkt . INIT
    + invalid_pkt . DOWN) +
    RouterDeadInterval_expiry .
    clear_retransmission_list . clear_summary_list .
    clear_LS_req_list . DOWN + (KillNbr + LLDown)
    . DOWN

ATTEMPT ≡ recv(hello_pkt) .
    reset_timer_for_HelloInterval .
    reset_timer_for_PollInterval .
    reset_RouterDead_timer . (valid_hello_pkt . INIT
    + invalid_pkt . DOWN) + (KillNbr +
    RouterDeadInterval_expiry + LLDown) . DOWN

INIT ≡ myIPaddress_founded_in_HELLO_pkt .
    (conn_to_pnt_to_pnt_net +
    conn_to_pnt_to_multipnt_net +
    conn_to_virtual_link + (conn_to_broadcast_net
    + conn_to_NBMA_net) .
    (conn_to_Designated_Router +
    conn_to_Backup_Designated_Router)) .
    EXSTART + (not_conn_to_pnt_to_pnt_net .
    not_conn_to_pnt_to_multipnt_net .
    not_conn_to_virtual_link .
    not_conn_to_broadcast_net .
    not_conn_to_NBMA_net +
    (conn_to_broadcast_net + conn_to_NBMA_net)
    . (not_conn_to_Designated_Router .
    not_conn_to_Backup_Designated_Router)) .
    2WAY +
    myIPaddress_not_founded_in_HELLO_pkt . INIT
    + (KillNbr + RouterDeadInterval_expiry +
    LLDown) . DOWN + recv(hello_pkt) .
    reset_timer_for_HelloInterval .
    reset_timer_for_PollInterval .
    reset_RouterDead_timer . valid_hello_pkt . INIT

2WAY ≡ myIPaddress_not_founded_in_HELLO_pkt .
    INIT + (conn_to_pnt_to_pnt_net +
    conn_to_pnt_to_multipnt_net +
    conn_to_virtual_link + (conn_to_broadcast_net
    + conn_to_NBMA_net) .
    (conn_to_Designated_Router +
    conn_to_Backup_Designated_Router)).
    EXSTART + (KillNbr +
    RouterDeadInterval_expiry + LLDown) . DOWN
    + recv(hello_pkt) .
    reset_RouterDeadInterval_timer . valid_hello_pkt
    . (myIPaddress_founded_in_HELLO_pkt . 2WAY
    + myIPaddress_not_founded_in_HELLO_pkt .
    clear_retransmission_list . clear_summary_list .
    clear_LS_req_list . INIT )

EXSTART ≡ Master_slave_negotiated .
    DD_seq_num_exchanged. EXCHNG + (KillNbr +
    RouterDeadInterval_expiry + LLDown) . DOWN
    + (recv(database_description_pkt) .
    SeqNumberMismatch + recv(LS_req_pkt) .
    BadLSReq + recv(Hello_pkt) .
    myIPaddress_not_founded_in_HELLO_pkt) .
    EXSTART + (not_conn_to_pnt_to_pnt_net .
    not_conn_to_pnt_to_multipnt_net .
    not_conn_to_virtual_link .
    not_conn_to_broadcast_net .
    not_conn_to_NBMA_net +
    (conn_to_broadcast_net + conn_to_NBMA_net)
    . (not_conn_to_Designated_Router .
    not_conn_to_Backup_Designated_Router)) .
    clear_retransmission_list . clear_summary_list .
    clear_LS_req_list . INIT + recv(hello_pkt) .
    reset_RouterDeadInterval_timer . valid_hello_pkt
    . (myIPaddress_founded_in_HELLO_pkt .
    EXSTART +
```

```
    myIPaddress_not_founded_in_HELLO_pkt .
    clear_retransmission_list . clear_summary_list .
    clear_LS_req_list . INIT)

EXCHNG ≡ send(data_description_pkt) . EXCHNG
    + no_receive_DD_in_ack .
    retransmission_last_DD . EXCHNG +
    receive_DD_in_ack .
    possibly_add_nb_linkstate_req_list .
    (LS_req_list_empty .FULL +
    not_LS_req_list_empty . send(LS_req_pkt) .
    LOADING) + (KillNbr +
    RouterDeadInterval_expiry + LLDown) . DOWN
    + (recv(database_description_pkt) .
    SeqNumberMismatch + recv(LS_req_pkt) .
    BadLSReq + recv(Hello_pkt) . valid_hello_pkt .
    myIPaddress_not_founded_in_HELLO_pkt)
    .clear_retransmission_list . clear_summary_list .
    clear_LS_req_list . EXSTART +
    (not_conn_to_pnt_to_pnt_net .
    not_conn_to_pnt_to_multipnt_net .
    not_conn_to_virtual_link .
    not_conn_to_broadcast_net .
    not_conn_to_NBMA_net +
    (conn_to_broadcast_net + conn_to_NBMA_net)
    . (not_conn_to_Designated_Router .
    not_conn_to_Backup_Designated_Router)) .
    clear_retransmission_list . clear_summary_list .
    clear_LS_req_list . INIT + recv(hello_pkt) .
    reset_RouterDeadInterval_timer . valid_hello_pkt
    . (myIPaddress_founded_in_HELLO_pkt .
    EXCHNG +
    myIPaddress_not_founded_in_HELLO_pkt .
    clear_retransmission_list . clear_summary_list .
    clear_LS_req_list . INIT)

LOADING ≡ LS_req_list_empty . FULL +
    recv(LS_update) . delete_LS_req_in_list .
    (LS_req_list_empty . FULL +
    not_LS_req_list_empty . send(LS_req_pkt) .
    LOADING ) + req_retrans_time_expire .
    send(LS_req_pkt) . LOADING + (KillNbr +
    RouterDeadInterval_expiry + LLDown) . DOWN
    + (recv(database_description_pkt) .
    SeqNumberMismatch + recv(LS_req_pkt) .
    BadLSReq + recv(Hello_pkt) .
    myIPaddress_not_founded_in_HELLO_pkt) .
    clear_retransmission_list . clear_summary_list .
    clear_LS_req_list . EXSTART + recv(hello_pkt) .
    reset_RouterDeadInterval_timer . LOADING +
    (not_conn_to_pnt_to_pnt_net .
    not_conn_to_pnt_to_multipnt_net .
    not_conn_to_virtual_link .
    not_conn_to_broadcast_net .
    not_conn_to_NBMA_net +
    (conn_to_broadcast_net + conn_to_NBMA_net)
    . (not_conn_to_Designated_Router .
    not_conn_to_Backup_Designated_Router)) .
    clear_retransmission_list . clear_summary_list .
    clear_LS_req_list . INIT + recv(hello_pkt) .
    valid_hello_pkt .
    myIPaddress_founded_in_HELLO_pkt .
    LOADING

FULL ≡ (local_rtr_state_change + LSA_Reflesh +
    Receive_Self_LSA) . ORIGINATE_LSA +
    recv_LS_update . RECV_N_FLOOD_LSA +
    retrans_timer_expire . build_retrans_update .
    flood_out_interfaces . FULL +
    route_calc_timer_expired . full_route_calc .
    updating_route_table . FULL + (KillNbr +
    RouterDeadInterval_expiry + LLDown) . DOWN
    + (recv(database_description_pkt) .
    SeqNumberMismatch + recv(LS_req_pkt) .
    BadLSReq + recv(Hello_pkt) .
    myIPaddress_not_founded_in_HELLO_pkt) .
    clear_retransmission_list . clear_summary_list .
    clear_LS_req_list . EXSTART +
    (not_conn_to_pnt_to_pnt_net .
    not_conn_to_pnt_to_multipnt_net .
    not_conn_to_virtual_link .
    not_conn_to_broadcast_net .
    not_conn_to_NBMA_net +
    (conn_to_broadcast_net + conn_to_NBMA_net)
    . (not_conn_to_Designated_Router .
    not_conn_to_Backup_Designated_Router)) .
    clear_retransmission_list . clear_summary_list .
    clear_LS_req_list . INIT + recv(hello_pkt) .
    reset_RouterDeadInterval_timer . valid_hello_pkt
    . (myIPaddress_founded_in_HELLO_pkt . FULL
    + myIPaddress_not_founded_in_HELLO_pkt .
    clear_retransmission_list . clear_summary_list .
    clear_LS_req_list . INIT)

ORIGINATE_LSA ≡
    interface_type_specific_LSA_orginate.
    Find_current_database_copy_LSA .
    (no_get_next_LS_seq_no . origination_postpone
    . FULL + get_next_LS_seq_no . build_LSA .
    (empty_result . flush_LSA_instead . FULL+
```

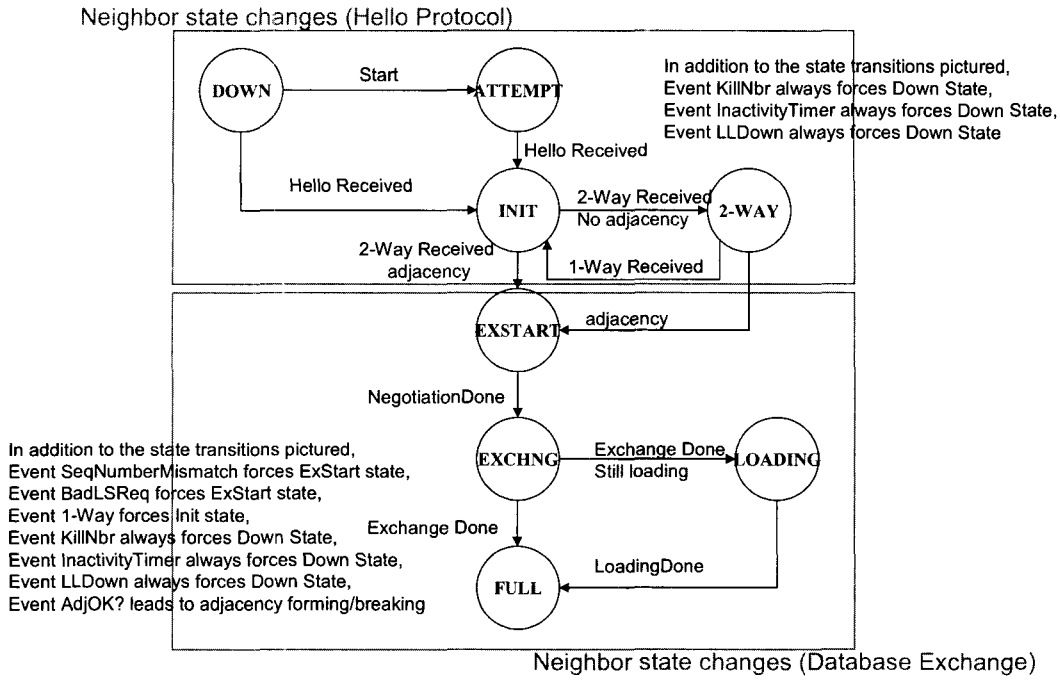

그림 1. OSPF 프로토콜의 이웃 상태 기계 [1]

no_empty_result . (forced + changed) .
checksum . ADD_DATABASE))

ADD_DATABASE ≡ (((localTopologyChanged +
reflesh) .orignateLSA) + receiveLSA) .
add_to_AVL_tree . PreparsingforRoutingCalc .
startaging . add_to_retrans_lists .
flood_out_interfaces .
SCHEDULING_ROUTE_CALC

RECV_N_FLOOD_LSA ≡ LSA_ not_valid .
discard_LSA . FULL + LSA_valid .
(not_more_recent_than_DB . send(dirt_ack_pkt)
. FULL + more_recent_than_DB .
schd_delayed_ACK . (AS_external_LSA_recvd .
EXTER_ROUTING + not_aaAS_external_LSA .
(ADD_DATABASE + DEL_DATABASE)))

DELETE_DATABASE ≡ (overwritten_by_LSA +
maxageLSA_all_acked) . stop_aging .
undo_preparing_for_routing_calc .
remove_from_AVL_tree . no_longer_referenced
. free_LSA_Storage.
SCHEDULING_ROUTE_CALC

SCHEDULING_ROUTE_CALC ≡ AS_external_LSA .
recalc_old_new_route_table .
remove_multicast_cache_entries_with_matching
_source . ORG_SUMMARY_LSA +
group_membership_LSA .
clear_multicast_cache_entries_for_group .
is_ABR . originate_group_membership_LSA .
ORG_SUMMARY_LSA + (no_AS_external_LSA +
no_group_membership_LSA) .
(intra_AS_calc_already_sched . FULL+
no_intra_AS_calc_already_sched . ((router_LSA
+ network_LSA) . sched_full_AS_route_calc.
ORG_SUMMARY_LSA + summary_LSA .
recalc_old_new_route_table .
remove_multicast_cache_entries_with_matching
_source . ORG_SUMMARY_LSA +
ASBR_summary_LSA .
recalc_asbr_route_table_entry .
ORG_SUMMARY_LSA))

ORG_SUMMARY_LSA ≡ no_LSA_changed . FULL +
LSA_changed . rerun_full_intra_AS_routing_calc
. scan_routing_table (change_intra_area_route+
change_unaggregated_intra_area_route +
change_aggregated_intra_area_route) .
reorgin_summary_LSA . ORG_SUMMARY_LSA

```
EXTR_ROUTING ≡ calc_preferred_path.
    (best_path_from_othersource_or_BGP .
    install_rt_entry_as_RT_STATIC .
    orginate_AS_external_LSA . FULL +
    best_path_from_AS_external_LSA .
    install_rt_entry_as_RT_EXT . FULL )  ■
```

이 LTS 명세의 상태 기계는 그림 1에 도시한 이웃 상태 기계(Neighbor State Machine)[1]와 동일한 여덟 개의 상태들과 여기에 추가적으로 명세된 일곱 개의 내부 상태들로 구성되어 있다. 표준에 명시된 상태들은 DOWN 상태, ATTEMPT 상태, INIT 상태, 2WAY 상태, EXSTART 상태, EXCHNG 상태, LOADING 상태, 그리고 FULL 상태 이다. 이들 상태들의 의미(Semantics)는, 다시 말해 이 모든 상태들과 이들간의 천이를 일으키는 프로토콜 행위들은, IETF 표준의 10.3절에 제시되어 있는 이웃 상태 기계[1]와 완전히 일치한다. 이 일치성은 다음 장에서 정형적으로 검증한다. 그리고 나머지 상태들은, 표준에 약간 언급되어 있으나, 체계적으로 정리가 않되어 있는 내부 상태들로써, 프로토콜의 동작 상황을 상세히 명시하기 위해, 원천 코드가 공개되어 있는 기존의 OSPF 구현의 원천 코드[11]를 참조하여 정의하였다. 이 정의는 다음 장에서 OSPF의 상세 프로토콜 동작의 안전성 검증에 필요한 다양한 특성들의 정형적 검증을 위해서 사용한다.

Ⅲ. OSPF 프로토콜의 정형적 검증

이 장에서는 앞 절에서 제시한 LTS 명세의 프로토콜 상태들과 그들 사이의 천이들이 OSPF 프로토콜 표준의 그것들과 완전히 일치함을 보여, 제시한 명세가 완전성을 가짐을 증명한다. 그리고나서 제시한 명세가 도달성과 생존성을 가짐을 대수 정형적으로 증명한다. 그리고 OSPF 프로토콜이 라이브락 부재이고, 교착상태 부재임을 대수 정형적으로 증명한다. 결과적으로 우리는 이들 증명들을 사용하여, OSPF 프로토콜의 LTS 명세가 안전성과 일관성을 지닌다는 사실을 증명한다.

1. 프로토콜 행위 변수의 개념

일반성을 잃지 않으면서, 증명에 필요없는 프로토콜 행위들을 정형적으로 생략하여, 상태 기계의 여러 가지 성질들을 증명하기 위해서, 다음과 같은 행위 변수[9,10]의 개념을 사용한다.

정 의 2 기호 A+를 무행위(Empty Action)을 제외한 정의 1에 나타난 모든 프로토콜 행위들의 집합이라 정의하자. 집합 A+에 속하는 원소는 0번 이상의 연산 "." 또는 연산 "+" 이 적용된 프로토콜 행위들로 만들어질 수 있는, 정의 1에 나타난, 행위 리스트들이다. 그리고, 집합 A+의 하나 이상의 원소들을 담을 수 있는, 하나 이상의 알파벳 소문자들을 나열하여 만들어지는 기호를 행위 변수로 정의한다.

보조 정리 1 일반성의 상실 없이, 정형적 증명과 관련 없는 프로토콜 행위들을 제거하기 위해서, 우리는 행위 변수를 사용 할 수 있다. 즉 우리가 분석하고자 하는 성질에 초점을 맞추기 위해, 관심의 대상이 아닌 프로토콜 행위들을 행위 변수로 대치하여, 정의 1의 정형적 명세를 간략하게 수정할 수 있다. ■

2. 프로토콜의 완전성 및 상호운용성

이제 다음에서 OSPF 프로토콜의 완전성과 상호운용성을 증명한다.

정 리 1 정의 1에 제시한 LTS 명세는 OSPF 표준 문서에 정의된 모든 상태들과 발생 가능한 모든 외부 프로토콜 행위들을 명세하고 있다. 따라서 본 LTS 명세는 완전성을 지니고 있다. 따라서 제시한 LTS 명세를 만족하는 OSPF 프로토콜의 가용한 모든 원천 코드들은 기존의 상용 라우터들의 OSPF 프로토콜과 상호 운용이 가능하다.

증 명: 완전성을 증명하기 위해서, 정의 1의 LTS 명세의 모든 상태들과 모든 외부 프로토콜 행위들이 IETF의 표준인 RFC2328[1]의 10.3절에 정의된 프로토콜 상태 기계의 모든 상태들과 모든 프로토콜 행위들과 일치함을 보여야 한다. 이를 보이기 위해, 우리는 먼저 OSPF 표준 문서의 10.3절에 정의된 이벤트 (Event)들로 거의 1대1로 사상되는 행위 변수들을 정의하고, 이들 변수들을 사용하여, 정의 1에 제시된 OSPF 프로토콜의 LTS 명세의 각 식을 간략화 한다.

먼저 표준에 정의된 DOWN 상태와 그 상태로부터의 천이들이 상태 DOWN의 상태 방정식과 일치함을 보이기 위해서, 필요한 행위 변수들을 먼저 정의하고, 이들 변수들을 사용하여 상태 방정식을 간략하게 만든다.

```
Start = HelloInterval_expiry . send(hello_pkt) .
    reset_timer_for_HelloInterval +
    PollInterval_expiry . send(hello_pkt) .
```

reset_timer_for_PollInterval

reset_all_timer_for_Hello_pkt =
reset_timer_for_HelloInterval .
reset_timer_for_PollInterval .
reset_RouterDead_timer

hello_received = recv(hello_pkt) .
reset_all_timer_for_Hello_pkt . valid_hello_pkt

exceptionOfDOWN = recv(hello_pkt) .
reset_all_timer_for_Hello_pkt . invalid_pkt
+RouterDeadInterval_expiry .
remove_adjacency + KillNbr + LLDown

이들 변수들로 대치하여, DOWN 상태 방정식을 재기술하면,

DOWN ≡ start . ATTEMPT + hello_received . INIT +
exceptionOfDown . DOWN

로 DOWN 상태 방정식을 간략화 할 수 있다.

상태 ATTEMPT의 상태 방정식이 표준의 ATTEMPT 상태와 이로부터의 천이들과 완전히 일치함을 보이기 위하여, 아래의 행위 변수를 정의하고,

exceptionOfATTEMPT = hello_received .
reset_all_timer_for_Hello_pkt . invalid_pkt .
KillNbr + RouterDeadInterval_expiry + LLDown

이것과 hello_received 변수로 행위들의 리스트를 대치하여, 상태 ATTEMPT를 위한 식을 재기술하면, 다음과 같이 간략화 할 수 있다.

ATTEMPT ≡ hello_received . INIT +
exceptionOfATTEMPT . DOWN

상태 INIT의 상태 방정식이 표준의 상태 INIT와 그 상태로 부터의 천이들과 일치함을 보이기 위해, 행위 변수들을 다음과 같이 정의하고,

2-way_received =
myIPaddress_founded_in_HELLO_pkt

adjacency= conn_to_pnt_to_pnt_net +
conn_to_pnt_to_multipnt_net +
conn_to_virtual_link + (conn_to_broadcast_net
+ conn_to_NBMA_net) .
(conn_to_Designated_Router +
conn_to_Backup_Designated_Router)

no_adjacency = not_conn_to_pnt_to_pnt_net .
not_conn_to_pnt_to_multipnt_net .
not_conn_to_virtual_link .
not_conn_to_broadcast_net .
not_conn_to_NBMA_net +
(conn_to_broadcast_net + conn_to_NBMA_net)
. (not_conn_to_Designated_Router .
not_conn_to_Backup_Designated_Router)

exceptionOfINIT =
myIPaddress_not_founded_in_HELLO_pkt +
recv(hello_pkt) . reset_all_timer_for_Hello_pkt .
valid_hello_pkt . reset_RouterDeadInterval_timer

exceptionOfNbrStateMachine = KillNbr +
RouterDeadInterval_expiry + LLDown

이들을 사용하여 재기술하여 간략화하면 다음과 같다.

INIT ≡ 2-way_received . (adjacency . EXSTART+
not_adjacency . 2WAY) + exceptionOfINIT .
INIT + exceptionOfNbrStateMachine . DOWN

아래의 행위 변수들을 정의하고,

1-way_received =
myIPaddress_not_founded_in_HELLO_pkt

exceptionOf2WAY = (KillNbr +
RouterDeadInterval_expiry + LLDown)

이들과 변수 2-way_received와 변수 1-way_received를 사용하여, 상태 2WAY의 상태 방정식을 재기술하면 다음과 같다.

2WAY ≡ 1-way_received . INIT + adjacency .
EXSTART + exceptionOf2WAY . DOWN +
hello_received . (2-way_received . 2WAY +
1-way_received . INIT)

상태 EXSTART 상태 방정식을 간략화하기 위해서, 아래의 행위 변수들을 정의한다.

NegotiationDone = Master_slave_negotiated .
DD_seq_no_exchanged

```
exceptionOfEXSTART1 =
    recv(database_description_pkt) .
    SeqNumberMismatch + recv(LS_req_pkt) .
    BadLSReq + recv(hello_pkt) .
    reset_RouterDeadInterval_timer .
    (myIPaddress_founded_in_HELLO_pkt .
    EXSTART +
    myIPaddress_not_founded_in_HELLO_pkt .
    removing_adjacency . INIT)

exceptionOfEXSTART2 =
    (not_conn_to_pnt_to_pnt_net .
    not_conn_to_pnt_to_multipnt_net .
    not_conn_to_virtual_link .
    not_conn_to_broadcast_net .
    not_conn_to_NBMA_net +
    (conn_to_broadcast_net + conn_to_NBMA_net)
    . (not_conn_to_Designated_Router .
    not_conn_to_Backup_Designated_Router)) .
    clear_retransmission_list . clear_summary_list .
    clear_LS_req_list + recv(hello_pkt) .
    reset_RouterDeadInterval_timer .
    myIPaddress_not_founded_in_HELLO_pkt

removing_adjacency = clear_retransmission_list .
    clear_summary_list . clear_LS_req_list

exceptionOfEXSTART2  = no_adjacency .
    removing_adjacency + recv(hello_pkt) .
    reset_RouterDeadInterval_timer .
    myIPaddress_not_founded_in_HELLO_pkt .
    removing_adjacency
```

그리고 이들 변수들과 변수 no_adjacency를 사용하여, EXSTART 상태 방정식을 재기술하면 다음과 같다.

```
EXSTART ≡ NegotiationDone . EXCHNG +
    exceptionOfNbrStateMachine  . DOWN +
    exceptionOfEXSTART1 . EXSTART +
    exceptionOfEXSTART2 . INIT
```

상태 EXCHNG의 상태 방정식의 간략화를 위해, 아래의 행위 변수들을 정의하자.

```
exchangeDone = LS_req_list_empty

exceptionOfEXCHNG1 =
    (recv(database_description_pkt) .
    SeqNumberMismatch + recv(LS_req_pkt) .
    BadLSReq + recv(hello_pkt) .
    myIPaddress_not_founded_in_HELLO_pkt)
    .removing_adjacency

exceptionOfEXCHNG2 =  no_adjacency .
    removing_adjacency . INIT + recv(hello_pkt) .
    reset_RouterDeadInterval_timer .
    myIPaddress_not_founded_in_HELLO_pkt .
    removing_adjacency . INIT

exceptionOfEXCHNG3 =recv(hello_pkt) .
    reset_RouterDeadInterval_timer .
    myIPaddress_founded_in_HELLO_pkt .
    EXCHNG
```

그리고 이들 변수와 변수 exceptionOfNbr-StateMachine과 변수 removing_adjacency를 사용하여, EXCHNG 상태식을 다음과 같이 간략화 할 수 있다.

```
EXCHNG ≡ send(data_description_pkt) . EXCHNG
    + no_receive_DD_in_ack .
    retransmission_last_DD . EXCHNG +
    receive_DD_in_ack .
    possibly_add_nb_linkstate_req_list .
    exchangeDone .FULL + not_exchangeDone  .
    send(LS_req_pkt) . LOADING +
    exceptionOfNbrStateMachine . DOWN +
    exceptionOfEXCHNG . EXSTART +
    exceptionOfEXCHNG2 . INIT +
    exceptionOfEXCHNG3 . EXCHNG
```

상태 LOADING의 상태 방정식의 간략화를 위해서, 다음의 행위 변수들을 정의하자.

```
loading_Done = LS_req_list_empty

exceptionOfLOADING = recv(hello_pkt) .
    reset_RouterDeadInterval_timer .
    myIPaddress_founded_in_HELLO_pkt
```

이들 변수들과 앞서 정의한 변수들 중 일부를 사용하여, LOADING 상태식을 다음과 같이 간략화하여 재기술 할 수 있다.

```
LOADING ≡ loading_Done  . FULL +
    recv(LS_update) . delete_LS_req_in_list .
    (loading_Done . FULL + not_loading_Done .
```

send(LS_req_pkt) . LOADING) +
req_retrans_time_expire . send(LS_req_pkt) .
LOADING + exceptionOfNbrStateMachine .
DOWN + exceptionOfEXSTART1 . EXSTART +
exceptionOfEXSTART2 . INIT +
exceptionOfLOADING . LOADING

FULL 상태 방정식의 간략화를 위해서, 다음 행위 변수를 정의한다.

exceptionOfFULL = recv(hello_pkt) .
reset_RouterDeadInterval_timer .
myIPaddress_founded_in_HELLO_pkt

이 변수와 앞서 정의한 변수들 중 일부를 사용하여, FULL 상태식을 다음과 같이 간략화 할 수 있다.

FULL ≡ (local_rtr_state_change + LSA_Reflesh +
Receive_Self_LSA) . ORIGINATE_LSA +
recv_LS_update . RECV_N_FLOOD_LSA +
retrans_timer_expire . build_retrans_update .
flood_out_interfaces . FULL +
route_calc_timer_expired . full_route_calc .
updating_route_table . FULL +
exceptionOfNbrStateMachine . DOWN +
exceptionOfEXCHNG1 . EXSTART +
no_adjacency . removing_adjacency . INIT +
recv(hello_pkt) . reset_RouterDeadInterval_timer
. myIPaddress_not_founded_in_HELLO_pkt .
removing_adjacency . INIT + exceptionOfFULL .
FULL

앞에서 재기술하여 간략화된 식들을 정리하면, 다음과 같다.

DOWN ≡ start . ATTEMPT + hello_received . INIT +
exceptionOfDown . DOWN

ATTEMPT ≡ hello_received . INIT +
exceptionOfATTEMPT . DOWN

INIT ≡ 2-way_received . (adjacency . EXSTART+
not_adjacency . 2WAY) + exceptionOfINIT .
INIT + exceptionOfNbrStateMachine . DOWN

2WAY ≡ 1-way_received . INIT + adjacency .
EXSTART + exceptionOf2WAY . DOWN +
hello_received . (2-way_received . 2WAY +
1-way_received . INIT)
EXSTART ≡ NegotiationDone . EXCHNG +
exceptionOfNbrStateMachine . DOWN +
exceptionOfEXSTART1 . EXSTART +
exceptionOfEXSTART2 . INIT

EXCHNG ≡ send(data_description_pkt) . EXCHNG
+ no_receive_DD_in_ack .
retransmission_last_DD . EXCHNG +
receive_DD_in_ack .
possibly_add_nb_linkstate_req_list .
exchangeDone .FULL + not_exchangeDone .
send(LS_req_pkt) . LOADING) +
exceptionOfNbrStateMachine . DOWN +
exceptionOfEXCHNG . EXSTART +
exceptionOfEXCHNG2 . INIT +
exceptionOfEXCHNG3 . EXCHNG

LOADING ≡ loading_Done . FULL +
recv(LS_update) . delete_LS_req_in_list .
(loading_Done . FULL + not_loading_Done .
send(LS_req_pkt) . LOADING) +
req_retrans_time_expire . send(LS_req_pkt) .
LOADING + exceptionOfNbrStateMachine .
DOWN + exceptionOfEXSTART1 . EXSTART +
exceptionOfEXSTART2 . INIT +
exceptionOfLOADING . LOADING

FULL ≡ (local_rtr_state_change + LSA_Reflesh +
Receive_Self_LSA) . ORIGINATE_LSA +
recv_LS_update . RECV_N_FLOOD_LSA +
retrans_timer_expire . build_retrans_update .
flood_out_interfaces . FULL +
route_calc_timer_expired . full_route_calc .
updating_route_table . FULL +
exceptionOfNbrStateMachine . DOWN +
exceptionOfEXCHNG1 . EXSTART +
no_adjacency . removing_adjacency . INIT +
recv(hello_pkt) . reset_RouterDeadInterval_timer
. myIPaddress_not_founded_in_HELLO_pkt .
removing_adjacency . INIT + exceptionOfFULL .
FULL

위에서 제시한 여덟 개의 상태 방정식들이 표준 문서의 이웃 상태 기계 (그림 1 참조)의 여덟 개의 상태들과 그들 사이의 천이를 표현하고 있다. 우리는 RFC2328 표준 문서의 10.3절의 프로토콜 상태 기계의 상태들과 그 상태들로부터 출발하는 이벤트들을 표준 문서와 동일한 이름의 상태를 동치 기호 '≡'의 좌측에 가지고 있는 상태 방정식과 각각 비교함으로써, 표준 문서의 프로토콜 상태 기계에 대

한 기술과 LTS 명세가 완전히 일치함을 알 수 있다. 특히, 표준 문서의 10.3절의 프로토콜 상태 기계의 이벤트들에 관한 기술과 제시한 프로토콜 행위 변수들의 정의에 나타난 세부 프로토콜 행위들을 각각 나누어 비교하여 봄으로써, 각 프로토콜 행위 변수의 정의가 표준 문서의 10.3절의 행위들의 기술과 각각 일치하고 있음을 간단히 확인 할 수 있다. 따라서, OSPF 프로토콜의 LTS 명세가 완전성 (Completeness)을 가짐을 알 수 있다.

따라서 본 LTS 명세를 만족하는 OSPF 프로토콜 소프트웨어는 기존의 상용 제품들의 그것들과 상호 운용이 가능하다. ■

그리고 우리가 상세한 명세를 만들기 위해 참조한 OSPF 프로토콜의 원천 코드는 RFC2328 표준에 정의된 모든 프로토콜 행위들에 대해 정확하게 반응을 하도록 구현되었다는 사실을 알수 있었음을 언급해둔다.

3. OSPF 표준 명세의 도달성 및 생존성 그리고 라이브락-부재 특성 증명

본 절에서는 제시한 LTS 명세를 사용하여, 프로토콜의 도달성과 생존성, 그리고 라이브락 부재를 정형적으로 증명한다. 먼저 모든 상태들이 초기 상태로부터 도달 가능하다는 것을 의미하는 도달성을 LTS로 명세한 프로토콜 상태 기계가 지님을 증명한다. 그리고 모든 상태들로부터 제한된 수의 천이들을 거친 후 초기 상태로 도달이 가능함을 의미하는 생존성을 프로토콜 상태 기계가 가짐을 대수적으로 증명한다. 마지막으로 OSPF 프로토콜의 LTS 명세의 프로토콜 동작들의 폐흐름 (Loop)들이 모두 생산적 폐흐름임을 보여, OSPF 프로토콜이 라이브락 부재 임을 대수적으로 증명한다.

이러한 특성들을 증명하기 위해, 이제 다음과 같이 행위 변수들을 정의하여, LTS 명세를 간략화 한다. 상태 ATTEMPT의 상태 방정식에 행위 변수를 도입하여, 보이고자 하는 성질에 관련없는 프로토콜 행위를 간략화하는 예를 보이면 다음과 같다. 여기서 행위 hello_received만이 증명에서 관심의 대상이 되는 외부 프로토콜 행위라 가정하자.

ATTEMPT ≡ hello_received . INIT +
exceptionOfATTEMPT . DOWN
≡ hello_received . INIT + a . DOWN

이제 OSPF 프로토콜의 LTS 명세의 도달성을 다음에서 보인다.

정 리 2 검증 대상인 OSPF 프로토콜의 LTS 명세는 도달성을 지닌다. 다시 말해서, LTS로 명세한 OSPF의 프로토콜 상태 기계의 모든 상태들은 초기 상태 (DOWN 상태)로부터 도달이 가능하다.

증 명: 도달성을 보이기 위해, 행위 변수들을 새로이 정의하여, 우리는 정의 1에서 제시한 정형적 명세를 다음과 같이 간략화 할 수 있다. 먼저 정의 1의 모든 상태식들을 수정하여, 각 식의 동치 부호 "≡"의 좌측에 나타난 상태를 다음과 같이 초기 상태와 초기 상태로부터 그 상태에 도달하기 전에 거쳐야하는 프로토콜 행위들을 담고있는 변수로 대치하자. 그리고, 각 식들의 동치 부호 "≡"의 우측에 나타난 항들을 다음과 같이 각 식에 나타난 상태들과 이 상태에 도달하기 이전에 발생하여야 하는 프로토콜 행위들을 담고있는 행위 변수로 간략화 하자.

DOWN ≡ a . ATTEMPT + b . INIT + c. DOWN

DOWN . a ≡ d . INIT + e . DOWN

DOWN . abd ≡ f. 2WAY + g . EXSTART + h . INIT + i . DOWN

DOWN . abdf ≡ j. INIT + k . EXSTART + l . DOWN + m . 2WAY

DOWN . abdfk ≡ n. EXCHNG + o . DOWN + p . EXSTART + + q . INIT

DOWN . abdfkn ≡ r . EXCHNG + s . FULL + t . LOADING + u . DOWN + v . EXSTART + w . INIT

DOWN . abdfknt ≡ x . FULL + y . LOADING

DOWN . abdfkntx ≡ z. ORIGINATE_LSA + aa . RECV_N_FLOOD_LSA + bb . FULL + cc . DOWN + dd . EXSTART + ee . INIT

DOWN . abdfkntxz ≡ ff . FULL + gg . ADD_DATABASE

DOWN . abdfkntxzgg ≡ hh . SCHEDULING_ROUTE_CALC

DOWN . abdfkntxzgghh ≡ ii . FULL + jj . EXTER_ROUTING + kk . (ADD_DATABASE + DEL_DATABASE)

DOWN . abdfkntxzgghhkk ≡ ll .
SCHEDULING_ROUTE_CALC

DOWN . abdfkntxzgghhkkll ≡ mm .
ORG_SUMMARY_LSA + nn . FULL

DOWN . abdfkntxzgghhkkllmm ≡ oo . FULL + pp .
ORG_SUMMARY_LSA

DOWN . abdfkntxzgghhjj ≡ qq . FULL

그러면, 위에서 보인 바와 같이, 상기 식들 중에 적어도 하나의 식의 동치 기호 우측에 상태들을 나타내는 모든 심볼들이 나타난다. 따라서, 모든 상태들은 초기상태로 부터 도달이 가능하다는 사실을 증명하였다. ■

다음으로 OSPF 프로토콜의 생존성을 증명한다.

정 리 3 OSPF 프로토콜의 LTS 명세는 생존성을 지닌다. 다시 말해서, OSPF 프로토콜의 LTS 명세의 모든 상태들은, 제한된 수의 천이들을 한 후, 초기 상태에 도달가능 하다.

증 명: 생존성을 보이기 위해, 행위 변수들을 새로이 정의하여, 정의 1에서 제시한 상태 방정식들을 간략화 하면 다음과 같다.

DOWN ≡ a . ATTEMPT + b . INIT + c. DOWN

ATTEMPT ≡ d . INIT + e . DOWN

INIT ≡ f. 2WAY + g . EXSTART + h . INIT + i .
DOWN

2WAY ≡ j. INIT + k . EXSTART + l . DOWN + m .
2WAY

EXSTART ≡ n. EXCHNG + o . DOWN + p .
EXSTART + + q . INIT

EXCHNG ≡ r . EXCHNG + s . FULL + t . LOADING
+ u . DOWN + v . EXSTART + w . INIT

LOADING ≡ x . FULL + y . LOADING

FULL ≡ z. ORIGINATE_LSA + aa .
RECV_N_FLOOD_LSA + bb . FULL + cc .
DOWN + dd . EXSTART + ee . INIT

ORIGINATE_LSA ≡ ff . FULL + gg .
ADD_DATABASE

ADD_DATABASE ≡ hh . (ii . ORG_SUMMARY_LSA +
jj . FULL)

RECV_N_FLOOD_LSA ≡ kk . FULL + ll .
EXTER_ROUTING + mm . (ADD_DATABASE +
DEL_DATABASE)

DELETE_DATABASE ≡ nn. (oo .
ORG_SUMMARY_LSA + pp . FULL)

SCHEDULING_ROUTE_CALC ≡ qq .
ORG_SUMMARY_LSA + rr . FULL

ORG_SUMMARY_LSA ≡ ss . FULL + tt .
ORG_SUMMARY_LSA

EXTR_ROUTING ≡ uu . FULL

OSPF 프로토콜의 초기 상태는 DOWN 상태 이다.[1] 위에서 보인바와 같이, FULL 상태 이전의 모든 상태 방정식들은 적어도 하나의 DOWN 상태로 끝나는 항이 있고, FULL 상태 이후의 모든 상태 방정식은 FULL 상태로 끝나는 항을 적어도 하나 가지고 있다. 따라서, 상기의 상태 식들의 동치 기호 좌측에 나타난 모든 상태들은, 프로토콜 변수가 포함할 수 있는 행위들을 수행한 후, 초기 상태에 도달 한다. 그러므로, 우리는 LTS 명세에 기술된 모든 상태들이, 제한된 수의 상태 천이들을 한 후, 초기 상태로 도달한다는 사실을 증명하였다. ■

정 리 4 OSPF 프로토콜의 LTS 명세에 기술된 프로토콜 기계는 라이브락 부재 특성을 갖는다. 다시 말해서, 명세된 OSPF 프로토콜 동작들의 폐흐름(Loop)들은 모두 생산적 폐흐름들 이다.

증 명: 정리 1에서 도입한 프로토콜 행위 변수들과 추가하여 정의하는 행위 변수들을 사용하여, 정의 1에 제시된 LTS 정형적 명세를 최소 하나의 생산적인 외부 프로토콜 행위 만을 남기고 간략화 하면, 다음과 같다.

DOWN ≡ start . ATTEMPT + hello_received . INIT +
exceptionOfDown . DOWN

ATTEMPT ≡ hello_received . INIT +
exceptionOfATTEMPT . DOWN

INIT ≡ 2-way_received . (a . EXSTART+ b . 2WAY)
+ exceptionOfINIT . INIT +
exceptionOfNbrStateMachine . DOWN

2WAY ≡ 1-way_received . INIT + adjacency .
EXSTART + exceptionOf2WAY . DOWN +
hello_received . (c . 2WAY + d . INIT)

EXSTART ≡ NegotiationDone . EXCHNG +
exceptionOfNbrStateMachine . DOWN +
exceptionOfEXSTART1 . EXSTART +
exceptionOfEXSTART2 . INIT

EXCHNG ≡ send(data_description_pkt) . EXCHNG
+ e . retransmission_last_DD . EXCHNG +
receive_DD_in_ack . f .FULL + g .
send(LS_req_pkt) . LOADING) +
exceptionOfNbrStateMachine . DOWN +
exceptionOfEXCHNG . EXSTART +
exceptionOfEXCHNG2 . INIT +
exceptionOfEXCHNG3 . EXCHNG

LOADING ≡ loading_Done . FULL +
recv(LS_update) . h . (i . FULL + j .
send(LS_req_pkt) . LOADING) + k .
send(LS_req_pkt) . LOADING +
exceptionOfNbrStateMachine . DOWN +
exceptionOfEXSTART1 . EXSTART +
exceptionOfEXSTART2 . INIT +
exceptionOfLOADING . LOADING

FULL ≡ (local_rtr_state_change + LSA_Reflesh +
Receive_Self_LSA) . ORIGINATE_LSA +
recv_LS_update . RECV_N_FLOOD_LSA + l .
flood_out_interfaces . FULL + m .
updating_route_table . FULL +
exceptionOfNbrStateMachine . DOWN +
exceptionOfEXCHNG1 . EXSTART + n .
removing_adjacency . INIT + recv(hello_pkt) . o
. INIT + exceptionOfFULL . FULL

ORIGINATE_LSA ≡
interface_type_specific_LSA_orginate. j . (k .
FULL + l . (m . FULL + n . ADD_DATABASE))

ADD_DATABASE ≡ o. flood_out_interfaces .
SCHEDULING_ROUTE_CALC

RECV_N_FLOOD_LSA ≡ p . discard_LSA . FULL +
LSA_valid . (q . send_dirt_ack . FULL + r .
schd_delayed_ACK . (s . EXTER_ROUTING + t .
(ADD_DATABASE + DEL_DATABASE)))

DELETE_DATABASE ≡ t . remove_from_AVL_tree .
uSCHEDULING_ROUTE_CALC

SCHEDULING_ROUTE_CALC ≡ v .
remove_multicast_cache_entries_with_matching
_source . ORG_SUMMARY_LSA + w .
orginate_group_membership_LSA .
ORG_SUMMARY_LSA + x .
(intra_AS_calc_already_sched . FULL+ y . (z .
sched_full_AS_route_calc.
ORG_SUMMARY_LSA + aa .
remove_multicast_cache_entries_with_matching
_source . ORG_SUMMARY_LSA + ab .
recalc_asbr_route_table_entry .
ORG_SUMMARY_LSA))

ORG_SUMMARY_LSA ≡ no_LSA_changed . FULL +
ac . reorgin_summary_LSA .
ORG_SUMMARY_LSA

EXTR_ROUTING ≡ ad . orginate_AS_external_LSA .
FULL + ae . install_rt_entry_as_RT_EXT . FULL)

위의 모든 상태 방정식에서 보듯이, 상태 식들의 모든 상태들은 적어도 하나의 생산적인 외부 프로토콜 행위를 하여야만 다른 상태들로 천이할 수 있다. 결과적으로 OSPF 프로토콜의 동작들의 모든 폐흐름(Loop)들은 적어도 하나의 생산적 프로토콜 행위를 수행하는 폐흐름 이다. 따라서, 명세된 OSPF 프로토콜 상태기계가 라이브락 부재 특성을 가짐을 증명하였다. ■

4. 교착상태 부재 및 안전성 그리고 일관성

본 절에서는 OSPF 프로토콜이 교착상태 부재 특성 및 안전성 그리고 일관성을 가지고 있음을 보인다. 교착 상태란 프로토콜 행위에 참여하는 개체들이, 일어날 수 없는 일이 일어나기를 기다리며, 프로토콜의 행위를 실행하지 않는 상태를 의미한다[4].

정 리 5 OSPF 프로토콜의 프로토콜 행위는 프로토콜 행위에 참여하는 개체들 간에 교착상태를 발생시키지 않는다.

증 명: 정의 1에서 제시한 OSPF의 프로토콜 상태 기계인 이웃 상태 기계는 인접한 라우터의 한 정합장치의 이웃 상태 기계와 일대일로 대응하여 동작한다. 따라서 이웃한 두개의 프로토콜 상태 기

계 간의 상호 작용만을 고려하여, 교착상태 부재를 증명할 수 있다. 이제 임의의 행위 변수들을 도입하여, OSPF 상태 기계에서 교착 상태를 만들 수 있는 유일한 외부 동작인 패킷의 송수신 동작과 타이머 동작 만을 남기고, 상태식들을 다음과 같이 간략화 하자.

DOWN ≡ a . send(hello_pkt) . reset_timer . ATTEMPT + recv(hello_pkt) . reset_timers . (b . INIT + c . DOWN) + d . DOWN

ATTEMPT ≡ recv(hello_pkt) . reset_timers . (b . INIT + e . DOWN) + f . DOWN

INIT ≡ g . EXSTART + h . 2WAY + i . INIT + j . DOWN + recv(hello_pkt) . reset_timers . k . INIT

2WAY ≡ l . INIT + m. EXSTART + n . DOWN + recv(hello_pkt) . reset_timer . (o . 2WAY + p . INIT)

EXSTART ≡ q. EXCHNG + r . DOWN + s . EXSTART + t . INIT

EXCHNG ≡ send(data_description_pkt) . EXCHNG + no_receive_DD_in_ack . retransmission_last_DD . EXCHNG + receive_DD_in_ack . u . (v .FULL + w . send(LS_req_pkt) . LOADING) + x . DOWN + (recv(database_description_pkt) . y + recv(LS_req_pkt) . z + recv(Hello_pkt) . aa) .bb . EXSTART + cc . INIT + recv(hello_pkt) . reset_timer . (dd . EXCHNG + ee . INIT)

LOADING ≡ ff . FULL + recv(LS_update) . gg. FULL + hh . send(LS_req_pkt) . LOADING + ii . send(LS_req_pkt) . LOADING + jj . DOWN + kk . EXSTART + recv(hello_pkt) . reset_timer . LOADING + ll . INIT + recv(hello_pkt) . mm . LOADING

FULL ≡ nn . ORIGINATE_LSA + recv_LS_update . RECV_N_FLOOD_LSA + retrans_timer_expire . build_retrans_update . flood_out_interfaces . FULL + oo . FULL + pp . DOWN + (recv(database_description_pkt) . pp + recv(LS_req_pkt) . qq + recv(Hello_pkt) . rr) . ss . EXSTART + tt . INIT + recv(hello_pkt) . reset_timer . (uu . FULL + vv . INIT)

ORIGINATE_LSA ≡ interface_type_specific_LSA_orginate. ww . FULL + xx . flush_LSA_instead . FULL+ yy . ADD_DATABASE

ADD_DATABASE ≡ ((zz .orignateLSA) + receiveLSA) . aaa . flood_out_interfaces . SCHEDULING_ROUTE_CALC

RECV_N_FLOOD_LSA ≡ bbb . FULL + LSA_valid . (ccc . send(dirt_ack_pkt) . FULL + ddd . EXTER_ROUTING + eee . (ADD_DATABASE + DEL_DATABASE))

DELETE_DATABASE ≡ fff. SCHEDULING_ROUTE_CALC

SCHEDULING_ROUTE_CALC ≡ ggg . ORG_SUMMARY_LSA + hhh . originate_group_membership_LSA . ORG_SUMMARY_LSA + iii . FULL+ jjj . ORG_SUMMARY_LSA

ORG_SUMMARY_LSA ≡ kkk . FULL + lll . reorgin_summary_LSA . ORG_SUMMARY_LSA

EXTR_ROUTING ≡ mmm . orginate_AS_external_LSA . FULL + nnn . FULL

위의 모든 상태 방정식들에 나타난 바와 같이, 위의 프로토콜 상태 기계는 OSPF 패킷의 송수신 동작 다음에, 내부 프로토콜 동작을 수행하고, 바로 다른 상태 (혹은 같은 상태로) 천이한다. 그리고 위의 각 식에 명시된 바와 같이, 모든 프로토콜 동작들이 이벤트로 구동되고, 외부 패킷을 기다리는 프로세스는 존재하지 않는다. 따라서 프로토콜에 의해 정의되는 개체들 간에 패킷의 수신을 기다리며 순환적인 대기를 하는 동작은 존재하지 않는다. 결과적으로 교착 상태가 발생할 수 없다는 사실을 보였다. ■

정 리 6 LTS로 명세한 OSPF 프로토콜은 안전성을 가진다. 다시 말해서, OSPF 프로토콜의 명세는 교착 상태 부재이고, 라이브락 부재 이다.

증 명: 정리 4과 정리 5에 의해서 증명된다.

정 리 7 LTS로 명세한 OSPF 프로토콜은 일관성을 가진다. 다시 말해서, OSPF 프로토콜은 안전성

을 가지고, 적절한 종료 조건들이 만족되어야만, 프로토콜의 행위가 끝난다.

증 명: 정리 1, 2, 3, 4, 5, 6에 의해서 증명된다.

VI. 결 론

본 논문에서, 초고속 라우터를 포함한, 라우터 시스템의 개발에 필수적이고 핵심적인 소프트웨어인 OSPF 프로토콜의 표준 문서를 분석하여, 직관적 문장들로 기술된 명세를 대수 정형적 기법인 Label Transition System (LTS) 명세 기법을 사용하여 정형적으로 명세하였다. 그리고, 기존의 구현된 OSPF 프로토콜 소프트웨어의 원천 코드를 참고하여, 표준 문서에 상세히 기술되지 않은 내부 상태들과 내부 프로토콜 행위들까지 상세히 표현하도록 LTS 명세를 보완하여, 상세 설계 수준의 정형적 명세를 제공하였다. 그리고 명세한 프로토콜 상태 기계의 상호 운용성과 안전성을 정형적인 방법으로 검증하였다.

초고속 라우터의 동작의 신뢰성 확보를 위해, OSPF 프로토콜 소프트웨어의 상호운용성과 안전성의 보장은 중요하다. 우리는 OSPF 프로토콜의 LTS 명세의 모든 상태 방정식들이, OSPF 프로토콜의 표준 문서인 IETF의 RFC 2328에 명시된 모든 상태들과 그 상태들로부터 발생가능한 모든 외부 프로토콜 행위들과 각각 일치하고 있음을 보여, 제시된 LTS 명세의 완전성을 보였다. 결과적으로, 제시한 OSPF 프로토콜의 LTS 명세를 만족하는, 세부적인 내부 상태들과 내부 프로토콜 행위들의 명세를 위해 참조한 OSPF 프로토콜 소프트웨어가 기존의 상용 라우터들의 그것들과 상호 운용될 수 있다는 사실을 보였다.

마지막으로, OSPF 프로토콜의 LTS 정형적 명세를 사용하여, 초기 상태로부터의 모든 상태들로의 도달성을 정형적으로 증명하고, 모든 상태들로부터 초기 상태로의 도달가능성을 나타내는 생존성을 정형적 기법을 사용하여 증명하였다. 그리고 프로토콜 동작에서의 폐흐름들이 모두 생산적 폐흐름임을 보여, LTS 명세가 라이브락 부재 특성을 가짐을 정형적으로 증명하였다. 또한 OSPF 프로토콜의 교착상태 부재 특성을 증명하였다. 이들 특성들의 증명을 기반으로 OSPF 프로토콜의 LTS 명세의 안전성 및 일관성을 검증하였다.

제시한 LTS 명세는 C 언어의 함수 수준으로 상세하게 명세되었기 때문에, 이 정형적 명세와 정형적 검증은 OSPF 프로토콜의 동작을 깊이 이해할 수 있게 도와줄 뿐만 아니라, 상용 라우터의 개발에 필수적인 OSPF 프로토콜의 신뢰성을 개선하기 위해 사용할 수 있다. 그리고 현재 Draft 상태에 있는 IPv6를 수용한 OSPF Version 3에서, 프로토콜 상태 기계는 기존의 OSPF Version 2의 메커니즘을 그대로 쓰고 있으므로, 본 검증은 OSPF Version 3에서도 유효하다. 또한 본 논문에서 제시한 검증 기법들은 다른 프로토콜들의 검증에도 사용될 수 있음을 밝혀둔다.

참 고 문 헌

[1] J. Moy, "OSPF Version 2,"*Internet RFC 2328*, Internet Engineering Task Force, April 1998.

[2] A. Fekete, N. Lynch, and L. Shrira, "A Modular Proof of Correctness for a Network Synchronizer," *In J. van Leeuwen, editor, Distributed Algorithms (2nd International Workshop, Amsterdam, The Netherlands, July 1987), volume 312 of Lecture Notes in Computer Science*, pp. 219-256. Springer-Verlag, 1988.

[3] C. Fetzer and F. Cristian, "A Highly Available Local Leader Election Service," *IEEE Transactions on Software Engineering*, Vol. 25, No. 5, pp. 603-618, 1999.

[4] Gerard J. Holzmann, *Design and Validation of Computer Protocols*, Reading Mass, Prentice Hall, 1991.

[5] D. Cypher, D. Lee, M. Martin-Villalba, C. Prins, and D. Su, "Formal Specification, Verification, and Automatic Test Generation of ATM Routing Protocol: PNNI," *Formal Description Techniques and Protocol Specification, Testing, and Verification (FORTE/PSTV 1998)*, IFIP, Kluwer, Boston, Mass, 1998.

[6] K. Bhargavan, D. Obradovic, and C. A. Gunter, "Formal Verification of Standards for Distance Vector Routing Protocols," *Journal of the ACM (JACM)*, Vol. 49, No. 4, pp. 538-576, 2002.

[7] C. A. R. Hoare, *Communicating Sequential Processes*, Reading Mass, Prentice Hall, 1985.

[8] R. Milner, *Communication and Concurrency*, Reading Mass, Prentice Hall, 1989.

[9] Jae-Hyun Park, "Validation of the Detailed Design of the Label Distribution Protocol for the Multiprotocol Label Switching System," *Proc. of Globecom 2001*, November 2001.

[10] Jae-Hyun Park, "Validation of the Detailed Design of the Label Distribution Protocol for the MultiprotocolLabel Switching System," *IEICE Tr. on Communications*, Vol. E86-B, No. 2, p.p. 506-517, Februrary 2003.

[11] J. Moy, *OSPF Complete Implementation*, Reading Mass, Addison Wesley, 2001.

박 재 현(Jae-Hyun Park) 정회원

1988년 2월 : 중앙대학교 전자계산학과 졸업

1991년 2월 : 한국 과학기술원 전산학과 석사

1995년 8월 : 한국 과학기술원 전산학과 박사

1995년 8월~2000년 2월 : 삼성전자 정보통신 본부 데이터네트웍 개발팀 MPLS/ATM 개발담당

2000년 3월~2002년 8월 : 영남대학교 전자정보공학부, 정보통신 전공 교수

2002년 9월~현재 : 중앙대학교 컴퓨터 공학부 교수

<주관심 분야> ATM Switch Arch., 상호연결 네트웍, Multiprotocol Label Switching System, Routing Protocols