HÁLÓZATI PROTOKOLLOK FORMÁLIS SZEMMEL

NETWORK PROTOCOLS FROM A FORMAL ASPECT

Dávid Ákos[1], Csatári János[2], Beleznay Péter[3]

Összefoglaló: A hálózati protokollok a nagyméretű és komplex szoftverrendszerek sajátosságait hordozzák, ezért egymásra épülő, hierarchikus rétegekbe szervezik őket (pl.: ISO/OSI referenciamodell). A rétegelt szerkezet lehetővé teszi, hogy az egyes kommunikációs funkciókat egymástól függetlenül definiáljuk, ezzel együtt ezek fejlesztése is külön – eltérő szabványok szerint – történjen. Egy adott réteghez tartozó protokoll egyfajta fekete dobozként képzelhető el, amely úgy kínál szolgáltatásokat a felette lévő rétegnek, hogy a vele egy szinten lévő egyedekkel kommunikálva igénybe veszi az alatta lévő réteg szolgáltatásait. A tényleges protokoll az adott réteg kommunikációs szabályaiból és a felette lévő rétegnek nyújtott szolgáltatásokból (szolgáltatásspecifikáció) tevődik össze. A protokoll egyedeinek leírását, valamint a szolgáltatásspecifikációt együttesen protokollspecifikációnak nevezzük. Miután az informális technikával és természetes nyelvek segítségével történő protokollspecifikáció elégtelennek bizonyult, a figyelem középpontjába a formális módszerek kerültek. A formális specifikációk használatának egyik előnye, hogy szigorúan elemezhetők a teljesség és a következetesség szempontjából. Jelen cikk egy valós, széles körben elterjedt irányítóprotokoll – az OSPF – példáján keresztül próbálja bemutatni, hogyan is alkalmazható a gyakorlatban már jól bevált modellellenőrzés egy adott hálózati protokoll tervezési fázisában, de akár utólagos ellenőrzésénél is, amelynek során az eredeti ajánlásokban specifikált tulajdonságok teljesülése vizsgálható.

Kulcsszavak: hálózat, protokoll, biztonság, verifikáció

Abstract: Network protocols have all the characteristics of large and complex software systems. For this reason they are organized into hierarchical layers (e.g. ISO/OSI reference model) built on one another. The layered structure enables the independent definition of specific communication functions, and the separate development of these, based on different standards. A protocol belonging to a specific layer operates like some kind of a black box that provides services to the layer above, communicates with entities of the same layer, while uses services provided by the layer below. The actual protocol consists of the communication rules of the given layer, and the services (service specification) provided to the layer above. The description of the entities of the protocol together with the service specification is referred to as protocol specification. As protocol specifications with informal techniques and natural languages became insufficient, formal methods have been placed into the focus. One of the advantages of using formal specifications is that they can be strictly analyzed from the aspects of completeness and consistency. This article attempts to illustrate through the example of a real, wide spread routing protocol – OSPF – how to apply the well known model checking in the design phase of a network protocol, or even for post verification purposes whether properties specified in the original RFC still hold.

Keywords: network, protocol, security, verification

1. Bevezetés

A számítógépes hálózatok óriási fejlődésen mentek keresztül az elmúlt években, évtizedekben (Karlin et al 2008). Nem csupán a hálózatok fizikai alkotóelemei fejlődtek az egyre modernebb gyártástechnológiának köszönhetően, hanem az eszközök együttműködését lehetővé tevő és szabályozó protokollok is (Kent et al 2000). Ennek eredményeképpen ma már nincs az életnek olyan

[1] Pannon Egyetem, Műszaki Informatikai Kar,
davida@almos.uni-pannon.hu
2 Szegedi Tudományegyetem, Természettudományi és Informatikai Kar,
csatarij@optanet.hu
3 Fast Lane Kft.
pbeleznay@flane.com

A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg (a támogatás száma TÁMOP 4.2.1./B-09/1/KMR-2010-0003).

szegmense, ahol eredményesen, hatékonyan lehetne boldogulni megfelelő hálózati támogatás nélkül. Az egyik vezető hálózati eszközöket gyártó cég vezetője foglalta össze találóan, hogy a hálózat megváltoztatta mindazt, ahogy dolgozunk, élünk, játszunk és tanulunk. A változás különösen szembetűnő a webes felületeken, illetve a közösségi oldalakon, hisz a fiatal generáció tagjai ennek segítségével beszélik meg az iskolai felkészülést, szervezik meg a közös programokat, majd ide töltik fel az ott készült fotókat és videókat. Jelenleg is óriási mennyiségű adat halad az információs szupersztrádán, és ez a mennyiség a videokommunikáció folyamatos előretörésével exponenciálisan emelkedik. 2015-re várhatóan a hálózati forgalom 91%-át videoanyagok továbbítása teszi ki, továbbá kb. 30 millió munkavállaló legalább heti egy napot otthonról fog dolgozni.

A hálózati infrastruktúra általánosságban készen áll a megnövekedett adatforgalom kiszolgálására, de mi a helyzet a hálózat és az azon keresztülhaladó adatok védelmével? Szerencsére egyre nagyobb hangsúlyt kap a megfelelő hálózatbiztonság kialakítása kis- és nagyvállalatoknál egyaránt. Ennek eredményeképpen használunk jogosultságkezelést, tűzfalat, demilitarizált zónát (DMZ), behatolás-érzékelő (IDS) vagy -megelőző (IPS) rendszereket.

A számítógépes bűnözés fejlődésével viszont a hálózat elleni támadások is egyre kifinomultabbá váltak (McDaniel et al 2006). Már nem feltétlenül igényelnek felhasználói interakciót, inkább a hálózat alsóbb rétegeit célozzák. Az itt működő irányító- és szállítási protokollok felelnek az összes áthaladó adat megfelelő irányba történő, legjobb szándékú továbbításáért. Ha egy támadás sikerrel jár, akkor az összes áthaladó információ eltéríthető.

2. A formális módszerek és a protokollok viszonya

Az alábbi alfejezetek a formális módszerek és a hálózati vagy kommunikációs protokollok viszonyát mutatják be röviden.

1.1. Protokollspecifikáció

A hálózati protokollok a nagyméretű és komplex szoftverrendszerek sajátosságait hordozzák, ezért egymásra épülő, hierarchikus rétegekbe szervezik őket (pl.: ISO/OSI referenciamodell). A rétegelt szerkezet lehetővé teszi, hogy az egyes kommunikációs funkciókat egymástól függetlenül definiáljuk, ezzel együtt ezek fejlesztése is külön – eltérő szabványok szerint – történjen. Egy adott réteghez tartozó protokoll egyfajta fekete dobozként képzelhető el, amely úgy kínál szolgáltatásokat a felette lévő rétegnek, hogy a vele egy szinten lévő egyedekkel kommunikálva igénybe veszi az alatta lévő réteg szolgáltatásait. A tényleges protokoll az adott réteg kommunikációs szabályaiból és a felette lévő rétegnek nyújtott szolgáltatásokból (szolgáltatásspecifikáció) tevődik össze. A protokoll egyedeinek leírását, valamint a szolgáltatásspecifikációt együttesen protokollspecifikációnak nevezzük. Miután az informális technikával és természetes nyelvek segítségével történő protokollspecifikáció elégtelennek bizonyult, a figyelem középpontjába a formális módszerek kerültek (Lécz és Zömbik 2004). A formális specifikációk használatának egyik előnye, hogy szigorúan elemezhetők a teljesség és a következetesség szempontjából. Ez nélkülözhetetlen az elosztott rendszerek esetében használt protokollok különböző megvalósításai közötti együttműködéshez.

A protokollspecifikációhoz használt formális módszereknél három modellt vehetünk alapul (Sidhu et al 1991):

1. Állapot-átmeneti modell – A hálózati protokoll olyan esemény alapú egyed, amely üzenetváltásokkal dolgozik. Az adott egyed az olyan tevékenységek (állapotátmenetek) alapján írható le, amelyekkel a külső vagy belső eseményekre reagál. Ezzel készíthető a legegyszerűbb, ám sok általános tulajdonság vizsgálatát lehetővé tevő modell. Túlságosan komplex protokolloknál azonban felléphet az állapotrobbanás problémája.
2. Programozási nyelv alapú modell – Egy hálózati protokoll olyan algoritmikus folyamatokat hajt végre, amelyek leírhatók egy magas szintű programozási nyelv elemeivel. Az ilyen modelleket általában nehéz a megfelelő működés szempontjából vizsgálni.
3. Hibrid modell – Az állapot-átmeneti és a programozási nyelv alapú modell képességeit kombinálja a protokoll leírásához. Az állapot-átmeneti rendszert változókkal és

programrészletekkel egészíti ki a nagyobb kifejezőerő érdekében, amelyek így kiterjesztett véges állapotú géppé válik.

1.2. Protokollellenőrzés

A protokollellenőrzés elsődleges célja még a tényleges implementáció előtt megbizonyosodni arról, hogy egy protokoll tervezési hibáktól mentes. A protokoll szolgáltatásspecifikációjával kell a protokollt összevetni, az alábbi módon.

Legyen *P* a szolgáltatásspecifikáció által sugallt eseménysorozatok halmaza, *Q* pedig a szolgáltatási interfészen (a felette lévő réteg felé) megjelenő eseménysorozatok halmaza, amelyet a protokoll egyed-egyed műveletei generálnak. Ez alapján a protokollellenőrzés általánosságban az alábbiak bizonyítását jelenti:

4. $Q \subseteq P$, ahol például a szolgáltatási interfészen előforduló szolgáltatási primitívek minden létező végrehajtását lehetővé teszi a szolgáltatásspecifikáció, valamint
5. $P \subseteq Q$, ahol például a szolgáltatásspecifikációnak eleget tevő szolgáltatási primitívek minden létező végrehajtása megvalósítható a protokoll egyed-egyed műveleteivel.

Q származtatásával a protokoll egyed-egyed műveletei vizsgálhatók, többek között olyan általános tulajdonságok ellenőrizhetők, mint például a teljesség (a protokoll minden egyes rendszerállapotban elfogadja az összes lehetséges bemenő adatot), holtpontmentesség (a protokoll soha nem kerülhet a végállapoton kívül olyan rendszerállapotba, ahonnan nincs továbblépés). Véges futású protokollnál az, hogy egy kiindulási állapotból mindig elérhető egy végállapot, nem véges futású protokollnál pedig ellenőrizhető a ciklikus viselkedés, vagyis minden rendszerállapotból van továbblépés.

1.3. Formális verifikációs módszerek

A protokollellenőrzési módszerek általában kétféle megközelítésmódot használnak: szintézist és analízist. Szintézist jellemzően akkor használunk, ha a protokollt olyan tervezési szabályok alkalmazásával hozzuk létre (az informális specifikációból), amelyek garantálják, hogy az eredményként kapott protokoll rendelkezni fog bizonyos tulajdonságokkal. A kívánt tulajdonságok megléte beépül a tervezési szabályokba.

Analízist akkor használunk, ha a hálózati protokoll specifikációja adott, és a protokoll elemzésével kell igazolni bizonyos elvárt tulajdonságok meglétét. Az állapottér vizsgálata (modellellenőrzés), valamint a programok helyességbizonyítása egyaránt ide tartozik.

3. Az irányítóprotokollok ellenőrzéséhez kapcsolódó tulajdonságok

Az irányítóprotokollok jellemzően az alábbi – formális verifikációs módszerek alkalmazhatóságát befolyásoló – tulajdonságokkal rendelkeznek:

6. Lényegében korlátlan számú, több példányban futó, egyszerű, konkurens folyamatról van szó.
7. Dinamikus összeköttetéseket feltételezünk, és elvárjuk a hibatűrést.
8. A folyamatok szerény komplexitású, diszkrét interfésszel rendelkező, reaktív rendszerek.
9. A valós idejűség alapvető fontosságú, mivel bizonyos tevékenységeknek van elévülési idejük, vagy épp arra kell reagálniuk.

Mivel a tesztelés nem tudja lefedni az összes eshetőséget (ellentétben egy matematikai bizonyítással), a protokollba, illetve annak implementációjába vetett bizalom is jelentősen csökkenne. Bizalomra adhat okot egy olyan módszer használata, amely az automatizált analízis segítségével matematikai ellenőrzésre képes. Itt tulajdonképpen arról van szó, hogy egy eszköz (modellellenőrző) automatikusan generálja az összes végrehajtási ágat (tesztesetet), így igazolva a protokoll megfelelőségét (Maag és Zaidi 2006).

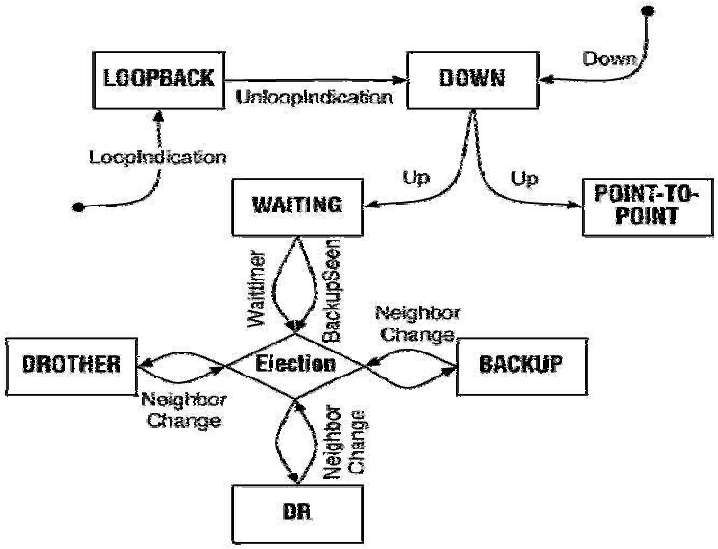
A hálózati protokollok verifikációja nem új keletű dolog. Számos példát láthattunk olyan protokollra, amelynek tervezése során apró hibát vétettek, amely az éles használatig ki sem derült, ott viszont komoly működésbeli zavart okozott.

4. Példa: az OSPF-irányítóprotokoll

Példának az OSPF-et vettük, ami az egyik legszélesebb körben használt belső irányítóprotokoll (IGP). Ennek a 2-es verzióját, ami az IPv4-alapú változatot definiálja, a 2328-as RFC tartalmazza. A dokumentum 9-es és 10-es szekciójában egy-egy állapotgép leírása található meg, konkrétan az interfész-állapotgépé, illetve a szomszédsági viszony-állapotgépé.

Az RFC részletesen leírja az egyes állapotokat, eseményeket, interfésztípusokat, valamint az átmeneteket is jól definiálja. Példánkban a modellezésre és ellenőrzésre a nyílt-forráskódú NuSMV-szoftvert használjuk, ahol CTL-szintaxisban adjuk meg az ellenőrizendő állításokat. Először azonban nézzük meg az RFC alapján felállított modelleket.

- Interfész-állapotgép:
  - Interfész-típusok:
    - Point-to-point
    - Broadcast
    - NBMA
    - Point-to-MultiPoint
    - Virtual link
  - Állapotok:
    - **Down**: kezdeti állapot, semmilyen protokollforgalom nem kerül továbbításra.
    - **Loopback**: az interfész normál forgalmat nem továbbít, csak információszerzésre szolgál.
    - **Point-to-point**: az interfész működőképes, és fizikai pont-pont vagy virtual-link kapcsolatot hoz létre.
    - **Waiting**: a forgalomirányító megpróbálja meghatározni a hálózat DR és BDR eszközeit a Hello csomagok figyelésével.
    - **DR Other**: szórásos vagy NBMA hálózatra csatlakozik az interfész, és nem került kiválasztásra, mint DR vagy BDR. Szomszédsági viszonyt hoz létre a DR és BDR forgalomirányítókkal.
    - **Backup**: BDR-nek választották ki a hálózaton az interfészt. Szomszédsági viszonyba lép a hálózaton lévő összes többi eszközzel.
    - **DR**: kijelölt forgalomirányítónak (Designated Router) választották, és hasonlóan a Backup (tartalék) állapothoz, szomszédsági viszonyba lép az összes többi eszközzel.
  - Események:
    - **InterfaceUp**: a hálózati interfész működőképes.
    - **WaitTimer**: a várakozási időzítő elindul, ennek a végén kerül kiválasztásra a DR és a BDR.
    - **BackupSeen**: a Hello-üzenetek segítségével felfedezett egy BDR-t a hálózaton az interfész.
    - **NeighborChange**: változás történt az interfészhez rendelt szomszédok viszonyában. A DR-t és BDR-t újra kell számolni.
    - **LoopInd**: jelzés, hogy az interfész loopback állapotba vált.
    - **UnloopInd**: jelzés, hogy az interfész kilép a loopback állapotból.
    - **InterfaceDown**: az interfész nem képes tovább működni az alacsonyabb szintű protokollok jelzése alapján.



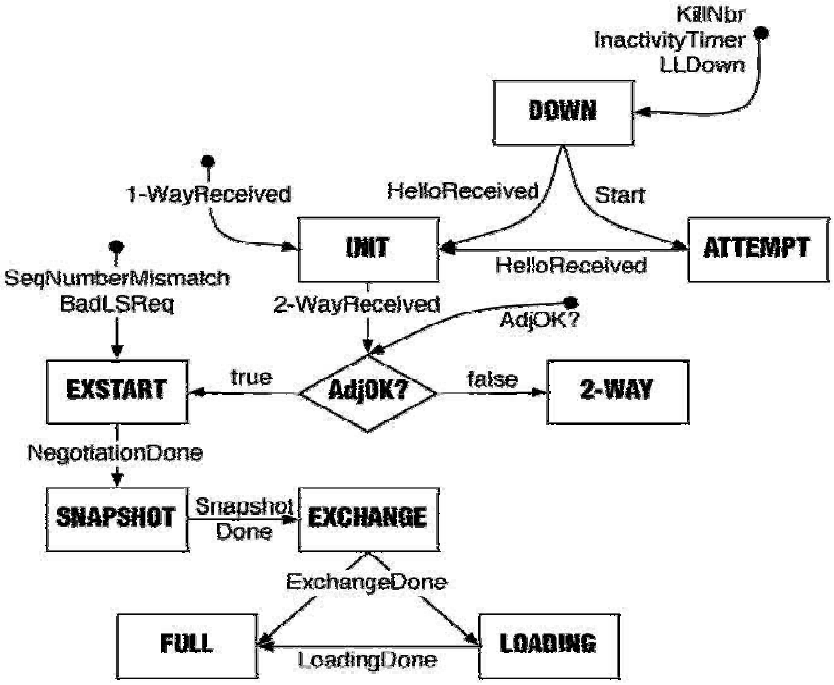
1. ábra Interfész-állapotgép Forrás: RFC 2328

```
MODULE interface
 VAR
  event : {interface_up, wait_timer, backup_seen, neighbor_change, loop_ind, unloop_ind, interface_down};
  state : {down, loopback, waiting, point-to-point, drother, backup, dr};
  type : {point-to-point, broadcast, nbma, point-to-multipoint, virtual_link};
 ASSIGN
  init(type) := {broadcast, point-to-point, nbma, point-to-multipoint, virtual_link};
  next(type) := type;
  init(event) := interface_up;
  next(event) := case
                        state = loopback : {unloop_ind, interface_down};
                        state = down : {interface_up, loop_ind};
                        state = point-to-point : {interface_down, loop_ind};
                        state = waiting : {wait_timer, backup_seen, interface_down, loop_ind};
                        state = drother : {neighbor_change, interface_down, loop_ind};
                        state = backup : {neighbor_change, interface_down, loop_ind};
                        state = dr : {neighbor_change, interface_down, loop_ind};
                        TRUE : event;
            esac;
  init(state) := down;
  next(state) := case
                        state = down & event = interface_up & type in {point-to-point, point-to-multipoint, virtual_link} :
point-to-point;
                        state = down & event = interface_up & type in {nbma, broadcast} : waiting;
                        state = waiting & event = backup_seen : {drother, backup, dr};
                        state = waiting & event = wait_timer : {drother, backup, dr};
                        state in {drother, backup, dr} & event = neighbor_change : {drother, backup, dr};
                        state in {down, loopback, waiting, point-to-point, drother, backup, dr} & event = loop_ind :
loopback;
                        state = loopback & event = unloop_ind : down;
                        state in {down, loopback, waiting, point-to-point, drother, backup, dr} & event =
interface_down : down;
                        TRUE : state;
             esac;
```

- Szomszédsági-állapotgép:
  - Állapotok:
    - **Down**: kezdeti állapot, nincs előzetes információ a szomszédról.

- **Attempt**: (csak NBMA-hálózatok esetén) nem érkezett információ a szomszédoktól, ezért direkt kapcsolatfelvételre van szükség.
- **Init**: Hello-csomag érkezett a szomszédtól, de kétirányú kommunikációra még nem került sor (azaz a forgalomirányító még nem jelent meg a szomszéd Hello-üzenetében).
- **2-Way**: ebben az állapotban a kapcsolat már kétirányú.
- **ExStart**: ez az állapot az első lépés a szomszédsági viszony létrehozásában. Ebben a lépésben kerül meghatározásra a „mester" forgalomirányító a kezdeti DD (Database Description) sorszám alapján.
- **Exchange**: a forgalomirányító a teljes LS adatbázisát átküldi DD csomagok segítségével a szomszédjának.
- **Loading**: ebben az állapotban a forgalomirányító LSR-kéréseket küld a szomszédjának az előző állapotban felfedezett, de még meg nem kapott legújabb LSA-hirdetésekért.
- **Full**: ez az állapot jelenti a teljes szomszédsági viszonyt.

- Események:
  - **HelloReceived:** egy Hello-csomag érkezett a szomszédtól.
  - **Start**: ettől az állapottól kezdve HelloInterval időközönként Hello-csomagokat küld a forgalomirányító a szomszédnak.
  - **2-WayReceived**: kétirányú kapcsolat épült fel egy szomszédos forgalomirányítóval.
  - **NegotiationDone**: a mester/szolga viszony kialakult és a DD sorszámok kicserélése megtörtént a szomszédos forgalomirányítóval.
  - **ExchangeDone**: mindkét forgalomirányító eljuttatta a másikhoz az összes DD-csomagját.
  - **LoadingDone**: megérkezett az összes LS-frissítés a szomszédtól.
  - **BadLSReq**: az adatbázisban nem található LSA-hoz érkezett LS-kérés, ez hibát jelöl az adatbázis-szinkronizáció során.
  - **AdjOk?:** el kell döntenie a forgalomirányítónak, hogy szükséges-e létrehoznia/fenntartania a szomszédsági viszonyt.
  - **SeqNumberMismatch**: egy DD-csomag érkezett váratlan DD-sorszámmal, beállított Init-bittel, vagy az Options mező különbözik a legutóbb kapott DD-csomagétól. Ez szintén hibát jelöl a szomszédsági viszony létrehozásában.
  - **1-Way**: egy olyan Hello-csomag érkezett a szomszédtól, amiben nincs megjelölve a forgalomirányító, azaz (még) nincs kétirányú kapcsolat.
  - **KillNbr**: jelzi, hogy a kommunikáció lehetetlen a szomszéddal, és visszaáll Down állapotba.
  - **InactivityTimer**: az Inactivity Timer elindítása, azaz egy ideje már nem érkezett Hello-csomag a szomszédtól, és visszaáll Down állapotra.
  - **LLDown**: alsóbb szintű protokollok jelzik a szomszéd elérhetetlenségét.



2. ábra Szomszédsági-viszony FSM

```
MODULE neighbor_relationship
 VAR
  state : {down, attempt, init_state, two-way, exstart, exchange, loading, full};
  event : {hello_received, start, two-way_received, negotiation_done, exchange_done, bad_lsr_req, loading_done,
adj_ok_question, seq_number_mismatch, one-way, kill_nbr, inactivity_timer, ll_down};
 ASSIGN
  init(event) := {hello_received, start};
  next(event) := case
                        state = down : {start, hello_received, kill_nbr, ll_down, inactivity_timer};
                        state = attempt : {hello_received, kill_nbr, ll_down, inactivity_timer};
                        state = init_state : {two-way_received, one-way, hello_received, kill_nbr, ll_down, inactivity_timer};
                        state = two-way : {hello_received, adj_ok_question, kill_nbr, ll_down, inactivity_timer, two-
way_received, one-way};
                        state = exstart : {negotiation_done, hello_received, adj_ok_question, kill_nbr, ll_down, inactivity_timer,
two-way_received, one-way};
                        state = exchange : {exchange_done, hello_received, adj_ok_question, seq_number_mismatch,
bad_lsr_req, kill_nbr, ll_down, inactivity_timer, two-way_received, one-way};
                        state = loading : {loading_done, hello_received, adj_ok_question, seq_number_mismatch, bad_lsr_req,
kill_nbr, ll_down, inactivity_timer, two-way_received, one-way};
                        state = full : {hello_received, adj_ok_question, seq_number_mismatch, bad_lsr_req, kill_nbr, ll_down,
inactivity_timer, two-way_received, one-way};
                        TRUE : event;
            esac;
  init(state) := down;
  next(state) := case
                        state = down & event = start : attempt;
                        state = attempt & event = hello_received : init_state;
                        state = down & event = hello_received : init_state;
                        state in {init_state, two-way, exstart, exchange, full, loading} & event = hello_received : state;
                        state = init_state & event = two-way_received : {exstart, two-way};
                        state = init_state & event = one-way : state;
                        state = exstart & event = negotiation_done : exchange;
                        state = exchange & event = exchange_done : {full, loading};
                        state = loading & event = loading_done : full;
                        state = two-way & event = adj_ok_question : {two-way, exstart};
                        state in {exstart, exchange, full, loading} & event = adj_ok_question : {state, two-way};
                        state in {exchange, full, loading} & event = seq_number_mismatch : exstart;
                        state in {exchange, full, loading} & event = bad_lsr_req : exstart;
                        state in {down, init_state, attempt, exstart, two-way, exchange, full, loading} & event = kill_nbr : down;
                        state in {down, init_state, attempt, exstart, two-way, exchange, full, loading} & event = ll_down : down;
                        state in {down, init_state, attempt, exstart, two-way, exchange, full, loading} & event = inactivity_timer :
down;
                        state in {two-way, exstart, exchange, full, loading} & event = two-way_received : state;
                        state in {two-way, exstart, exchange, full, loading} & event = one-way : init_state;
                        TRUE : state;
            esac;
```

A megadott modulokat az NuSMV-ben példányosítani tudjuk, majd CTL-szintaxisú állításokat adhatunk meg, amit a szoftver leellenőriz, és ha hamis eredménnyel tér vissza, akkor egy ellenpéldát is ad. Ezt kihasználva az interfész-állapotgéphez négy, a szomszédsági-állapotgéphez pedig egy állítást adtunk meg. Ránézésre úgy tűnik, hogy az első és a harmadik helyes, a többi pedig hamis.

```
CTLSPEC !EF((interface.type = point-to-point xor interface.type = point-to-multipoint xor interface.type = virtual_link) &
(interface.state = dr xor interface.state = drother xor interface.state = backup));
CTLSPEC !EF((interface.type = point-to-point xor interface.type = point-to-multipoint xor interface.type = virtual_link) &
(interface.state = point-to-point));
CTLSPEC !EF((interface.type = broadcast xor interface.type = nbma) & (interface.state = point-to-point));
CTLSPEC !EF((interface.type = broadcast xor interface.type = nbma) & (interface.state = dr xor interface.state = drother
xor interface.state = backup));

CTLSPEC !EF(neighbor.state = full);
```

Az NuSMV a modell alapján (az előzetes sejtéseknek megfelelően) három állítást hamisnak, kettőt pedig igaznak ítélt a következő kimenettel:

```
-- specification !(EF (((interface.type = point-to-point xor interface.type = point-to-multipoint) xor
interface.type = virtual_link) & ((interface.state = dr xor interface.state = drother) xor interface.state =
backup)))  is true
-- specification !(EF (((interface.type = point-to-point xor interface.type = point-to-multipoint) xor
interface.type = virtual_link) & interface.state = point-to-point))  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  interface.event = interface_up
  interface.state = down
  interface.type = point-to-multipoint
  neighbor.state = down
  neighbor.event = hello_received
-> Input: 1.2 <-
  _process_selector_ = interface
  running = FALSE
  neighbor.running = FALSE
  interface.running = TRUE
-> State: 1.2 <-
  interface.state = point-to-point
-- specification !(EF ((interface.type = broadcast xor interface.type = nbma) & interface.state = point-to-
point))  is true
-- specification !(EF ((interface.type = broadcast xor interface.type = nbma) & ((interface.state = dr xor
interface.state = drother) xor interface.state = backup)))  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 2.1 <-
  interface.event = interface_up
  interface.state = down
  interface.type = broadcast
  neighbor.state = down
  neighbor.event = hello_received
-> Input: 2.2 <-
  _process_selector_ = interface
  running = FALSE
  neighbor.running = FALSE
  interface.running = TRUE
-> State: 2.2 <-
  interface.state = waiting
-> Input: 2.3 <-
-> State: 2.3 <-
  interface.event = wait_timer
-> Input: 2.4 <-
-> State: 2.4 <-
  interface.state = backup
```

```
-- specification !(EF neighbor.state = full)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 3.1 <-
  interface.event = interface_up
  interface.state = down
  interface.type = point-to-point
  neighbor.state = down
  neighbor.event = hello_received
-> Input: 3.2 <-
  _process_selector_ = neighbor
  running = FALSE
  neighbor.running = TRUE
  interface.running = FALSE
-> State: 3.2 <-
  neighbor.state = init_state
-> Input: 3.3 <-
-> State: 3.3 <-
  neighbor.event = two-way_received
-> Input: 3.4 <-
-> State: 3.4 <-
  neighbor.state = exstart
  neighbor.event = hello_received
-> Input: 3.5 <-
-> State: 3.5 <-
  neighbor.event = negotiation_done
-> Input: 3.6 <-
-> State: 3.6 <-
  neighbor.state = exchange
  neighbor.event = adj_ok_question
-> Input: 3.7 <-
-> State: 3.7 <-
  neighbor.event = exchange_done
-> Input: 3.8 <-
-> State: 3.8 <-
  neighbor.state = full
  neighbor.event = bad_lsr_req
```

Sejtésünk tehát beigazolódott. Ez alapján:

- igaz, hogy nincs olyan FSM-állapot, ahol az interfész típusa *pont-pont*, *pont-multipont* vagy *virtual-link* és állapota pedig *DR*, *BDR* vagy *DROther*,
- nem igaz, hogy nincs olyan FSM-állapot (azaz van olyan állapot), ahol az interfész típusa *pont-pont*, *pont-multipont* vagy *virtual-link*, állapota pedig *pont-pont*,
- igaz, hogy nincs olyan FSM-állapot, ahol az interfész típusa *broadcast* vagy *NBMA*, állapota pedig *pont-pont*,
- nem igaz, hogy nincs olyan FSM-állapot (azaz van olyan állapot), ahol az interfész típusa *broadcast* vagy *NBMA*, állapota pedig *DR*, *BDR* vagy *DROther*,
- nem igaz, hogy nincs olyan FSM-állapot (azaz van olyan állapot), ahol a szomszédsági viszony *FULL*. Azaz a modell alapján ebbe az állapotba el tud jutni a véges állapotú gép.

Végül létrehoztunk egy olyan NuSMV-konfigurációt, melyben két OSPF-forgalomirányítót helyezünk el egymással szemben. Ezek egy-egy broadcast interfészen keresztül vannak összekapcsolva, eltérő (OSPF) prioritással az állapotátmenetek demonstrálására.

```
MODULE main
 DEFINE
  r1_priority := 1;
  r2_priority := 2;
 VAR
  r1 : process ospf_router(r1_priority, r2_priority);
  r2 : process ospf_router(r2_priority, r1_priority);
 FAIRNESS
  running;

CTLSPEC !EF(r1.int1.state = dr & r2.int1.state = backup);

MODULE ospf_router(own_priority, other_priority)
 VAR
  int1 : process interface(own_priority, other_priority);
```

A tesztelt állítás pedig, hogy nem létezik olyan végállapot, ahol az R1 DR, az R2 pedig BDR. Ez hamisnak tűnik, mivel az R1-nek nagyobb a prioritása. Az NuSMV-kimenet ezt igazolja:

```
-- specification !(EF (r1.int1.state = dr & r2.int1.state = backup))  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  r1.int1.event = interface_up
  r1.int1.state = down
  r1.int1.type = broadcast
  r2.int1.event = interface_up
  r2.int1.state = down
  r2.int1.type = broadcast
  r2_priority = 2
  r1_priority = 1
-> Input: 1.2 <-
  _process_selector_ = r1.int1
  running = FALSE
  r2.running = FALSE
  r2.int1.running = FALSE
  r1.running = FALSE
  r1.int1.running = TRUE
-> State: 1.2 <-
  r1.int1.state = waiting
-> Input: 1.3 <-
-> State: 1.3 <-
  r1.int1.event = wait_timer
-> Input: 1.4 <-
-> State: 1.4 <-
  r1.int1.state = election
-> Input: 1.5 <-
-> State: 1.5 <-
  r1.int1.state = dr
-> Input: 1.6 <-
  _process_selector_ = r2.int1
  r2.int1.running = TRUE
  r1.int1.running = FALSE
-> State: 1.6 <-
  r2.int1.state = waiting
-> Input: 1.7 <-
-> State: 1.7 <-
  r2.int1.event = wait_timer
-> Input: 1.8 <-
-> State: 1.8 <-
  r2.int1.state = election
-> Input: 1.9 <-
-> State: 1.9 <-
  r2.int1.state = backup
```

5. Konklúzió

A fenti munkákban egy olyan modellt sikerült leírnunk a rendelkezésre álló hivatalos információkból, mely megfelelő módon reprezentálja az OSPF irányítóprotokoll működését. Ezt néhány példaként vett állítással ellenőriztük, ami a várakozásoknak megfelelő eredményeket adta. Ezen kívül egy két-forgalomirányítós konfigurációban demonstráltuk a DR- és BDR-választást.

Ezt a modellt később tovább lehet bővíteni: több-forgalomirányítós konfigurációkat lehet építeni, valamint hitelesítési, LSA-szinkronizációs folyamatokkal/állapotgépekkel további tulajdonságokat lehet ellenőrizni/demonstrálni:

- megfelelő-e a védelem a hitelesítetlen, külső támadok által generált LSA-hirdetések ellen?
- hogyan történik az LSDB szinkronizálása az LSA-hirdetésekkel?

Irodalomjegyzék

RFC 2328: OSPF Version 2 (1998) available at http://www.ietf.org/rfc/rfc2328.txt

Bhargavan, K., Obradovic, D., Gunter, C. A. (2002) Formal Verification of Standards for Distance Vector Routing Protocols. Journal of the ACM (JACM) , Volume 49, Issue 4, pp. 538-576.

Karlin, J., Forrest, S., Rexford, J. (2008) Autonomous security for autonomous systems. Computer Networks, Oct. 2008.

Kent, S., Lynn, C., Seo, K. (2000) Secure border gateway protocol (S-BGP). J. Selected Areas in Communications, vol. 18, pp. 582-592.

Manna, Z., Pnueli, A. (1995) Temporal Verification of Reactive Systems-Safety, Springer.

Lécz, B., Zömbik, L. (2004) Hálózati protokollok biztonsági tesztelése. Híradástechnika, 2004/3: 2-6.

McDaniel, P., Aiello, W., Butler, K., Ioannidis, J. (2006) Origin authentication in interdomain routing. Computer Networks, Nov. 2006.

Sidhu, D., Chung, A., Blumer, T. P. (1991) Experience with formal methods in protocol development. ACM SIGCOMM Computer Communication Review, Volume 21, Issue 2, pp. 81-101.

Goldberg, S., Schapira M., Hummon, P., Rexford, J. (2010) – How secure are secure interdomain routing protocols? SIGCOMM'10, New Delhi, India, ISBN: 978-1-4503-0201-2, pp. 87-98.

Maag, S., Zaidi, F. (2006) – Testing methodology for an ad hoc routing protocol. PM2HW2N'06, Torremolinos , Malaga, Spain, ISBN:1-59593-502-9, pp. 48-55.