ใบรับรองวิทยานิพนธ์

บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์

วิทยาศาสตรมหาบัณฑิต (วิทยาการคอมพิวเตอร์)

ปริญญา

วิทยาการคอมพิวเตอร์ วิทยาการคอมพิวเตอร์

สาขา **ภาควิชา**

เรื่อง วิธีการตรวจสอบการซ้ำกันของเลขหมาย IP ในเครือข่าย IPv6 แบบเคลื่อนที่ด้วยขั้นตอนที่รวดเร็วและทนทานต่อความผิดพลาด

Fast and Robust Duplicate Address Detection (FR-DAD) Method in Mobile IP version 6

นามผู้วิจัย นายพหล โสตถิวิรัช

ได้พิจารณาเห็นชอบโดย

อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก .....

(ผู้ช่วยศาสตราจารย์สุขุมาล กิติสิน, Ph.D.)

อาจารย์ที่ปรึกษาวิทยานิพนธ์ร่วม .....

(อาจารย์พนิตา พงษ์ไพบูลย์, Ph.D.)

อาจารย์ที่ปรึกษาวิทยานิพนธ์ร่วม .....

(ผู้ช่วยศาสตราจารย์ชวลิต ศรีสถาพรพัฒน์, Ph.D.)

หัวหน้าภาควิชา .....

(ผู้ช่วยศาสตราจารย์ศิริกร จันทร์นวล, M.Sc.)

บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์รับรองแล้ว

.....

(รองศาสตราจารย์กัญจนา ธีระกุล, D.Agr.)

คณบดีบัณฑิตวิทยาลัย

วันที่ ..... **เดือน** ..... **พ.ศ.** .....

วิทยานิพนธ์

เรื่อง

วิธีการตรวจสอบการซ้ำกันของเลขหมาย IP ในเครือข่าย IPv6 แบบเคลื่อนที่ ด้วยขั้นตอนที่รวดเร็วและทนทานต่อความผิดพลาด

Fast and Robust Duplicate Address Detection (FR-DAD) Method in Mobile IP version 6

โดย

นายพหล โสตถิวิรัช

เสนอ

บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์
เพื่อความสมบูรณ์แห่งปริญญาวิทยาศาสตรมหาบัณฑิต (วิทยาการคอมพิวเตอร์)
พ.ศ. 2551

พหล โสตถิวิรัช 2551: วิธีการตรวจสอบการซ้ำกันของเลขหมาย IP ในเครือข่าย IPv6 แบบเคลื่อนที่ด้วยขั้นตอนที่รวดเร็วและทนทานต่อความผิดพลาด ปริญญาวิทยาศาสตรมหาบัณฑิต (วิทยาการคอมพิวเตอร์) สาขาวิทยาการคอมพิวเตอร์ ภาควิชาวิทยาการคอมพิวเตอร์ อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก: ผู้ช่วยศาสตราจารย์สุขุมาล กิติสิน, Ph.D. 196 หน้า

ในเครือข่าย IPv6 แบบเคลื่อนที่ (Mobile IPv6 network) ขั้นตอนการตรวจสอบการซ้ำกันของเลขหมาย (DAD: Duplicate Address Detection) เป็นขั้นตอนที่จำเป็นในการยืนยันว่าไม่มีการซ้ำกันของการใช้เลขหมาย IPv6 เมื่อ Mobile Node (MN) เคลื่อนย้ายไปที่เครือข่ายใหม่ การทำงานตามมาตรฐานนั้นใช้เวลาของขั้นตอน DAD เป็นเวลาอย่างน้อย 1 วินาที การลดเวลาการทำงานลงมีความสำคัญถ้ามีการใช้แอปพลิเคชั่นที่ต้องการการส่งข้อมูลแบบทันที (Real-time application) ในเครือข่าย IPv6 แบบเคลื่อนที่ งานวิจัยนี้ได้นำเสนอวิธีขั้นตอน DAD วิธีใหม่ที่เรียกว่า Fast and Robust DAD (FR-DAD) เพื่อช่วยลดเวลาที่ใช้สำหรับการ handover และออกแบบโพรโตคอลให้สามารถเข้ากันได้กับการทำงานของมาตรฐานที่ใช้จริงสำหรับเครือข่าย IPv6 และมีความทนทานต่อความผิดพลาด FR-DAD เป็นระบบการจ่ายเลขหมายแบบกึ่งเต็มสถานะ (semi-stateful) ซึ่งมีเซิร์ฟเวอร์จัดการเก็บและแจกจ่ายเลขหมาย IPv6 ที่ได้รับการตรวจสอบแล้ว

ผลการทดลองจากโปรแกรม OMNET++ Simulator พบว่า FR-DAD สามารถลดเวลาในขั้นตอนการตรวจสอบ DAD ได้ 99.74% เมื่อเทียบกับการทำงานของ DAD แบบมาตรฐานสำหรับกรณีที่ไม่มีความผิดพลาดหรือใช้เวลาน้อยที่สุด และเมื่อเทียบกับวิธีขั้นตอน A-DAD ที่มีการพัฒนาขึ้นมาแล้ว ในกรณีที่มีความผิดพลาดหรือใช้เวลามากที่สุด FR-DAD ใช้เวลาน้อยกว่าที่ 23% เมื่อเทียบกับวิธีดังกล่าว

_____ _____ ___ / ___ / ___

ลายมือชื่อนิสิต ลายมือชื่ออาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

Pahol Sotthivirat 2008: Fast and Robust Duplicate Address Detection (FR-DAD) Method in Mobile IP version 6. Master of Science (Computer Science), Major Field: Computer Science, Department of Computer Science. Thesis Advisor: Assistant Professor Sukumal Kitisin, Ph.D. 196 pages.

In Mobile IPv6 networks, duplicate address detection (DAD) process is necessary for confirming uniqueness of IPv6 address when Mobile Node moves to a new network. In standard IPv6 protocol, DAD delay takes at least 1 second. It is desirable to reduce DAD delay especially if nodes wish to run real-time applications on the Mobile IPv6 networks. This research proposed the new DAD Method, Fast and Robust DAD (FR-DAD), to improve handover delay and provide reliability and backward compatibility with standard IPv6 networks. FR-DAD is a semi-stateful address assignment system. It utilizes a server that manages a list of unique IPv6 addresses.

The experiment results from OMNET++ simulator show that FR-DAD successfully reduces delay by 99.74% of standard DAD delay, in best case. In the worst case, FR-DAD still outperforms a similar stateful DAD technique by 23%.

_____ _____ ___ / ___ / ___

Student's signature Thesis Advisor's signature

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้เป็นการงานวิจัยร่วมกับศูนย์เทคโนโลยีอิเลิกทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC: National Electronics and Computer Technology Center) ซึ่งสำเร็จลุล่วงได้ดีด้วยความช่วยเหลืออย่างดียิ่งของ อาจารย์พนิตา พงษ์ไพบูลย์ กรรมการที่ปรึกษาร่วมจาก NECTEC ที่ได้สั่งสอน ให้คำแนะนำและคำปรึกษา เกี่ยวกับทฤษฎี ความรู้และการออกแบบต่างๆ ชี้แนวทางแก้ไขปัญหาและตรวจแก้ข้อบกพร่องต่างๆ ในงานวิจัยตลอดมา ขอกราบขอบพระคุณ ผู้ช่วยศาสตราจารย์ สุขุมาล กิติสิน ประธานกรรมการที่ปรึกษา และผู้ช่วยศาสตราจารย์ ชวลิต ศรีสถาพรพัฒน์ กรรมการที่ปรึกษาร่วม ที่ให้ความช่วยเหลืออย่างดียิ่งในด้านวิธีการและขั้นตอนในการดำเนินงาน ให้คำแนะนำ และข้อเสนอแนะ ในการทำวิทยานิพนธ์นี้

ขอกราบขอบพระคุณ คุณพ่อสรรชัย โสตถิวิรัช และคุณแม่จันทร์เพ็ญ โสตถิวิรัช ที่เลี้ยงดูมาเป็นอย่างดีโดยตลอดและให้การสนับสนุนในศึกษาต่อปริญญาโทจนจบการศึกษา และขอขอบคุณทุกคนในครอบครัวที่ได้ให้การสนับสนุน และความช่วยเหลืออย่างที่สุดมาโดยตลอดจนสำเร็จการศึกษา ขอขอบพระคุณคณาจารย์ทุกท่านที่ให้การอบรม สั่งสอนวิชาความรู้ต่างๆ มาจนสำเร็จการศึกษา

ขอขอบคุณเพื่อนรุ่นที่ 6 และรุ่นพี่ทุกคนที่ช่วยเหลือ ให้คำแนะนำอย่างดีมาตลอด รวมถึงทุกสิ่งทุกอย่างที่ทำให้วิทยานิพนธ์นี้สำเร็จลุล่วงไปได้ด้วยดี และขอขอบคุณโครงการปริญญาโทภาคที่ได้สนับสนุนด้านอุปกรณ์ ทุนไปเสนอผลงานวิชาการ และสถานที่ตลอดระยะเวลาในการทำวิทยานิพนธ์

พหล โสตถิวิรัช

สิงหาคม 2551

สารบัญ

หน้า

สารบัญตาราง

สารบัญภาพ

สารบัญภาพ (ต่อ)

คำอธิบายสัญลักษณ์และคำย่อ

A-DAD	=	Advance Duplicate Address Detection
AP	=	Access Point
AR	=	Access Router
BA	=	Binding Acknowledgement
BSSID	=	Access Point's basic service set ID
BU	=	Binding Update
CN	=	Correspond Node
CoA	=	Care-of Address
DAD	=	Duplicate Address Detection
DHCP	=	Dynamic Host Configuration Protocol
DHCPv6	=	Dynamic Host Configuration Protocol for IPv6
DUID	=	DHCP Unique Identifier
DUID-EN	=	DUID Assigned by Vendor Based on Enterprise Number
DUID-LIT	=	DUID Based on Link-layer Plus Time
DUID-LL	=	DUID Based on Link-layer
EUI64	=	64-bit extended unique identifier
FN	=	Foreign Network
FR-DAD	=	Fast and Robust Duplicate Address Detection
FSM	=	Finite State Machine
HA	=	Home Agent
HN	=	Home Network
HoA	=	Home Address
IA_NA	=	Non-temporary Address Option
IA_TA	=	Identifier Association for Temporary Address Option
IAID	=	Identifier Association ID
ICMP	=	Internet Control Message Protocol
ICMPv6	=	Internet Control Message Protocol for IPv6
IP	=	Internet Protocol

คำอธิบายสัญลักษณ์และคำย่อ (ต่อ)

IPv4	=	Internet Protocol version 4
IPv6	=	Internet Protocol version 6
LAN	=	Local Area Network
LLA	=	Link Layer Address
MAC	=	Media Access Control
MAP	=	Mobile-Node Attachment point
MN	=	Mobile Node
MTU	=	Maximum Transmission Unit
NA	=	Neighbor Advertisement
NAR	=	New Access Router
NAT	=	Network Address Translation
NCoA	=	New Care-of Address
ND	=	Neighbor Discovery
NS	=	Neighbor Solicitation
O-DAD	=	Optimistic Duplicate Address Detection
PAR	=	Previous Access Router
PCoA	=	Previous Care-of Address
P-DAD	=	Proactive Duplicate Address Detection
RA	=	Router Advertisement
ranMN	=	random IPv6 address for Mobile Node
ranSer	=	random IPv6 address for P-Server
RFC	=	Request for Comments
RIP	=	Regional Information Point
RS	=	Router Solicitation
UDP	=	User Datagram Protocol
UHF	=	Ultra High Frequency
WLAN	=	Wireless Local Area Network

วิธีการตรวจสอบการซ้ำกันของเลขหมาย IP ในเครือข่าย IPv6 แบบเคลื่อนที่ด้วยขั้นตอนที่รวดเร็วและทนทานต่อความผิดพลาด

Fast and Robust Duplicate Address Detection (FR-DAD) Method in Mobile IP version 6

คำนำ

ในปัจจุบันอินเทอร์เน็ตมีการใช้งานอย่างแพร่หลาย เพราะเป็นแหล่งที่มาของข้อมูล และความรู้ต่างๆที่บุคคลทั่วไปสามารถรับรู้ข่าวสารได้อย่างรวดเร็ว ในระดับที่อยู่อาศัยเพียงมีคอมพิวเตอร์หรืออุปกรณ์เชื่อมต่อกับผู้ให้บริการอินเทอร์เน็ตหรือ ISP ผ่านสายโทรศัพท์ก็สามารถใช้งานอินเทอร์เน็ตได้ หรือถ้าอยู่ในองค์กร บริษัทหรือมหาวิทยาลัยก็สามารถเชื่อมต่อผ่านเครือข่าย LAN (Local Area Network) แต่เมื่อมองในแง่มุมของความคล่องตัวของการใช้งานอินเทอร์เน็ตแล้ว เห็นได้ว่าสามารถใช้ได้เฉพาะจุดที่อยู่ใกล้กับสายโทรศัพท์ หรือสายสัญญาณ ซึ่งถ้าต้องการเคลื่อนย้ายหรือใช้ในพื้นที่ที่ต้องการอาจจะต้องเพิ่มความยาวของสายสัญญาณ และถ้ามีจำนวนเครื่องมากขึ้นการจัดสรรจุดเชื่อมต่อให้เพียงพอต่อความต้องการของบุคคลนั้นเป็นเรื่องที่ยุ่งยาก จากปัญหาดังกล่าว เทคโนโลยีเครือข่ายไร้สาย (Wireless Network) ถูกคิดค้นขึ้นในปี ค.ศ. 1979 เพื่อเพิ่มความคล่องตัวและลดความยุ่งยากของการวางสายสัญญาณโดยการเพิ่มอุปกรณ์ส่ง-รับโดยแปลงสัญญาณจากสายสัญญาณของเครือข่าย LAN เป็นคลื่นแม่เหล็กไฟฟ้าเรียกว่า Access Point (AP) และเพื่อให้สอดคล้องกับระบบเครือข่าย LAN เดิมตามมาตรฐาน IEEE 802.3 Ethernet จึงได้มีการกำหนดเป็นมาตรฐานของ LAN ไร้สายหรือ Wireless LAN (WLAN) IEEE 802.11 ขึ้น WLAN มีการใช้งานที่สะดวกสบาย ทำให้มีผลิตภัณฑ์สนันสนุนการทำงานได้แก่ คอมพิวเตอร์มีอุปกรณ์ที่รองรับการเชื่อมต่อ WLAN หรือ WLAN Adapter เป็นอุปกรณ์มาตรฐานในปัจจุบัน และการเพิ่มจุดรับ-ส่งสัญญาณ AP มีอย่างครอบคลุม ความคล่องตัวของการใช้งานอินเทอร์เน็ตเพิ่มขึ้นเพราะถ้าเป็นที่ที่มีสัญญาณ WLAN จาก AP ก็สามารถใช้งานอินเทอร์เน็ตได้ทุกที่

เมื่อการใช้งานอินเทอร์เน็ตเริ่มสามารถใช้ได้ทุกที่ที่มี AP แล้ว ปัญหาที่ตามมาคือแอปพลิเคชั่นที่ต้องการการส่งข้อมูลแบบทันที (Real-time application) นั้น ถ้าผู้ใช้ต้องการให้สามารถทำงานได้ทุกที่ขณะที่ใช้งานอินเทอร์เน็ตอยู่ไม่ว่าเคลื่อนที่ไปติดต่อกับ AP ใดก็ตาม ปัญหาจะเกิดขึ้นที่เทคโนโลยีระดับเครือข่าย (Network layer) คือเลขหมาย IP ไม่เพียงพอ เนื่องจากการใช้

งานเลขหมาย IP ในปัจจุบันใช้เลขหมาย IP เวอร์ชั่น 4 (IPv4) ใช้เลขหมายทั้งหมดขนาด 32 บิต แบ่งเป็น 8 บิต 4 ส่วน ถ้าทุกคนต้องการเลขหมาย IP ที่ไม่ซ้ำกันเพื่อใช้อ้างอิงตัวอุปกรณ์ได้อย่างเฉพาะเจาะจงแล้วการจัดสรรเลขหมายนั้นจะไม่เพียงพอต่อความต้องการของแต่ละบุคคล ซึ่งแก้ไขได้โดยใช้เทคนิคการแปลงเลขหมาย (Network Address Translation: NAT) โดยในองค์กรหนึ่งจะได้เลขหมาย IP จริงน้อยกว่าจำนวนผู้ใช้ภายในองค์กรมากจะใช้เลขหมายกลุ่มภายใน (Private IP) ตามมาตรฐาน IPv4 ผู้ใช้สามารถใช้งานอินเทอร์เน็ตได้อย่างปกติ แต่เมื่อสังเกตที่เลขหมาย IP อุปกรณ์ภายนอกจะรู้จักเพียงเลขหมาย IP จริงเท่านั้น ทำให้อุปกรณ์ภายนอกไม่สามารถระบุตัวตนของคู่สื่อสารที่อยู่ภายในองค์กรที่ใช้เทคนิค NAT ถ้าเครือข่ายภายในองค์กรเป็น WLAN และผู้ใช้เคลื่อนที่ออกจาก AP ภายในองค์กรไปยัง AP ภายนอกองค์กรโดยที่มีการใช้งาน Real-time application อยู่นั้น การสื่อสารจะต้องหยุดไปเนื่องจากการใช้เลขหมาย IP ไม่เป็นเลขหมาย IP จริง อาจต้องได้เลขหมาย IP ของเครือข่ายใหม่ก่อนแล้วจึงทำขั้นตอนระบุตัวตนเพื่อเชื่อมต่อกับคู่สื่อสารเดิมหรือต้องเชื่อมต่อใหม่เลย ซึ่งล้วนแล้วแต่เสียเวลาในการเชื่อมต่อไม่สอดคล้องกับลักษณะงานที่ต้องการส่งข้อมูลตลอดเวลา สามารถแก้ไขได้โดยใช้งานเลขหมาย IP เวอร์ชั่น 6 (IPv6) ที่มีขนาด 128 บิตที่มีเลขหมายได้ทั้งหมด 2^{128} เลขหมาย เพียงพอกับอุปกรณ์แต่ละตัว และเมื่อนำลักษณะการทำงานของอุปกรณ์ที่สามารถทำงานและเคลื่อนที่ไปพร้อมๆ กันในเครือข่าย WLAN โดยไม่ว่าจะเคลื่อนที่ข้ามไปยังกลุ่มเครือข่ายใดที่เป็น WLAN ก็ยังไม่ขาดการติดต่อสื่อสารนี้แล้ว ได้กำหนดเป็นหลักการของเทคโนโลยีใหม่คือ Mobile IP ซึ่งออกแบบสำหรับการใช้งานเลขหมาย IPv6 โดยเฉพาะเรียกว่า เครือข่ายแบบเคลื่อนที่ที่ใช้เลขหมาย IPv6 (Mobile IPv6 Network)

ในระบบของเครือข่ายแบบเคลื่อนที่ที่ใช้เลขหมาย IPv6 (Mobile IPv6 Network) นี้ มีลักษณะการทำงานเมื่อเคลื่อนที่ข้ามเครือข่ายของอุปกรณ์ไร้สายเคลื่อนที่ได้คล้ายกับการทำงานของเดิม คือต้องใช้งานเลขหมาย IP ในกลุ่มของเครือข่ายนั้น แต่ระบบเลขหมาย IPv6 สามารถมีเลขหมาย IPv6 มากกว่าหนึ่งเลขหมายในหนึ่งอุปกรณ์ เลขหมาย IPv6 จริงของอุปกรณ์จะเชื่อมต่อกับเลขหมาย IPv6 ของอุปกรณ์ที่ใช้ภายในเครือข่าย ทุกอุปกรณ์จะทราบการเชื่อมต่อนี้จากการแลกเปลี่ยนข้อมูลผ่านทางกลุ่มข้อความ ICMPv6 ทำให้อุปกรณ์ของผู้ใช้สามารถเคลื่อนที่ไปที่ใดก็ได้โดยไม่ขาดการติดต่อ แต่เมื่อพิจารณาของการกำหนดเลขหมาย IPv6 ภายในเครือข่ายยังเป็นปัญหาเรื่องของการใช้เวลาการกำหนดเลขหมาย IPv6 นานจนการสื่อสารที่กำลังเชื่อมต่อหยุดไปขณะหนึ่ง เพราะเมื่อใช้เลขหมาย IPv6 แล้วในเครือข่ายหนึ่งสามารถรองรับจำนวนลูกข่ายได้มากมาย ซึ่งถ้าเป็นเลขหมาย IPv4 มีเทคนิคที่นิยมเป็นการจ่ายเลขหมาย IP แบบอัตโนมัติโดยมี server เป็นอุปกรณ์จัดสรรเลขหมาย IP ให้จ่ายเลขหมาย IP โดยไม่ซ้ำกันเรียกเทคนิคนี้ว่า Dynamic

Host Configuration Protocol (DHCP) แต่ถ้าเลขหมาย IPv6 ใช้เทคนิคนี้ server จะทำงานหนัก เพราะต้องจัดสรรเลขหมาย IPv6 จำนวนมากที่มีการเข้า-ออกจากเครือข่ายอยู่ตลอดเวลา จึงไม่เหมาะสมกับเครือข่ายที่ใช้เลขหมาย IPv6

เทคนิคที่ใช้จัดสรรเลขหมาย IPv6 ที่ไม่มีศูนย์กลางการทำงาน ทำงานโดยอุปกรณ์ที่ต้องการเลขหมาย IPv6 ภายในเครือข่ายจะกำหนดเลขหมาย IPv6 ขึ้นมาเลขหมายหนึ่งและใช้ข้อความถามทุกอุปกรณ์ภายในเครือข่ายว่ามีการใช้เลขหมาย IPv6 หรือไม่ โดยใช้กลุ่มข้อความ ICMPv6 ของ Neighbor Discovery สำหรับ IPv6 RFC4861 (T. Narten et al., 2007) รอคำตอบจากอุปกรณ์ที่ใช้เลขหมาย IPv6 นี้แล้วเป็นเวลา 1000 มิลลิวินาที RFC4862 (S. Thomson et al., 2007) ถ้าไม่มีอุปกรณ์ใดในเครือข่ายใช้เลขหมาย IPv6 นี้ จะกำหนดใช้เลขหมาย IPv6 นี้ทันทีหลังจากเวลาผ่านไป 1000 มิลลิวินาที การจัดสรรเลขหมาย IPv6 ขั้นตอนนี้เรียกว่า Duplicate Address Detection (DAD) ใช้การติดตั้งเลขหมายอัตโนมัติแบบไม่เก็บสถานะ (Stateless Address Autoconfiguration) เหมาะสมกับเครือข่ายที่ใช้เลขหมาย IPv6 ที่มีจำนวนลูกข่ายมาก ซึ่งช่วงเวลารอคำตอบ 1000 มิลลิวินาทีนี้ เป็นปัจจัยหนึ่งที่เป็น delay time ของการเคลื่อนย้ายข้ามเครือข่าย ดังนั้นถ้าสามารถลดเวลารอคำตอบนี้ได้ก็จะลด delay time ของการเคลื่อนย้ายข้ามเครือข่ายโดยรวมลงได้ เป็นที่มาของการทำวิทยานิพนธ์ฉบับนี้ ซึ่งการพัฒนาขั้นตอน Duplicate Address Detection (DAD) ได้มีผู้คิดค้นหลากหลายวิธีออกมาเป็นบทความที่มีแนวความคิดต่างๆกันออกไป ในวิทยานิพนธ์นี้ขอเสนอขั้นตอนที่ลดเวลาทำ Duplicate Address Detection (DAD) ในอีกแง่มุมหนึ่ง ซึ่งคำนึงถึงการลดเวลาขั้นตอน Duplicate Address Detection (DAD) และวิเคราะห์ประโยชน์แวดล้อมด้านอื่นๆ นอกเหนือจากการลดเวลาการทำงาน โดยวัดประสิทธิภาพของงานวิจัยอาศัยการจำลองสถานการณ์การทำงานของเครือข่าย ด้วยโปรแกรมจำลองเครือข่าย Omnet++

วัตถุประสงค์

1. เพื่อปรับปรุงวิธีการลดระยะเวลาในการตรวจสอบการซ้ำกันของเลขหมาย IPv6 ในเครือข่าย
2. เพื่อเปรียบเทียบประสิทธิภาพในเรื่องของเวลาการทำงานและผลดี-ผลเสียด้านต่างๆ ของวิธีการลดระยะเวลาในการตรวจสอบการซ้ำกันของเลขหมาย IPv6 กับวิธีอื่นๆ

ประโยชน์ที่คาดว่าจะได้รับ

1. ได้องค์ความรู้ใหม่เกี่ยวกับทฤษฎีขั้นตอนการทำงานของเครือข่าย IPv6 แบบเคลื่อนที่ วิธีการ Duplicate Address Detection (DAD) ต่างๆ
2. สามารถนำวิธีที่ได้พัฒนาขึ้นไปใช้งานในระบบให้เกิดประสิทธิภาพสูงสุดตามความเหมาะสม
3. สามารถเป็นแนวคิดเพื่อใช้ในการพัฒนาต่อไปได้

ขอบเขตและข้อจำกัด

1. การศึกษาการทำงานของงานวิจัยโดยใช้โปรแกรมจำลองการทำงาน Omnet++
2. การศึกษาการทำงานของวิธีการ Duplicate Address Detection (DAD) วิธีต่างๆ บนเครือข่าย IPv6 แบบเคลื่อนที่
3. ใช้ความเร็วของการได้รับเลขหมาย IPv6 จากวิธีการตรวจสอบ DAD และความเหมาะสมในหลายๆด้าน เป็นตัวแปรในการเปรียบเทียบวิธีต่างๆ
4. ไม่รวมถึงปัญหาความคับคั่งของเครือข่าย

การตรวจเอกสาร

ทฤษฎีพื้นฐานเกี่ยวกับงานวิจัย

เนื่องจากงานวิจัยนี้เป็นการพัฒนาขั้นตอน Duplicate Address Detection (DAD) บนเครือข่ายแบบเคลื่อนที่ที่ใช้เลขหมาย IPv6 (Mobile IPv6 Network) หรือเครือข่ายที่ใช้เลขหมาย IPv6 แบบทั่วไป ดังนั้นในหัวข้อนี้จะขอกล่าวถึงทฤษฎีพื้นฐานที่เกี่ยวข้องอันได้แก่ ระบบเลขหมาย IPv6 เครือข่ายแบบเคลื่อนที่ที่ใช้เลขหมาย IPv6 (Mobile IPv6 Network) ข้อความต่างๆในกลุ่มของ Neighbor Discovery สำหรับ IPv6 ขั้นตอน Duplicate Address Detection (DAD) มาตรฐานที่กำหนดไว้ใน Stateless Address Autoconfiguration สำหรับเลขหมาย IPv6

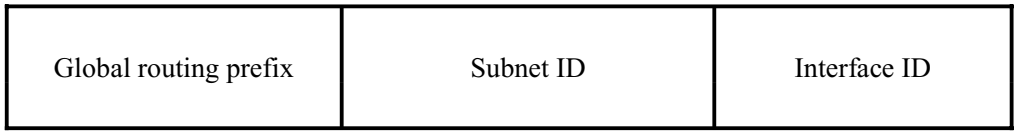
ระบบเลขหมาย IPv6

เลขหมาย IPv6 (Silvia, 2002) ที่ใช้กับอยู่นี้มีระบบการแบ่งชนิดของเลขหมาย และความหมายของเลขหมายแต่ละตำแหน่ง เลขหมาย IPv6 ประกอบด้วยตัวเลขทั้งหมด 128 บิต แบ่งเป็นเลขฐานสิบหกกลุ่มละ 4 ตัว ทั้งหมด 8 กลุ่ม โดยมีเครื่องหมาย ":" คั่นระหว่างกลุ่ม เช่น

FE80:0000:0000:0000:0202:B3FF:FE1E:8329 ……(a)

ในกรณีที่ค่าของกลุ่มแต่ละกลุ่มมีค่าเป็นศูนย์หรือ "0000" สามารถละเว้นได้โดยการใช้ "::" ในการละเว้น และถ้ามีกลุ่มที่มีค่าเป็นศูนย์ติดกันสามารถละเว้นได้เลย จากตัวอย่างเลขหมาย (a) สามารถลดรูปเลขหมายได้เป็น

FE80::0202:B3FF:FE1E:8329



ภาพที่ 1 แสดงส่วนประกอบของเลขหมาย IPv6

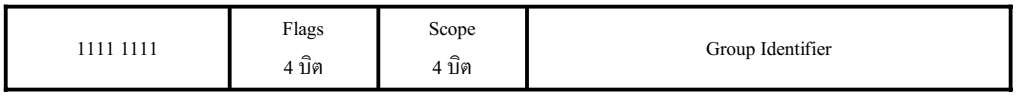
นอกจากนี้ยังมีการแบ่งส่วนประกอบของเลขหมาย IPv6 ทั้ง 128 บิต ออกเป็น 3 ส่วนดังภาพที่ 1 คือ

(1) Global routing prefix เป็นกลุ่มของ Prefix ในส่วนมหาภาค ได้แก่ แต่ละ ISP

(2) Subnet ID ใช้รวมกับ Global routing prefix เป็นการระบุกลุ่มของ Subnet แยกย่อยอีกทีหนึ่ง

(3) Interface ID เป็นส่วนที่แต่ละอุปกรณ์จะซ้ำกันไม่ได้

ในระบบเลขหมาย IPv6 นี้มีการกำหนดชนิดของเลขหมาย IPv6 คล้ายกับของ IPv4 โดยเลขหมาย IPv6 ที่ใช้เป็นเลขหมายจริงในการติดต่อสื่อสารเครือข่าย IPv6 สามารถใช้ได้ทั่วทุกเครือข่ายและมีเพียงอุปกรณ์เดียวในหนึ่งเลขหมายที่สามารถใช้ได้ หรือเป็นเลขหมายที่ไม่มีการใช้ซ้ำกัน เรียกเลขหมาย IPv6 ชนิดนี้ว่า Unicast address เมื่อเปรียบเทียบกับระบบเลขหมาย IPv4 เป็นข้อได้เปรียบของระบบที่ใช้เลขหมาย IPv6 ที่สามารถระบุตัวตนของอุปกรณ์ได้ทุกอุปกรณ์เพราะจำนวนเลขหมายที่รองรับนั้นมีมากเพียงพอ



ภาพที่ 2 แสดงส่วนประกอบของ Multicast Address

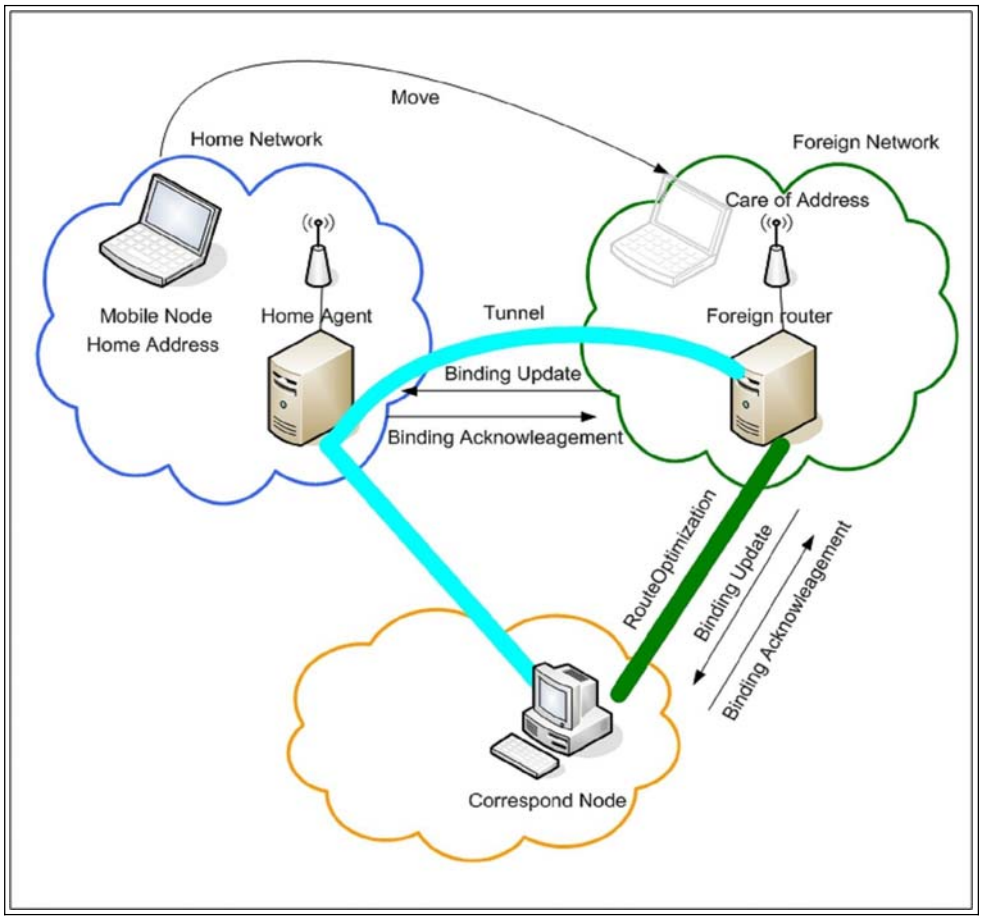
เลขหมาย IPv6 อีกชนิดหนึ่งที่ใช้ในการติดต่อสื่อสารกับอุปกรณ์ต่างๆ ในเครือข่าย IPv6 เพื่อแลกเปลี่ยนข้อมูลและพารามิเตอร์สำหรับการเชื่อมต่อคือ Multicast address ที่มีการแบ่งกลุ่มของอุปกรณ์ในการติดต่อสื่อสารภายในกลุ่ม ซึ่งในหนึ่งอุปกรณ์สามารถอยู่ได้หลายกลุ่ม สามารถส่งข้อมูลแบบระบุอุปกรณ์เดียวหรือส่งไปยังทุกอุปกรณ์ในกลุ่มก็ได้ มีส่วนสำคัญในการช่วยสื่อสารภายในเครือข่ายก่อนได้รับ Unicast Address ลักษณะของเลขหมาย IPv6 ชนิดนี้มีข้อกำหนดของรูปแบบตามมาตรฐานดังภาพที่ 2 โดย 8 บิตแรกเป็น “1” ทั้งหมดหรือ “FF” ในฐานสิบหกบอกว่าเป็นเลขหมาย IPv6 ชนิด Multicast address 4 บิตถัดมาเป็นส่วนของ Flags ซึ่งถ้าเป็น “0000” หมายถึงกลุ่ม Multicast address ตามมาตรฐานสากลเป็นที่ตกลงกันว่าทุกเครือข่ายที่ใช้ระบบ Multicast address ทราบโครงสร้างถัดมาได้หรือ well-know multicast address และ “0001” หมายถึงกลุ่มที่สำรองไว้ใช้หรือ temporary multicast address ไม่มีการกำหนดโครงสร้างไว้ และ 4 บิตถัดมาเป็นส่วนของ Scope ในกรณีที่มี Flags เป็น “0000” จะเป็นการแบ่งกลุ่มของเครือข่าย

multicast address ได้แก่ Link-local scope “2” ในฐานสิบหก ใช้ในกลุ่มอุปกรณ์ที่อยู่ภายในเครือข่ายเดียวกัน และบิตที่เหลือเป็นส่วนของ Group ID ทำหน้าที่คล้ายกับ Interface ID ของ Unicast address แต่มีการอ้างอิงเป็นกลุ่มได้ ตามมาตรฐานสากลนั้น ถ้ากลุ่มของเลขหมายกลุ่มทางขวาสุดเป็น “1” หมายถึงอุปกรณ์ทุกตัวที่อยู่ในกลุ่ม และเป็น “2” หมายถึงอุปกรณ์ที่เป็น router ทุกตัวที่อยู่ในกลุ่ม

จากเลขหมาย IPv6 แต่ละชนิดที่ได้กล่าวไปนั้น มีกลุ่มของเลขหมาย IPv6 ที่ใช้ในการทำวิจัย ได้แก่ Link-local unicast address (FE08::/10) เป็นกลุ่มของเลขหมาย IPv6 ชนิด Unicast address ที่ใช้ภายในเครือข่ายเดียวกันเท่านั้น ซึ่งมีลักษณะเหมือนกับ Private IP address ในระบบเลขหมาย IPv4 ใช้ subnet mask 10 บิต ใช้ในขั้นตอน Duplicate Address Detection (DAD) สำหรับทดสอบ Interface ID ก่อนที่ได้ Prefix ของเครือข่ายนั้นมา และเลขหมาย IPv6 อีกกลุ่มคือ Link-local scope multicast address (FF02::/8) เป็นกลุ่มของเลขหมาย IPv6 ชนิด Multicast address ใช้ร่วมกับข้อความ ICMPv6 ของ Neighbor Discovery สำหรับ IPv6 ในขั้นตอน Duplicate Address Detection (DAD) และขั้นตอนอื่นๆ โดยมี FF02::1 เป็นเลขหมาย IPv6 ที่ใช้ส่งทุกอุปกรณ์ในเครือข่าย และ FF02::2 เป็นเลขหมาย IPv6 ที่ใช้ส่งทุกอุปกรณ์ที่เป็น router ในเครือข่าย ส่วนในเลขหมาย IPv6 ส่วนอื่นจะกล่าวถึงในเนื้อที่เกี่ยวข้องหลังจากนี้

เครือข่ายแบบเคลื่อนที่ที่ใช้เลขหมาย IPv6 (Mobile IPv6 Network)

เครือข่ายแบบเคลื่อนที่ที่ใช้เลขหมาย IPv6 (Mobile IPv6 Network) ถูกกำหนดขึ้นเป็นมาตรฐาน RFC 3775 (D. Johnson et al., 2004; Wolfgang and Florian, 2000) เพื่อรองรับความต้องการของเทคโนโลยีในปัจจุบันและอนาคต ใช้ระบบเลขหมาย IPv6 เป็นพื้นฐาน ลูกข่ายสามารถเคลื่อนที่ไปได้ทุกแห่งตลอดเวลาที่มี AP แม้ว่าจะเป็นเครือข่ายที่มีเลขหมาย IPv6 ต่าง prefix กันก็ยังทำงานได้ปกติและระบุตัวตนของอุปกรณ์ได้ด้วยเลขหมาย IPv6 จริง จากคุณสมบัติพื้นฐานของเครือข่ายแบบเคลื่อนที่ที่ใช้เลขหมาย IPv6 (Mobile IPv6 Network) นี้ ทำให้ต้องมีขั้นตอนการส่ง-รับข้อมูล การเตรียมความพร้อมของอุปกรณ์เมื่อเคลื่อนย้ายไปเครือข่ายใหม่ที่มีกลุ่มเลขหมาย IPv6 และโครงสร้างของเครือข่ายเพิ่มเติมจากเครือข่าย WLAN

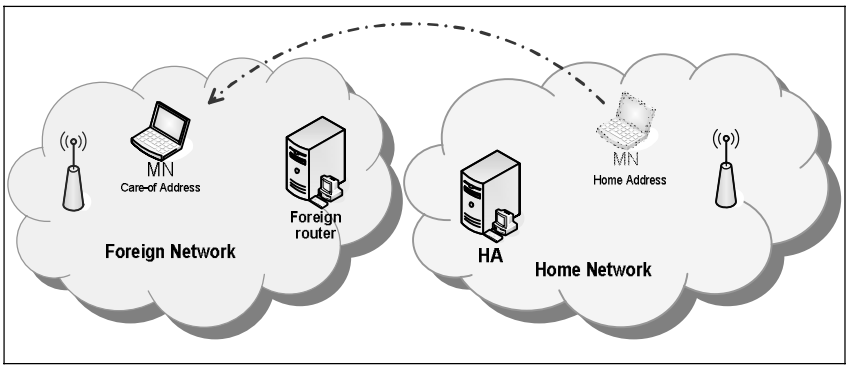


ภาพที่ 3 แสดงส่วนประกอบของระบบ Mobile IPv6

จากภาพที่ 3 เป็นอุปกรณ์และขั้นตอนที่กำหนดขึ้นตามมาตรฐาน RFC 3775 โดยอุปกรณ์ลูกข่ายทุกตัว เรียกว่าโหนด (node) ซึ่งโหนดที่สามารถเคลื่อนที่ได้เรียกว่า Mobile Node (MN) โดยมีเลขหมาย IPv6 2 เลขหมายหลัก คือเลขหมาย IPv6 ที่หนึ่งเป็นเลขหมาย IPv6 แรกที่ได้รับเมื่อเริ่มติดต่อกับเครือค่ายเป็นครั้งแรกเรียกว่าเลขหมาย Home Address (HoA) เป็นเลขหมาย IPv6 จริงของ MN ซึ่งเครือข่ายแรกที่ MN เริ่มการติดต่อนี้เรียกว่า Home Network เลขหมาย IPv6 ที่สองคือเลขหมาย Care-of Address (CoA) เป็นเลขหมายที่ได้รับเมื่อ MN เคลื่อนย้ายไปยังเครือข่ายที่มีกลุ่มของเลขหมาย IPv6 หรือ prefix ต่างออกไป ใช้เพื่อเชื่อมการติดต่อสื่อสารผ่าน Foreign router ในเครือข่ายใหม่ (Foreign Network) ในด้านของผู้ใช้จะรู้จักเพียงเลขหมาย HoA ในการอ้างถึงตัวตนของ MN แต่จะไม่รู้จักเลขหมาย CoA เลย และเพื่อให้ติดตามอุปกรณ์เมื่ออุปกรณ์เคลื่อนย้ายออกจาก Home Network ได้นั้น ต้องมีอุปกรณ์รับรู้ที่อยู่ของ MN ไม่ว่าย้ายไปเครือข่ายไหนเรียกว่า Home Agent (HA) โดย MN ต้องลงทะเบียนกับ HA หลังจากประกาศใช้เลขหมาย HoA เป็นเลข

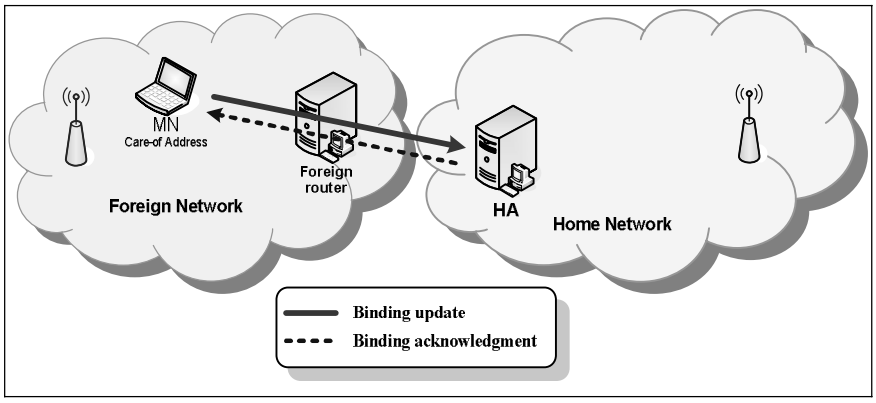
หมาย IPv6 จริง เมื่อ MN เคลื่อนย้ายไปเครือข่ายใดก็ตามต้องส่งข้อความกลับมาบอก HA ว่าย้ายไปที่เครือข่ายไหนและใช้เลขหมาย CoA อะไรอยู่ การส่งข้อความมาบอก HA นี้เรียกว่าขั้นตอน Binding ซึ่งถ้ามีอุปกรณ์ที่ต้องการติดต่อกับ MN (Correspond Node: CN) โดยรู้เลขหมาย IPv6 จริงแล้ว ต้องติดต่อ MN ผ่านทาง HA แล้ว HA จะส่งข้อความนั้นผ่านไปยัง MN ที่อยู่ที่ Foreign Network เสมือนเป็นอุโมงค์ (Tunnel) ที่ติดต่อตรงกันระหว่าง HA กับ MN

1. ขั้นตอนการทำงานเมื่อ Mobile Node เคลื่อนย้ายไปเครือข่ายใหม่



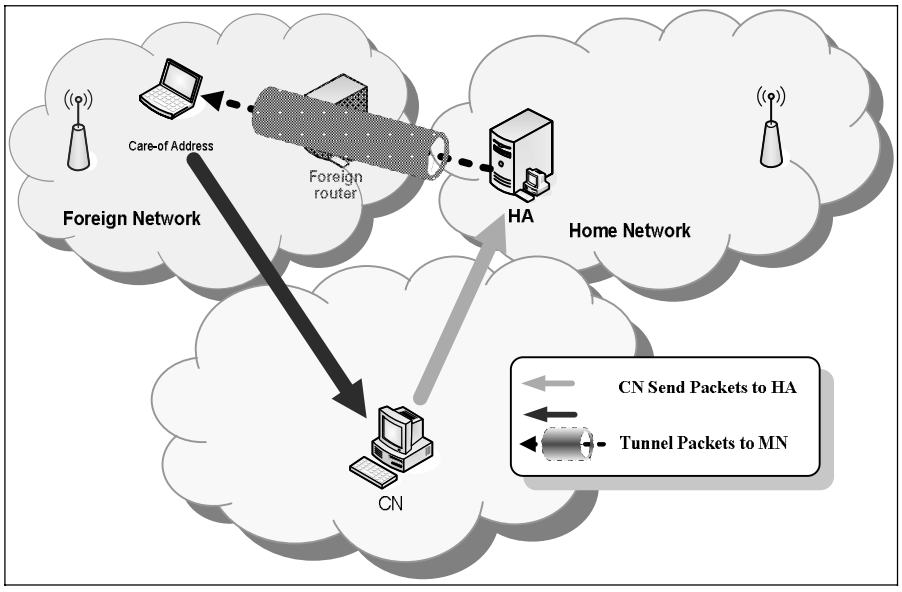
ภาพที่ 4 แสดงการเคลื่อนย้ายของ MN

เมื่อ MN ย้ายไปยัง Foreign Network ดังภาพที่ 4 ขั้นตอนแรก MN ต้องจัดสรรเลขหมาย CoA โดยส่งข้อความที่อยู่ใน Neighbor Discovery for IPv6 (RFC 4861) ในการติดต่อเพื่อตรวจสอบการใช้งานของเลขหมาย CoA ว่ามีการใช้งานอยู่แล้วหรือไม่ ซึ่งเป็นขั้นตอน DAD ที่เป็นส่วนที่จะพัฒนาในวิทยานิพนธ์นี้ เมื่อได้เลขหมาย CoA แล้ว จะสามารถติดต่อกับ Foreign router ได้



ภาพที่ 5 แสดงขั้นตอน HA Registration

เมื่อได้เลขหมาย CoA ที่ได้รับการตรวจสอบขั้นตอน DAD แล้ว MN จะทำขั้นตอนต่อไปที่เรียกว่า HA Registration ไปยัง HA ดังภาพที่ 5 เพื่อลงทะเบียนกับ HA เป็นการบอกที่อยู่ใหม่และเลขหมาย CoA ของ MN มีขั้นตอนคือ MN ส่งข้อความ Binding Update ให้กับ HA โดยผ่านทาง Foreign router และเมื่อ HA ได้รับข้อความ จะตอบกลับด้วยข้อความ Binding Acknowledgement ผ่านทาง Foreign router เพื่อบอกถึงการได้รับข้อความ Binding Update



ภาพที่ 6 แสดงขั้นตอน Triangle Routing

จากภาพที่ 6 เมื่อ CN ต้องการติดต่อกับ MN ที่อยู่ที่ Foreign Network ซึ่ง CN ไม่ทราบการเคลื่อนย้ายของ MN โดย CN ส่งข้อความถึง MN ไปยังเครือข่ายที่เป็นกลุ่มเลขหมาย IPv6 จริงของ MN นั่นก็คือ Home Network โดย HA เป็นผู้ที่ได้รับข้อความที่ส่งมาจาก CN ก่อนเมื่อข้อความนั้นมาถึง Home Network และ HA ได้ทำขั้นตอน HA Registration กับ MN เรียบร้อยแล้ว จึงทราบที่อยู่ใหม่ของ MN จากนั้น HA จะนำข้อความทั้งหมดเป็น payload ของ ข้อความที่ส่งถึงเลขหมาย CoA และส่งไปให้ MN (การส่งข้อความของ HA นี้เรียกว่า Tunnel Packet) เมื่อ MN ได้รับข้อความและทราบว่า CN เป็นผู้ส่ง สามารถตอบข้อความกลับไปให้ CN โดยตรงไม่ต้องผ่าน HA ถ้า CN ต้องการส่งข้อความอีกครั้งต้องติดต่อผ่าน HA ทุกครั้ง การติดต่อแบบสองทางนี้เป็นลักษณะแบบสามเหลี่ยมเรียกว่า Triangle Routing สาเหตุที่ไม่อนุญาติให้ CN กับ MN ติดต่อกันโดยตรงเพื่อความปลอดภัยของ MN ในการระบุตัวตนของคู่ติดต่อสื่อสารที่กำลังสื่อสารกันขณะ MN เคลื่อนที่ไป Foreign Network

การสื่อสารแบบ Triangle Routing นี้ทำให้การส่งข้อความใช้เวลาเพิ่มขึ้น ดังนั้นจึงมีวิธีที่ให้ CN สามารถส่งให้ MN ได้โดยตรงคือวิธี Route Optimization แต่จะมีขั้นตอนเรื่องความปลอดภัยเพื่อระบุตัวตนก่อนส่งข้อความจริงโดยใช้กุญแจในการเข้ารหัสข้อความส่งไป 2 ทางคือ ส่งไป MN โดยตรงและส่งผ่าน HA ไปถึง MN แล้วนำกุญแจของทั้งสองทางมาประกอบเป็นกุญแจเดียวแล้วเข้ารหัสข้อความของ MN ส่งกลับไปที่ CN เรียกขั้นตอนแลกเปลี่ยนกุญแจนี้ว่า Return routability จากนั้นทุกครั้งที่มีการสื่อสารระหว่าง MN และ CN สามารถส่งข้อความได้โดยตรงแต่ต้องใช้กุญแจเข้ารหัสทุกครั้งในการส่ง

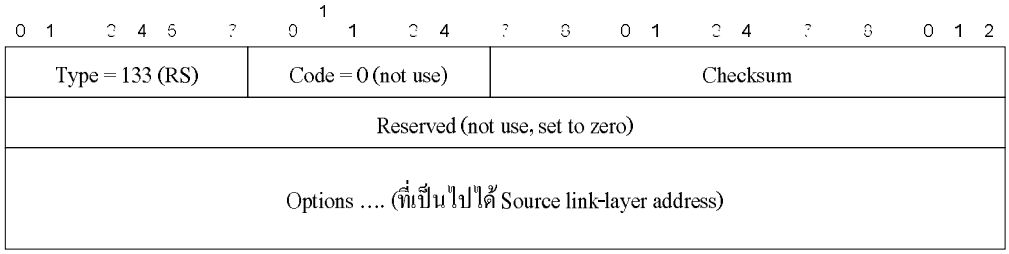
ถ้า MN มีการเคลื่อนย้ายไปที่เครือข่ายอื่นต้องทำขั้นตอนทั้งหมดนี้อีกครั้งได้แก่ ขั้นตอน DAD ตามมาด้วย HA Registration และถ้า CN ต้องการติดต่อต้องสื่อสารแบบ Triangle Routing ถ้าไม่ได้ทำ Route Optimization

ข้อความในกลุ่ม Neighbor Discovery for IPv6

จากหัวข้อที่ผ่านมาแสดงให้เห็นภาพรวมในการทำงานของอุปกรณ์บนเครือข่ายแบบเคลื่อนที่ที่ใช้เลขหมาย IPv6 (Mobile IPv6 Network) และความสำคัญและลำดับการทำงานของขั้นตอน DAD ในหัวข้อนี้จะกล่าวถึงข้อความมาตรฐานที่ใช้ติดต่อเพื่อเตรียมการส่งข้อมูลรวมถึงขั้นตอน DAD คือข้อความในกลุ่ม Neighbor Discovery สำหรับเลขหมาย IPv6 RFC4861

ข้อความในกลุ่ม Neighbors Discovery สำหรับเลขหมาย IPv6 เป็นข้อความ ICMPv6 ชนิดหนึ่งใช้สำหรับกลไก 9 ชนิด ได้แก่ กลไกค้นหา router (Router Discovery) prefix (Prefix Discovery) และพารามิเตอร์ของ link (Parameter Discovery) ในเครือข่าย กลไกเกี่ยวกับเลขหมาย IPv6 คือใช้กำหนดเลขหมาย IPv6 แบบอัตโนมัติ (Address Autoconfiguration) ใช้ในการถาม link-layer address (Address Resolution) เป็นเลขหมายใช้ระบุตัวตนในระดับ link เปรียบเทียบได้กับ MAC Address ของเลขหมาย IPv4 และขั้นตอน Duplicate Address Detection (DAD) กลไกเกี่ยวกับการหาเส้นทางส่งข้อมูลคือ กลไกตัดสินการส่งข้อมูลไปยังปลายทางว่าส่งไป next-hop ใดหรือส่งผ่านทางอุปกรณ์ใดถึงสั้นที่สุด และกลไก Redirect เป็นการหา hop หรือโหนดแรกจาก router ที่ส่งไปยังปลายทางได้ดีที่สุด กลไกสุดท้ายเป็นกลไกค้นหาโหนดว่ายังอยู่ในเครือข่ายหรือไม่ (Neighbor Unreachability Detection) ซึ่งทั้ง 9 กลไกนี้ใช้ข้อความ ICMPv6 5 ประเภทเพื่อสื่อสารตามการทำงานของกลไก

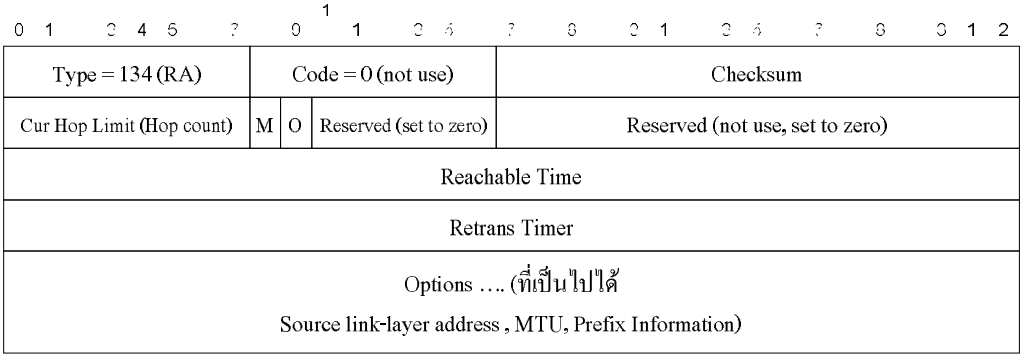
1. ข้อความ Router Solicitation (RS)



ภาพที่ 7 แสดงโครงสร้างของข้อความ RS

เป็นข้อความ ICMPv6 ที่ต้องการส่งให้กับ router มี type code คือ 133 โครงสร้างของข้อความเป็นดังภาพที่ 7 สามารถใส่ option ในกรณีที่ IPv6 Source Address ไม่ได้ระบุไว้จะต้องเพิ่ม Source link-layer address เพื่อระบุตัวตนระดับ link

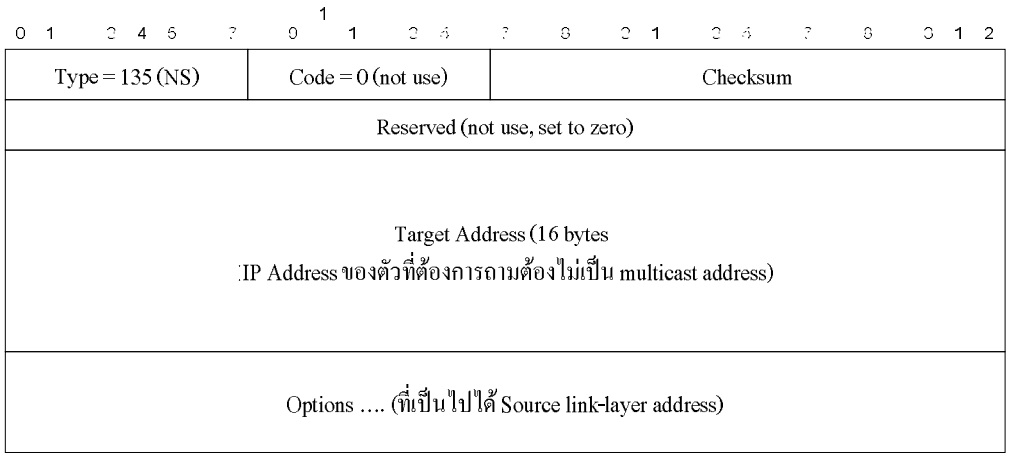
2. ข้อความ Router Advertisement (RA)



ภาพที่ 8 แสดงโครงสร้างของข้อความ RA

เป็นข้อความที่ตอบรับกับข้อความ RS โดยกลุ่มของ IPv6 Source address จะเป็นกลุ่มของ router เท่านั้น เพื่อที่จะบอกค่าที่จำเป็นในการสื่อสารข้อมูล ได้แก่ prefix address สำหรับการทำ Address Autoconfiguration ใช้ใน RFC4862 (S. Thomson, T. Narten and T. Jinmei, 2007) ค่า MTU และ Limit Hop ที่ใช้สำหรับ Internet parameter มี type code คือ 134 โครงสร้างของข้อความเป็นดังภาพที่ 8 มี flag อยู่ 2 ชนิดได้แก่ ‘M’ flag คือ Managed address configuration flag ถ้ามีค่าเป็น ‘1’ เป็นการระบุใช้ stateful address autoconfigrulation และถ้ามีค่าเป็น ‘0’ จะเป็นการใช้ stateless address autoconfigrulation แต่ถ้าต้องการระบุเป็น stateful configuration ชนิดอื่นจะใช้ ‘O’ flag โดยกำหนดให้เป็น ‘1’ ทั้งสอง flag นี้ใช้สำหรับกลไกกำหนดเลขหมาย IPv6 แบบอัตโนมัติ นอกจากนี้ยังมีพารามิเตอร์อื่นอีก ได้แก่ Router Lifetime เป็นตัวแปรชนิด unsigned integer ขนาด 16 บิต มีหน่วยเป็นมิลลิวินาที ถ้าเป็น 0 ทุกบิตบอกได้ว่า router ที่เป็นผู้ส่งนี้ไม่ใช้เป็น default router และไม่ได้อยู่ใน default router list ถัดมาเป็น Reachable Time เป็นตัวแปรชนิด unsigned integer ขนาด 16 บิต มีหน่วยเป็นมิลลิวินาที ใช้สำหรับกลไกค้นหาโหนดว่ายังอยู่ในเครือข่ายหรือไม่ (Neighbor Unreachability Detection) ถ้าเป็น 0 ทุกบิตแสดงว่าไม่ได้ใช้งาน และ Retrans Timer เป็นตัวแปรชนิด unsigned integer ขนาด 32 บิต มีหน่วยเป็นมิลลิวินาที ใช้สำหรับกลไกค้นหาโหนดว่ายังอยู่ในเครือข่ายหรือไม่ (Neighbor Unreachability Detection) และกลไกในการถาม link-layer address (Address Resolution) ถ้าเป็น 0 แสดงว่าไม่ได้ใช้งานเช่นกัน

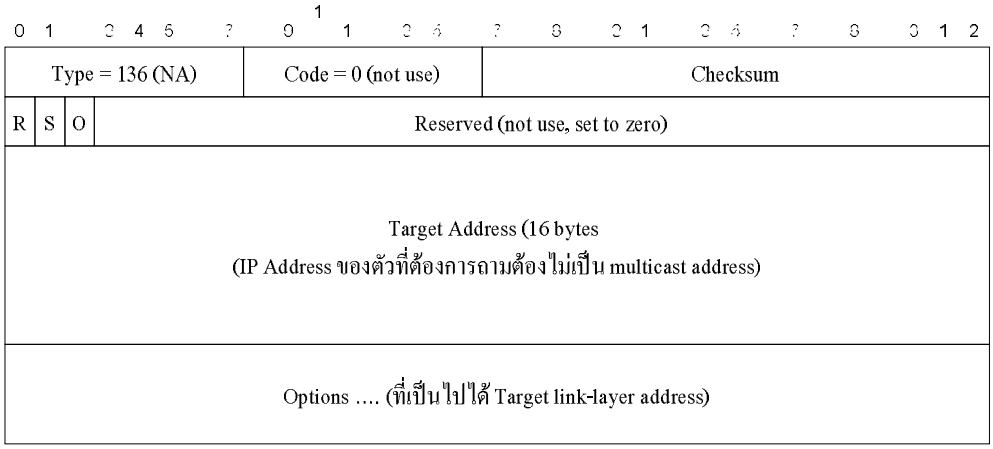
3. ข้อความ Neighbor Solicitation (NS)



ภาพที่ 9 แสดงโครงสร้างของข้อความ NS

เป็นข้อความ ICMPv6 ที่ต้องการส่งให้กับโหนดใดๆ ก็ตามที่อยู่ในเครือข่าย มี type code คือ 135 โครงสร้างของข้อความเป็นดังภาพที่ 9 Target Address ที่อยู่ในข้อความ NS อ้างถึง IPv6 address ที่ต้องการรู้สถานะของโหนดที่ใช้ Target Address นี้ใช้สำหรับกลไกค้นหาโหนดว่ายังอยู่ในเครือข่ายหรือไม่ (Neighbor Unreachability Detection) กลไกในการถาม link-layer address (Address Resolution) และขั้นตอน Duplicate Address Detection (DAD) ซึ่งตามข้อกำหนดต้องไม่เป็น Multicast address และสามารถใส่ option ในกรณีที่ IPv6 Source Address ไม่ได้ระบุไว้จะต้องเพิ่ม Source link-layer address เพื่อระบุตัวตนระดับ link

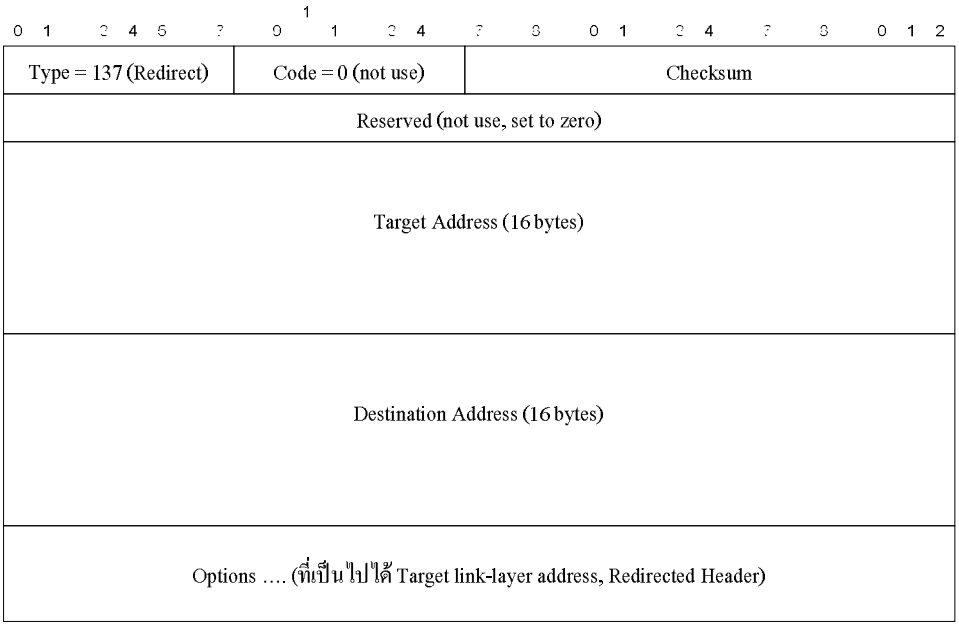
4. ข้อความ Neighbor Advertisement (NA)



ภาพที่ 10 แสดงโครงสร้างของข้อความ NA

เป็นข้อความที่ตอบรับกับข้อความ NS ซึ่งตอบในกลุ่มเครือข่ายเดียวกันเท่านั้น เพื่อที่จะให้ข้อมูลสถานะที่จำเป็นในการสื่อสารข้อมูล มี type code คือ 136 โครงสร้างของข้อความเป็นดังภาพที่ 10 โดยใช้ Target address เป็นเลขหมาย IPv6 เดียวกันกับข้อความ NS ที่ต้องการตอบ ซึ่งตามข้อกำหนดต้องไม่เป็น Multicast address หรือถ้าไม่ระบุต้องใช้เป็น Target link-layer address ใน option เพื่อระบุตัวตนระดับ link แทน มี flag อยู่ 3 ชนิดได้แก่ ‘R’ flag หรือ Router flag มีค่าเป็น ‘1’ หมายถึงผู้ส่งเป็น router ใช้สำหรับกลไกค้นหาโหนดว่ายังอยู่ในเครือข่ายหรือไม่ (Neighbor Unreachability Detection) ‘S’ flag หรือ Solicited flag มีค่าเป็น ‘1’ หมายถึงการตอบคำขอของข้อความ RS ใช้สำหรับยืนยันการมีอยู่ของกลไกค้นหาโหนดว่ายังอยู่ในเครือข่ายหรือไม่ (Neighbor Unreachability Detection) และ ‘O’ flag หรือ Override flag มีค่าเป็น ‘1’ หมายถึงข้อความนี้ override

5. ข้อความ Redirect



ภาพที่ 11 แสดงโครงสร้างของข้อความ Redirect

เป็นข้อความที่ใช้ส่งข้อมูลบอกเส้นทางการส่งข้อมูลไปยังปลายทางที่ผ่าน router อื่นที่ไม่ใช่ default router ซึ่งสั้นกว่าผ่าน default router ใช้สำหรับกลไก Redirect มี type code คือ 137 โครงสร้างของข้อความเป็นดังภาพที่ 11 Target Address เป็นเลขหมาย IPv6 ของ router ที่เป็นเส้นทางที่สั้นกว่า หรือถ้าไม่ระบุใช้เป็น Target link-layer address ของ router มาแทนใน option และ Destination Address เป็นเลขหมาย IPv6 ของปลายทางที่ต้องการส่ง และในกรณีที่ข้อมูลที่บรรจุอยู่ในข้อความนี้เกิน 1280 ไบต์ จะต้องแบ่งข้อความโดยแต่ละข้อความบรรจุได้ไม่เกิน 1280 ไบต์และใส่ Redirected Header option เพื่อระบุตำแหน่งการต่อข้อมูล

ขั้นตอน Duplicate Address Detection

จากข้อความ ICMPv6 ทั้ง 5 ประเภทที่ได้กล่าวไปนั้น การเรียกใช้งานข้อความของแต่ละกลไกจะต้องแตกต่างกันและใช้พารามิเตอร์ไม่เหมือนกัน ทำให้อุปกรณ์ภายในเครือข่ายสามารถแยกประเภทของกลไกและตอบสนองได้ถูกต้อง ในบทความนี้จะข้อกล่าวถึงขั้นตอนของกลไก Duplicate Address Detection (DAD) เพียงกลไกเดียว เพื่อประโยชน์ในการพัฒนาขั้นตอน ซึ่งมีขั้นตอนกำหนดในมาตรฐาน RFC 4862: IPv6 Stateless Address Autoconfiguration

IPv6 Stateless Address Autoconfiguration เป็นการกำหนดเลขหมาย IPv6 อัตโนมัติที่ไม่มีการเก็บสถานะของอุปกรณ์ว่าได้ใช้เลขหมายนี้ไปแล้วโดยถ้าต้องการใช้เลขหมาย IPv6 ต้องประกาศถามเพื่อไม่ให้เลขหมายที่ใช้ไม่ซ้ำกันทุกครั้ง แต่ถ้าเป็นการทำงานที่เก็บสถานะหรือ Stateless Address Autoconfiguration ต้องมีอุปกรณ์อย่างน้อยหนึ่งอุปกรณ์จัดการเก็บสถานะของเลขหมายที่ได้มีการใช้งานอยู่ ตัวอย่างของวิธีที่มีการเก็บสถานะคือ DHCP ที่ต้องมี DHCP Server จัดการจ่ายเลขหมายที่ยังไม่ได้ใช้งาน DHCP ได้รับความนิยมในระบบเลขหมาย IPv4 ที่จำนวนลูกข่ายในเครือข่ายสามารถมีได้ไม่มากเท่าไร แต่เมื่อใช้เครือข่ายเลขหมาย IPv6 จำนวนลูกข่ายต่อหนึ่งเครือข่ายจะเพิ่มขึ้นมาก server ต้องจัดการเลขหมายมากขึ้น เนื่องจาก DHCP เป็นการทำงานที่มีศูนย์กลาง (Centralized) ดังนั้นถ้า server เสียหรือไม่สามารถทำงานได้ การส่งข้อมูลที่ต้องการเลขหมาย IP ต้องหยุดชะงักไป การสื่อสารไม่เป็นไปอย่างต่อเนื่อง เห็นได้ว่าการทำงานที่มีศูนย์กลางในระบบใหญ่ๆ นั้นเริ่มไม่เหมาะสม IPv6 Stateless Address Autoconfiguration จึงเป็นแนวคิดแก้ไขปัญหาอีกทางหนึ่งที่ไม่ต้องมีศูนย์กลางของระบบ แต่ต้องเพิ่มเวลาสำหรับขั้นตอนการตรวจสอบเลขหมายแทนนั้นคือขั้นตอน Duplicate Address Detection (DAD) ที่เป็นขั้นตอนหนึ่งใน IPv6 Stateless Address Autoconfiguration

Duplicate Address Detection (DAD) เริ่มขั้นตอนหลังจาก MN กำหนดเลขหมายมาไม่ว่าจะอยู่ที่ Home network หรือ Foreign network เป็นเลขหมาย HoA หรือ CoA ก็ตาม เมื่อกำหนดขึ้นมาต้องตรวจสอบว่ามีการใช้งานแล้วหรือไม่ ตามขั้นตอนจะใช้เวลา 1000 มิลลิวินาที ในการรอการตอบกลับ มีขั้นตอนตามมาตรฐาน RFC 4862 ดังนี้

1) ในการกำหนดเลขหมาย IPv6 ใหม่นั้น MN ต้องการ 2 ส่วนมาประกอบกัน ส่วนที่หนึ่งคือ prefix ความยาว 64 บิต ซึ่งได้มาจากการรอข้อความ RA ที่มี prefix information option ที่ส่งมา

เป็นทุกช่วงเวลา หรือส่งมาเมื่อ MN ส่งข้อความ RS ที่ source address = ไม่ระบุ, destination address = FF02::2 (all router multicast group) ซึ่งเป็นลักษณะของการถามหาพารามิเตอร์ของเครือข่ายจากนั้น default router จะส่งข้อความ RA ที่มี prefix information option กลับมา และส่วนที่สอง Interface ID ความยาว 64 บิต เป็นส่วนที่ MN คิดคำนวณขึ้นมาเองจากการสุ่มหรือใช้ MAC address (เรียกว่าวิธี EUI64) หรือวิธีอื่นๆ เมื่อนำทั้งสองส่วนมาประกอบกันจะได้เลขหมาย IPv6 ที่มีความยาว 128 บิตเรียกว่า Tentative address ซึ่งยังไม่สามารถใช้ส่งหรือรับข้อมูลได้จนกว่าจะได้ทำการตรวจสอบว่าเลขหมาย IPv6 นี้ไม่ซ้ำกับอุปกรณ์อื่นในเครือข่าย

2) หลังจากที่ MN ได้ Tentative address จะเริ่มขั้นตอนการตรวจสอบ DAD โดย MN จะส่งข้อความ ICMPv6 ชนิด NS ที่ Source address = ไม่ระบุ, Destination address = FF02::1 (all node muticast group) Target Address = Tentative address เป็นลักษณะของกลไก DAD ถามอุปกรณ์ต่างๆ ว่ามีใครใช้เลขหมาย IPv6 เดียวกันนี้หรือไม่

3) MN รอการตอบกลับของข้อความ NA เป็นเวลา 1000 มิลลิวินาที โดยถ้ามีการตอบกลับแสดงว่ามีอุปกรณ์ที่ใช้เลขหมาย IPv6 ที่ตรงกับ Tentative address อยู่แล้ว หากพ้นระยะเวลานี้แล้ว MN ไม่ได้รับข้อความ NA จากอุปกรณ์ใดเลย MN จะเข้าใจว่า Tentative address นี้ยังไม่มีใครใช้หรือ uniqueness และสามารถใช้เป็นเลขหมาย HoA หรือ CoA ในเครือข่ายได้เลย เป็นการจบขั้นตอน DAD อย่างไรก็ตามหาก MN ได้รับข้อความ NA ต้องกำหนด Tentative address ขึ้นมาใหม่และย้อนกลับไปทำขั้นตอนที่ 2) ใหม่อีกครั้ง

จากขั้นตอนทั้งหมดจะเห็นได้ว่าการตรวจสอบ DAD นั้นจะต้องรอเวลาอย่างน้อยที่สุดคือ 1000 มิลลิวินาที และในเวลา 1000 มิลลิวินาที เป็นช่วงเวลาที่ไม่สามารถติดต่อสื่อสารได้เนื่องจากเป็น Tentative address ถ้า MN มีการส่งข้อมูลอยู่ แล้วเคลื่อนที่มาในเครือข่ายใหม่ต้องหยุดการติดต่ออย่างน้อยที่สุดคือ 1000 มิลลิวินาที ซึ่งถ้าเป็นงานที่ต้องการความต่อเนื่องจะมีผลกระทบที่สามารถสังเกตได้ทันที แสดงให้เห็นว่าการรอการตรวจสอบ DAD นั้นนานเกินไปสำหรับโปรแกรมบางประเภท การแก้ปัญหาในเรื่องนี้มีแนวคิดอยู่ด้วยกันหลายวิธีไม่ว่าจะเป็นการเพิ่มตัวกลางในการจัดการเลขหมาย IPv6 แบบต่างๆ การพยายามปรับให้ช่วงที่เป็น Tentative address สามารถส่งหรือรับข้อความที่ใช้สำหรับติดต่อสื่อสารข้อมูลได้ หรือการปรับโครงสร้างเครือข่ายและเพิ่มโพรโตคอลใหม่ให้เอื้ออำนวยกับการเตรียม CoA แสดงให้เห็นในหัวข้อถัดจากนี้

บทความที่เกี่ยวข้องกับงานวิจัย

จากขั้นตอนการทำงานและปัญหาด้านเวลาที่เกิดขึ้นกับ Duplicate Address Detection (DAD) แบบมาตรฐานในการทำงานที่ต้องการความต่อเนื่องของข้อมูล ซึ่งในขั้นตอน Duplicate Address Detection (DAD) ต้องใช้เวลาอย่างน้อย 1000 มิลลิวินาที คิดเป็นกว่า 10% ของเวลาในการย้ายข้ามเครือข่ายทั้งหมด (Adisak et al., 2006) ทำให้มีนักวิจัยต่างคิดค้นวิธี Duplicate Address Detection (DAD) วิธีใหม่ที่ใช้เวลาของการได้รับเลขหมาย IPv6 แบบไม่ซ้ำกันลดลง โดยแนวคิดของแต่ละวิธีมีจุดเด่นและจุดด้อยที่ต่างกันออกไป ในหัวข้อนี้ขอนำเสนอด้วยกัน 4 วิธี ได้แก่ Optimistic DAD (O-DAD) (N. Moore, 2006), Advance DAD (A-DAD) (Y. Han et al., 2003; Y.-H. Han and S.-H. Hwang, 2006), Proactive DAD (P-DAD) (Chien-Chao Tseng et al., 2006) และ DHCPv6 (R. Droms. Ed., 2003)

1. Optimistic DAD

วิธีนี้ใช้หลักการที่คาดว่าในระบบเครือข่ายที่ใช้เลขหมาย IPv6 มีความน่าจะเป็นที่การซ้ำกันของเลขหมาย IPv6 มีค่าน้อยมากตามหลักความน่าจะเป็นของ Birthday Paradox จากแนวคิดนี้สรุปได้ว่ามีความเป็นไปได้ที่ Tentative address ยังไม่มีใครนำไปใช้ จึงกำหนดให้ Tentative address หรือในวิธีนี้เรียกว่า Optimistic address สามารถส่งหรือรับข้อความที่ใช้สำหรับติดต่อสื่อสารข้อมูลได้บางชนิด โดยใช้แนวคิดของการปรับมาตรฐาน RFC4861: Neighbor Discovery for IPv6 และ RFC 4862: IPv6 Stateless Address Autoconfiguration บางข้อกำหนดและเพิ่ม Optimistic flag เข้าไปเพื่อใช้สำหรับ Optimistic address แนวคิดในการปรับมาตรฐานคือไม่ต้องการให้ Optimistic address ส่ง Link Layer address (MAC address) ของตนไปกับข้อความทุกข้อความที่ส่งออกเพื่อป้องกันการจับคู่เลขหมาย IPv6 กับ Link Layer address ที่ไม่ถูกต้องใน Neighbor cache entries ของอุปกรณ์อื่นๆ ในกรณีที่ Optimistic address นี้มีอุปกรณ์ใช้ไปแล้ว โดยมีหลักการคือ

1) ใช้ Override flag = 0 ในข้อความ Neighbor Advertisement เป็นการบอกให้เขียน Optimistic address ทับใน Neighbor cache entries ถ้ามีการจับคู่ที่ไม่เหมือนกัน

2) ไม่ส่ง ข้อความ Neighbor Solicitation จาก Optimistic address เพราะปกติจะมีการใส่ Source Link Layer Address Option อาจมีการใส่ใน Neighbor cache entries แต่ถ้าเป็น Neighbor Solicitation ในขั้นตอน DAD ต้องไม่ระบุเลขหมาย IPv6 ของผู้ส่ง

3) ไม่ส่งข้อความ Router Solicitation ที่มี Source Link Layer Address Option ถ้าต้องการส่งต้องไม่มี Source Link Layer Address Option

เพื่อให้ตรงตามหลักการแนวคิดที่ได้กำหนด จึงมีการเปลี่ยนแปลงข้อกำหนด RFC4861: Neighbor Discovery for IPv6 บางอย่าง ได้แก่ ในการใช้ข้อความ Router Solicitation ถาม Prefix Information ควรส่งจาก MN ที่ไม่ได้ใช้ Optimistic address หรือ จาก Optimistic address ซึ่งต้องไม่ใส่ Source Link Layer Address Option สำหรับกลไกใช้ในการถาม link-layer address (Address Resolution) ต้องไม่ใช้ Optimistic address เป็น Source address ของข้อความ Neighbor Solicitation และถ้าการส่งข้อความ Router Solicitation ที่ไม่มี Source Link Layer Address Option ทำให้การตอบข้อความ Router Advertisement ไม่สามารถทำได้ให้รอจนการตรวจสอบ DAD เสร็จก่อนจึงสามารถทำการติดต่อ router ได้ และกรณีที่ต้องการส่งข้อความไปให้ neighbor ขณะที่ใช้ Optimistic address ห้ามถาม Access Router ให้ใช้ข้อความ Redirect ที่มี Target Link Layer Address Option ไปที่ default router

เช่นเดียวกันมีการเปลี่ยนแปลงข้อกำหนด RFC 4862: IPv6 Stateless Address Autoconfiguration คือ ในการสร้าง Global และ Site-Local address ไม่ควรนำ Optimistic address มาเป็นเลขหมาย IPv6 ใหม่ถ้ายังไม่รู้ Link Layer Address ของ router ขณะที่เป็น Optimistic address ต้องใช้ Optimistic flag ในการใช้งานก่อน และถ้า DAD Process เสร็จแล้วให้เปลี่ยนสถานะเป็น Preferred หรือ Deprecated address ตามแบบมาตรฐานปกติ และ router ต้องไม่ตอบข้อความ Router Solicitation จากข้อความที่ไม่ระบุ Source address แต่จะตอบกับข้อความที่มากจาก Optimistic address ที่ใช้ Unicast address โดยการใช้ Override flag ร่วมด้วย (O=0)

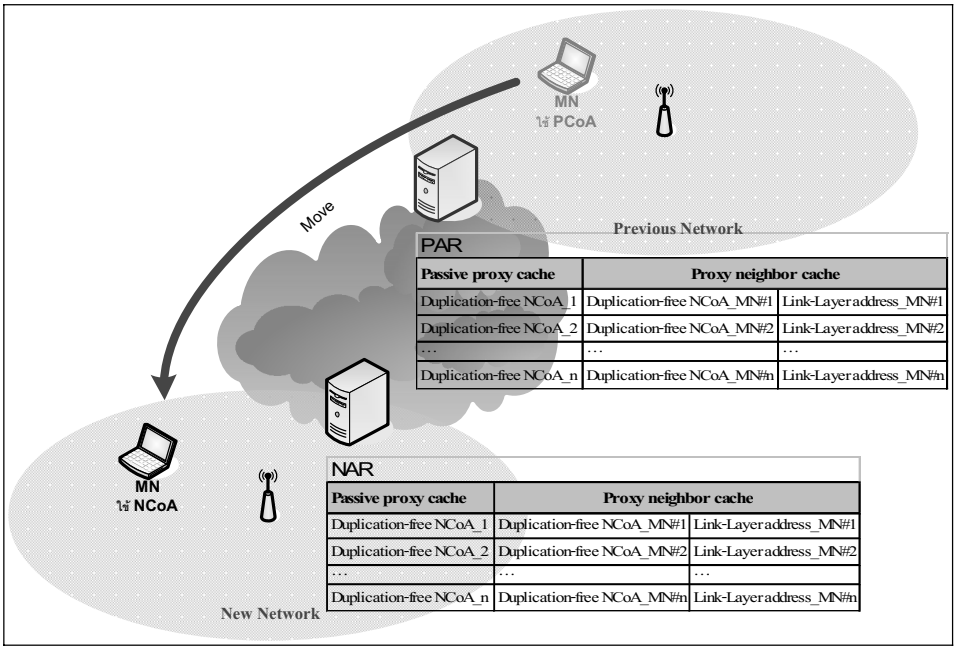
จากการปรับเปลี่ยนข้อกำหนดจะทำให้ได้ขั้นตอนการตรวจสอบที่เปลี่ยนไปคือ เมื่อ MN เข้ามาในเครือข่ายใหม่ให้รอรอบของข้อความ Router Advertisement ที่มีการบอก Prefix Information แล้วทำการกำหนด Optimistic address ที่มีการรวม Prefix Information จากนั้นให้ MN ส่งข้อความ Neighbor Solicitation ที่ไม่มี Source Link Layer Address Option และไม่ระบุปลายทางทันที เป็นการทำงานขั้นตอน DAD พร้อมทั้งใช้งานอื่นๆได้เลยโดยมีข้อกำหนดการส่งข้อความ

ตามข้างต้น และเมื่อรอถึงเวลา 1000 มิลลิวินาที แล้วยังไม่มีใครตอบกลับ Optimistic address จะเปลี่ยนสถานะไปเป็น Preferred address ตาม RFC 4862

จากแนวคิดนี้ ข้อดีที่เห็นได้คือการปรับเปลี่ยนมาตรฐาน ซึ่งสามารถทำงานกับ DAD แบบมาตรฐานได้ ผลที่ได้ดีขึ้นโดยไม่มีการเพิ่มอุปกรณ์ ยังคงเป็นการทำงานที่ไม่ต้องมีศูนย์กลาง อย่างไรก็ตาม การส่งและรับข้อความยังไม่สามารถทำงานได้อย่างเต็มที่ขณะที่เป็น Optimistic address

2. Advance DAD

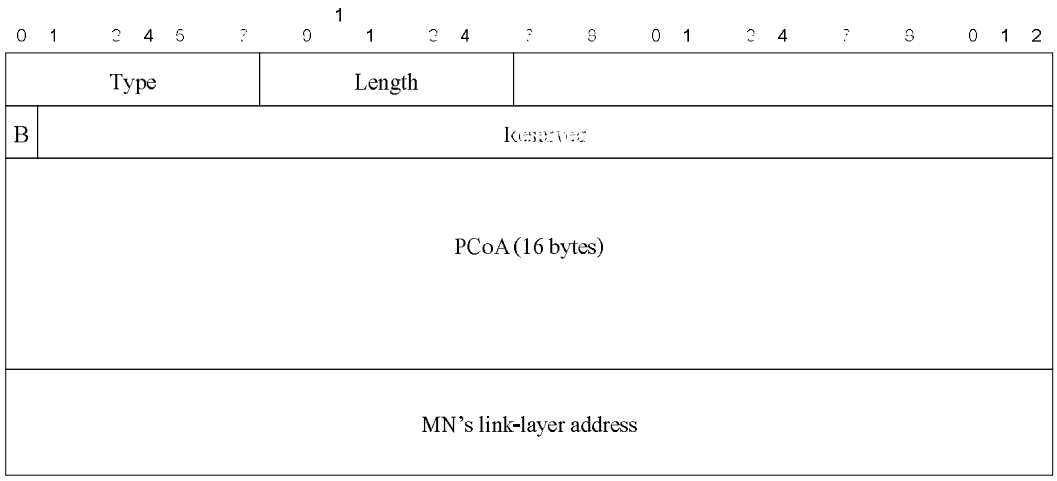
Advance DAD หรือ A-DAD ใช้แนวคิดของการเก็บเลขหมาย IPv6 ที่ผ่านขั้นตอน DAD ไว้แล้วจำนวนหนึ่ง (กำหนดไว้ที่ 100 เลขหมาย) ไว้ที่ Access router ของแต่ละเครือข่าย และใช้การทำนายการเคลื่อนที่ของ MN ด้วยวิธี Fast Handover (R. Koodli, 2005) เพื่อเตรียมเลขหมาย IPv6 ที่ไม่ซ้ำกันไว้ล่วงหน้า



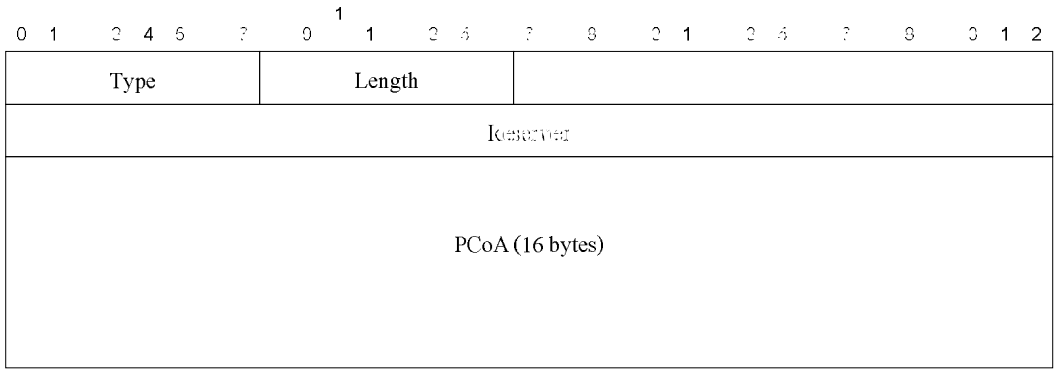
ภาพที่ 12 แสดงโครงสร้างเครือข่ายของวิธี A-DAD

ในด้านของการออกแบบนั้น มีโครงสร้างเครือข่ายตามภาพที่ 12 กำหนดให้เลขหมาย CoA ของ MN ในเครือข่ายใหม่เรียกว่าเลขหมาย NCoA (New Care-of Address) และในเครือข่ายก่อนหน้านี้เป็นเลขหมาย PCoA (Previous Care-of Address) โดยเลขหมาย NCoA ที่ผ่านขั้นตอน DAD แล้วจะเรียกว่า Duplication-free NCoA ซึ่งเก็บอยู่ใน Passive proxy cache และกำหนดให้ Access router ในเครือข่ายใหม่เรียกว่า NAR (New Access Router) และเครือข่ายก่อนหน้าเรียกว่า PAR (Previous Access Router)

A-DAD ใช้ส่วนเพิ่มเติมและโพรโตคอลใหม่ที่เพิ่มขึ้น เพื่อให้ได้ตามแนวคิดโดยเพิ่ม Passive proxy cache เพื่อเก็บ Duplication-free NCoA ที่อยู่ใน Access router และเพิ่ม Proxy neighbor cache เป็น cache ที่เก็บ Duplication-free NCoA กับ link layer address ของ MN ที่จับคู่กับหลังจากมีการขอ IPv6 address จาก Passive proxy cache แล้ว มีข้อกำหนดคือ (1) Passive proxy cache ต้องไม่ตอบข้อความ Neighbor Solicitation ที่มี Target address เป็นเลขหมาย IPv6 ที่อยู่ใน Passive proxy cache และลบเลขหมาย IPv6 นั้นออกจาก Passive proxy cache เลยเพราะแสดงว่าเลขหมาย IPv6 นั้นมีการใช้งานอยู่หรือกำลังถูกตรวจสอบ (2) Passive proxy cache ต้องไม่มีผลกระทบกับ Neighbor cache ของ Neighbor node เพราะ Duplication-free NCoA ที่ได้ยังไม่มีการใช้งานเลขหมาย IPv6 (3)ไม่ใช้ Duplication-free NCoA ใน Passive proxy cache ส่ง multicast ในข้อความ unsolicited Neighbor Advertisement และ (4) การหา IPv6 address ของ Passive proxy cache จะใช้วิธีสุ่มเอาแล้วใช้การตรวจสอบ DAD แบบมาตรฐาน



ภาพที่ 13 โครงสร้างของ Duplication-free NCoA Request Option



ภาพที่ 14 โครงสร้างของ Duplication-free NCoA Reply Option

นอกจากนี้ยังมีการเพิ่มโพรโตคอลใหม่ 2 ข้อความ เป็น option ของข้อความ ICMPv6 ในกลุ่ม Neighbor Discovery for IPv6 เพื่อใช้ในการขอเลขหมาย IPv6 จาก Passive proxy cache คือ Duplication-free NCoA Request Option ดังภาพที่ 13 เป็นข้อความที่ใช้ร้องขอ NCoA ใหม่ที่ได้ทดสอบว่าไม่มีการซ้ำกันของเลขหมาย ซึ่ง ‘B’ flag คือ Buffer packets flag จะใช้เมื่อ MN ไม่ได้อยู่ในเครือข่ายของ NAR ซึ่งหมายถึงว่าสามารถทำนายได้ว่า MN จะย้ายมา Network ที่ NAR และ Duplication-free NCoA Reply Option ดังภาพที่ 14 เป็นข้อความที่ใช้ตอบการขอ NCoA ใหม่ที่ได้ทดสอบว่าไม่มีการซ้ำกันของ Address แล้วมี Identifer เป็นรหัสที่ตรงกันของคู่ข้อความที่ร้องขอเพื่อระบุว่าเป็นข้อความถาม-ตอบที่เป็นคู่กันที่ถูกต้อง

ขั้นตอนการทำงานของ A-DAD จะแบ่งได้เป็น 2 กรณีคือ กรณีที่ทำนายการเคลื่อนที่ของ MN ได้ (Predictive case) มีขั้นตอนดังนี้

(1) เมื่อทราบว่า MN จะเคลื่อนที่เครือข่ายไหนแล้วก็ทำการส่ง Duplication-free NCoA Request Option message ที่ใช้ ‘B’ flag ผ่านทาง PAR ไปยัง NAR

(2) เมื่อ NAR ได้รับข้อความ จะเลือก Duplication-free NCoA จาก Passive proxy cache 1 ตัวไปใส่ใน Duplication-free NCoA Reply Option message และลบ address นั้นออกจาก Passive proxy cache แล้วไปใส่ใน Proxy neighbor cache ที่ใช้ IP address นั้นกับ Link layer address ของ MN แล้วทำการส่ง Duplication-free NCoA Reply Option message กลับไปให้ MN ผ่านทาง PAR

(3) เมื่อ MN ได้รับ จะเริ่มเคลื่อนย้ายไปที่เครือข่ายใหม่จะติดต่อกับ NAR แล้ว NAR จะย้ายข้อมูลของ Proxy neighbor cache ไปยัง Neighbor cache พร้อมกำหนดสถานะเป็น REACHABLE

(4) เริ่มทำ Binding update ได้

และกรณีที่ไม่สามารถทำนายการเคลื่อนที่ของ MN ได้ (Non-predictive case) มีขั้นตอนดังนี้

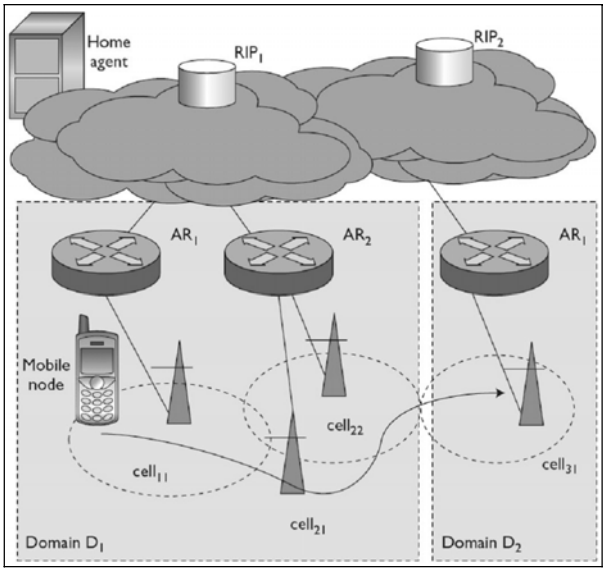
(1) เมื่อ MN มาถึงเครือข่ายใหม่จะทำการติดต่อกับ NAR เพื่อขอ Duplication-free NCoA ทันทีโดยการส่งข้อความ Duplication-free NCoA Request Option ไปที่ NAR

(2) เมื่อ NAR ได้รับข้อความจะเลือก Duplication-free NCoA จาก Passive proxy cache 1 ตัวไปใส่กับข้อความ Duplication-free NCoA Reply Option และลบ Duplication-free NCoA นั้นออกจาก Passive proxy cache แล้วไปใส่ใน Neighbor cache ที่ใช้ Duplication-free NCoA นั้นกับ Link layer address ของ MN พร้อมกำหนดสถานะเป็น REACHABLE แล้วทำการส่งข้อความ Duplication-free NCoA Reply Option

(3) เมื่อ MN ได้รับ NCoA เริ่มทำ Binding Update ได้

จากกระบวนการทำงานของ A-DAD เห็นได้ว่าไม่ต้องรอการตรวจสอบ DAD 1000 มิลลิวินาที เพราะมีการตรวจสอบ DAD และเก็บเลขหมาย IPv6 ล่วงหน้าจำนวนหนึ่งแล้ว ทำให้สามารถทำขั้นตอน Binding update ได้เร็วขึ้น โอกาสข้อมูลสูญหายน้อยลง อย่างไรก็ตามอุปกรณ์ MN ต้องรู้จักโพรโตคอลใหม่ถึงทำงานได้ และเมื่อสังเกตการทำงานของ Access router นอกจากจะทำหน้าที่สร้างทางเชื่อมต่อแล้ว ยังต้องทำการตรวจสอบ DAD และรับ-ส่งข้อความของ A-DAD ซึ่งเป็นการเพิ่มภาระหน้าที่ของ Access router ถ้า Access router ไม่สามารถทำงานได้ A-DAD ก็ไม่สามารถใช้ได้เช่นกัน

3. Proactive DAD



ภาพที่ 15 แสดงโครงสร้างเครือข่ายที่ใช้ใน P-DAD

AP's BSSID	AR's IP address	Advertised prefix
(AP_{11}) 00-01-4A-C3-07-01	(AR_1) 2001:0E10:6440:0001::1	$Prefix_1$
(AP_{21}) 00-01-4A-C3-07-02	(AR_2) 2001:0E10:6440:0002::1	$Prefix_2$
(AP_{22}) 00-01-4A-C3-07-03	(AR_2) 2001:0E10:6440:0002::1	$Prefix_2$
(AP_{31}) 00-01-4A-C3-07-04	(AR_3) 2001:0E10:6440:0003::1	$Prefix_3$

ภาพที่ 16 แสดง Mobile-node Attachment point (MAP) table

Proactive DAD หรือ P-DAD มีแนวคิดที่เตรียมทำการตรวจสอบก่อนถึงเครือข่ายใหม่โดยไม่มีการเก็บเลขหมาย IPv6 จำนวนหนึ่งเหมือนกับ A-DAD ซึ่งต้องใช้การทำนายการเคลื่อนที่ของ MN ด้วยวิธี Location-Based Fast Handoff for 802.11 Networks (C.C. Tseng, 2005) เพื่อเป็นข้อมูลในการส่งข้อความไปตรวจสอบ DAD ล่วงหน้าที่เครือข่ายที่จะเคลื่อนย้ายไป มีการเพิ่มอุปกรณ์ที่เรียกว่า Regional Information Point (RIP) server อยู่เหนือ Access router และ Access point ใน domain เดียวกันดังภาพที่ 15 RIP server นี้จะเก็บ Mobile-node Attachment point (MAP) table เป็นตารางที่มีการเก็บค่า 3 ค่าใน 1 entry ตามภาพที่ 16 คือ access point's basic service set ID (BSSID), access router IP address ที่ใช้ BSSID นั้นและ Prefix information ของ access router นั้น และเพิ่มข้อความใหม่สำหรับติดต่อกัน 4 ข้อความคือ ข้อความที่ 1 คือข้อความ CoA-preAllocate Request ใช้สำหรับตรวจสอบ Care-of Address ของเครือข่ายใหม่ขณะที่ยังไม่ย้ายมาโดยมีพารามิเตอร์ addr

เป็น Care-of Address ของเครือข่ายใหม่ที่ต้องการตรวจสอบ ข้อความที่ 2 คือ ข้อความ CoA-preAllocate Reply ใช้ตอบข้อความ CoA-preAllocate Request โดยมี U bit บอกว่า Care-of Address ที่ตรวจสอบยังไม่มีการใช้งาน ข้อความที่ 3 คือข้อความ CoA_activation Request ใช้เมื่อย้ายไปเครือข่ายใหม่และต้องการใช้ Care-of Address ที่ได้ตรวจสอบแล้ว ต้องบอกกับ Access router ของเครือข่ายใหม่และข้อความที่ 4 คือข้อความ CoA_activation Replyใช้ตอบข้อความ CoA_activation Request message

จากโครงสร้างทั้งหมดนำมาเป็นขั้นตอนของ P-DAD ได้ดังนี้

(1) ก่อนที่จะทำการย้ายไปยัง Access point ใหม่ MN เอา prefix ของ Access router ใหม่จาก MAP table ของ RIP Server และคิดคำนวณ Tentative address แล้วจึงทำการส่งข้อความ CoA-preAllocate Request กับพารามิเตอร์ addr สำหรับขั้นตอน DAD

(2) เมื่อ Access router ใหม่ได้รับข้อความ CoA-preAllocate Request จะตรวจสอบข้อมูลใน Registration cache ซึ่งเป็น cache ที่เก็บข้อมูล 3 อย่างคือ Home address, Pre-allocated new Care-of Address และ life time ของเลขหมาย CoA และถ้าจำเป็นอาจจะใช้การตรวจสอบ DAD แบบมาตรฐานตรวจสอบเพิ่มเติม หลังจากที่ตรวจสอบแล้วว่าเลขหมาย CoA ที่มาจากพารามิเตอร์ addr นี้ไม่มีใครใช้มาก่อนจะส่งข้อความ CoA_preAllocate Reply ที่มี U bit

(3) เมื่อ MN ได้รับข้อความ CoA_preAllocate Reply จะเก็บเลขหมาย CoA นั้นไว้ที่ Registration cache ของตนและส่งข้อความ CoA_activation Request ถึง Access router ใหม่และ Access router ใหม่จะเก็บ Care-of Address นี้ลงไปใน Registration cache แต่ถ้าไม่ได้รับ U bit จะต้องทำขั้นตอน DAD แบบมาตรฐาน และข้ามไปข้อ 5

(4) ในกรณีที่ได้รับ U bit เมื่อไปถึง Access router ใหม่จะรู้การมาถึงของ MN และส่ง CoA_activation Reply message และลบเลขหมาย CoA นี้ออกจาก Registration cache แต่ถ้า MN ไม่ได้รับข้อความCoA_activation Reply ให้ทำขั้นตอน DAD แบบมาตรฐาน

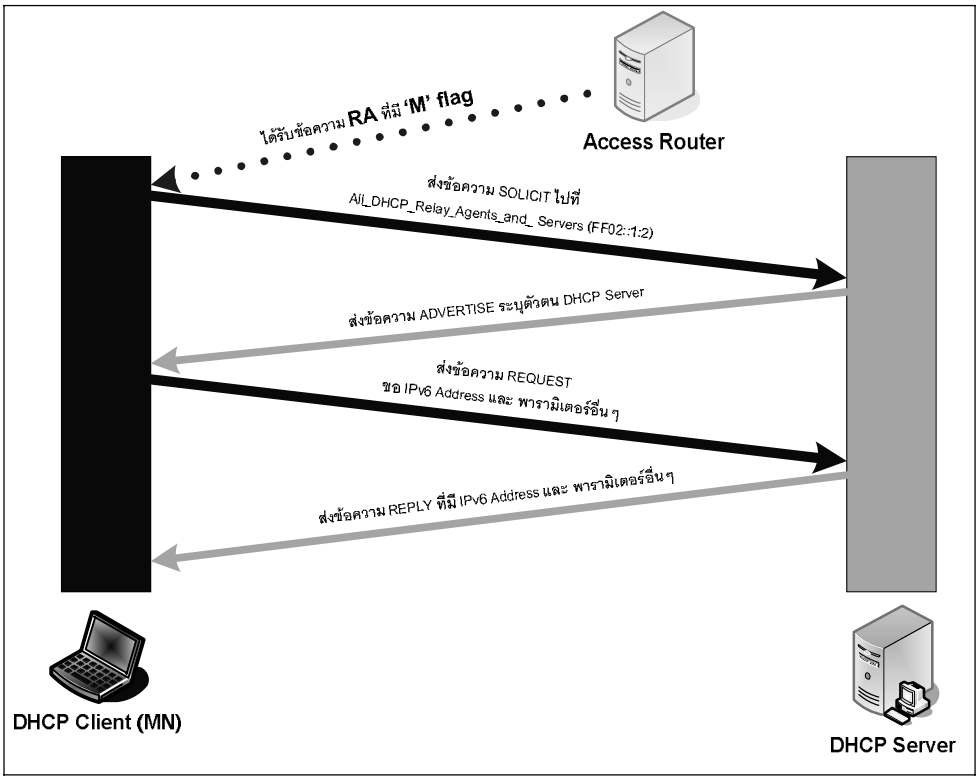
(5) MN เริ่มทำ Binding Update กับ Home Agent

การทำงานของ P-DAD สามารถแบ่งได้เป็นสองกรณีคือ กรณีที่หนึ่งคือกรณีที่สามารถทำนายการเคลื่อนย้ายของ MN ได้จะสามารถตรวจสอบ DAD ได้ล่วงหน้า เมื่อ MN มาถึงที่เครือข่ายใหม่สามารถใช้ Care-of Address ทำ Binding Update ได้เลย ซึ่งการที่สามารถทำให้การทำนายได้ผลจะต้องมี RIP Server เป็น Hardware ที่เพิ่มเติมขึ้นมาในกลุ่มเครือข่ายใดๆ และข้อความ

การติดต่อใหม่เพื่อการตรวจสอบ DAD ล่วงหน้า รวมถึงวิธีการทำนายการเคลื่อนที่ของ C.C. Tseng สังเกตได้ว่าการใช้วิธีการทำนายใช้ได้เฉพาะส่วนของเครือข่ายไร้สายตามมาตรฐาน IEEE 802.11 เท่านั้น กรณีที่สองคือกรณีที่ไม่สามารถทำนายได้หรือทำนายผิดพลาดจะทำงานขั้นตอน DAD แบบมาตรฐาน ซึ่งเสียเวลาการตรวจสอบประมาณ 1000 มิลลิวินาที

4. DHCPv6

Dynamic Host Configuration Protocol (DHCP) สำหรับ IPv6 นั้นมีลักษณะ ขั้นตอนการทำงาน และแนวคิดเหมือนกับ DHCP ที่ใช้กับ IPv4 นอกจากการพัฒนาและปรับเปลี่ยนให้เข้ากับการใช้งานระบบ IPv6 แล้วการเรียกชื่ออุปกรณ์ในระบบยังเหมือนเดิมคือมี DHCP Server คอยจัดสรรเลขหมาย IPv6 ให้ DHCP Client โดย DHCP Server ต้องจำสถานะการจัดสรรเลขหมายเพื่อป้องกันการจ่ายเลขหมาย IPv6 ซ้ำ ในส่วนของโพรโตคอลที่มีการปรับให้เหมาะสมกับระบบ IPv6 ได้แก่ ใช้ Multicast address All_DHCP_Relay_Agents_and_Servers (FF02::1:2) ในข้อความ SOLICIT ที่ DHCP Client ถาม DHCP Server แทนที่ใช้ Boardcast address (255.255.255.255) กำหนด UDP Port ใหม่ได้แก่ UDP port 546 เป็นช่องทาง DHCP Client รับข้อความและ UDP port 547 เป็นช่องทาง DHCP Server รับข้อความ และจากที่มีการใช้ MAC address ติดต่อสื่อสารกันก็เปลี่ยนมาเป็น DHCP Unique Identifier (DUID) ภายในมีชนิดของโครงสร้างข้อความ 3 ชนิดคือ DUID Based on Link-layer Plus Time (DUID-LIT), DUID Assigned by Vendor Based on Enterprise Number (DUID-EN) และ DUID Based on Link-layer (DUID-LL) มี Identifier Association ID (IAID) ใช้สำหรับระบุตัวตนของ DHCP Client ที่จะมาติดต่อ DHCP Server นั้นๆ ซึ่งจะไม่ใช้ซ้ำกันโดย DHCP Client เป็นคนสร้างขึ้นมาเอง



ภาพที่ 17 แสดงขั้นตอนการทำงานได้รับเลขหมาย IPv6 ตามขั้นตอน DHCPv6

ข้อความหลักๆในการรับ-ส่งขอเลขหมาย IPv6 มีการตอบรับทั้งหมด 4 ข้อความ ซึ่งอธิบายไปพร้อมกับขั้นตอนการทำงานตามภาพที่ 17 ดังนี้

(1) เมื่อ MN เข้ามาที่เครือข่ายใหม่ จะได้รับข้อความให้รู้ว่ามีการใช้ Stateful Address Autoconfiguration โดย Neighbor Discovery ทำให้ทราบได้ว่ามีการใช้ DHCPv6 อยู่ จากนั้น MN จะส่งข้อความถามหา DHCP Server ด้วยข้อความ SOLICIT ที่มี การสุ่มเลข transaction-id ที่ใช้สำหรับการระบุคู่ข้อความใดๆ และใส่ option เป็น Client Identifier option เป็นการบอก DUID ของ DHCP Client ส่งไปที่ All_DHCP_Relay_Agents_and_ Servers (FF02::1:2)

(2) เมื่อ DHCP Server ได้รับข้อความ SOLICIT จะตอบกลับด้วยข้อความ ADVERTISE ที่มี transaction-id เดียวกันกับข้อความ SOLICIT และใส่ option เป็น Server Identifier option กับ Client Identifier option เป็นการบอก DUID ของ DHCP Client และใช้เลขหมายต้นทางเป็นเลขหมาย DHCP Server

(3) เมื่อ MN หรือ DHCP Client ได้รับข้อความ ADVERTISE จะส่งข้อความ REQUEST ถึง DHCP Server โดยสุ่มเลข transaction-id ใหม่ และมีการใส่ option คือ Server Identifier option

กับ Client Identifier option และ Option request option ที่ภายในจะระบุ option-code ที่ต้องการให้ DHCP Server ส่งมาซึ่งเป็น configuration information ต่างๆ มีการใช้ Identifier Association for Non-temporary Address Option (IA_NA) หรือ Identifier Association for Temporary Address Option (IA_TA) แล้วแต่การจัดการของระบบ DHCP ซึ่งภายในจะมี IAID และใน IA_NA หรือ IA_TA ส่งไปที่ All_DHCP _Relay_Agents_and_Servers (FF02::1:2)

(4) เมื่อ DHCP Server ได้รับข้อความ REQUEST จะตรวจสอบ IAID และเก็บไว้เป็นข้อมูลว่า MN นี้ขึ้นกับ DHCP Server ตัวที่รับข้อความ และส่งข้อความ REPLY ตอบไปยัง DHCP Client ที่ถามมาโดยมี IA_NA หรือ IA_TA ที่มี option เป็น IA Address option เป็น option ใน IA_NA หรือ IA_TA ที่มี IPv6 Address อยู่ภายใน และใส่ option อื่นตามที่ Option request option ขอมา รวมทั้ง Server Identifier option กับ Client Identifier option ใช้ใช้เลขหมายต้นทางเป็นเลขหมาย DHCP Server

สังเกตได้ว่า การใช้งาน DHCPv6 นั้นอุปกรณ์ทุกตัวที่ใช้ระบบนี้ต้องรู้จักโพรโตคอลของ DHCPv6 และการทำงานแบบ Server-Client เป็นการทำงานเป็นมีศูนย์กลางโดยมี DHCP Server เป็นศูนย์กลาง DHCP และทำให้ DHCPv6 เป็น Stateful Address Autoconfiguration คือมีการใช้สถานะในการจดจำขั้นตอนต่างๆ ปัญหาที่พบใน DHCPv6 คือ ขั้นตอนการเรียกใช้เลขหมาย IPv6 ซ้ำอีกครั้ง (Renumbering) ทำได้ยาก และถ้ามีเลขหมาย IPv6 จำนวนมาก การจำเลขหมาย IPv6 ที่มีการใช้งานอยู่หรือไม่ได้ใช้งานเป็นไปได้ยาก ในเรื่องของการกำหนดขอบเขตของเลขหมาย IPv6 (Address range) ที่ต้องการจ่ายให้พอดีกับความต้องการของผู้ใช้ซึ่งถ้ามากเกินไปจะจัดการได้ยาก หรือถ้าน้อยเกินไปจะใช้ขั้นตอนใช้เลขหมาย IPv6 ซ้ำอีกครั้ง (Renumbering) บ่อยขึ้น และถ้า DHCP Server ไม่สามารถทำงานได้ DHCP Client จะไม่สามารถแลกเปลี่ยนข้อมูลที่จำเป็นสำหรับการสื่อสารได้เลย แต่ข้อดีของ DHCPv6 คือสามารถแจกจ่าย parameters ต่างๆ ได้ง่าย เช่น DNS parameters, Routing parameter ซึ่งวิธี Stateless Address Autoconfiguration ไม่สามารถส่ง parameters ดังกล่าวได้

อุปกรณ์และวิธีการ

อุปกรณ์

1. ฮาร์ดแวร์

- Intel Processor 1.73 GHz, 533 MHz, 2MB L2 cache)
- 1024MB DDR2
- 80GB HDD

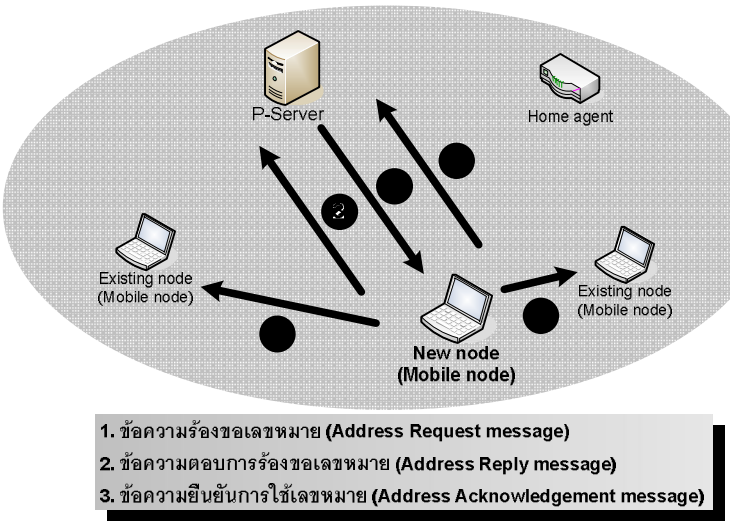
2. ซอฟต์แวร์

- Omnet++ Simulator
- Editor: GVim
- Operating System: Kubuntu Linux 7.04

วิธีการ

ขั้นตอน Duplicate Address Detection (DAD) ที่พัฒนาขึ้น

วิธีที่ผ่านมาของนักวิจัยหลายคนพัฒนาขึ้นมานั้น มีทั้งข้อดีและข้อเสียของการทำงานที่ต่างออกไปเพื่อให้เหมาะสมกับแต่ละวัตถุประสงค์ แนวคิดต่างๆ ที่ได้กล่าวไปนั้น ล้วนแล้วแต่ต้องการให้ขั้นตอน DAD ใช้เวลาลดลง ซึ่งเป็นเรื่องปกติที่เมื่อต้องการความเร็วของขั้นตอน DAD ต้องเพิ่มขั้นตอนวิธีการทำงาน เพิ่มอุปกรณ์ใดๆเข้าไปในระบบหรือลดความสามารถในการติดต่อสื่อสารลง แต่สิ่งที่เปลี่ยนแปลงไปนั้นต้องมีผลกระทบกับสภาพแวดล้อมของระบบในส่วนสำคัญรองลงมา จึงได้วิธีที่มีประสิทธิภาพเพิ่มขึ้น ขั้นตอน DAD ที่คิดค้นขึ้นใหม่นี้ มีแนวคิดที่นอกจากจะลดเวลาที่ใช้ในขั้นตอน DAD แล้วยังพิจารณาถึงความเข้ากันได้กับระบบมาตรฐานในระดับโพรโตคอล ซึ่งหากเพิ่มวิธีนี้เข้าไป อุปกรณ์จะสามารถทำงานได้เป็นปกติแม้ในอุปกรณ์ที่ไม่รู้จักวิธีนี้ และการออกแบบคำนึงถึงความต่อเนื่องของการทำงานของแต่ละอุปกรณ์เพื่อประสิทธิภาพการทำงานที่ดีขึ้น



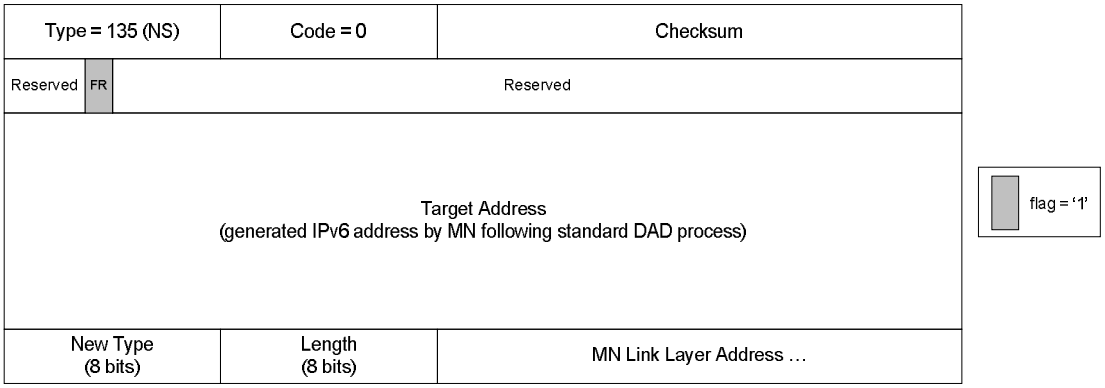
ภาพที่ 18 แสดงโครงสร้างของอุปกรณ์ของวิธีใหม่

วิธีใหม่ตามภาพที่ 18 นี้เรียกว่า Fast and Robust DAD (FR-DAD) โดยใช้การทำงานที่เร็วและมีความทนทานต่อความผิดพลาดในการสื่อสารได้ FR-DAD มีแนวคิดการทำงานเป็นแบบมีศูนย์กลางการทำงาน (Centralized) โดยเพิ่มอุปกรณ์ Pool uniquess IP address server หรือ P-Server ที่จัดการทำขั้นตอน DAD แบบมาตรฐานและเก็บจำนวนของเลขหมาย IPv6 ไว้จำนวนหนึ่งไว้ในส่วนสำรองเลขหมาย (pool) ที่ไม่มีผลกับระบบแม้ว่าไม่มี P-Server อยู่ในเครือข่าย P-Server มีลักษณะการทำงานคล้ายกับ access router ของ A-DAD ซึ่งวิธีใหม่นี้ได้แยกการทำงานส่วนการจัดการสำรองเลขหมายออกมาเป็นอุปกรณ์ใหม่เพื่อลดการทำงานที่มากเกินไปของ access router P-Server ออกแบบให้เฝ้าฟังการตรวจสอบ DAD แบบมาตรฐานของเลขหมาย IPv6 ที่ MN สุ่มขึ้นมาและส่งออกมาพร้อมกันข้อความร้องขอเลขหมายกับ P-Server เมื่อรอจนเวลาครบ 1000 มิลลิวินาที P-Server จะนำเลขหมายดังกล่าวไปเก็บใน pool ทำให้ P-Server ไม่เสียเลขหมาย IPv6 ใน pool ไปเลยในขั้นตอนเริ่มต้นที่ยังไม่มีอุปกรณ์ใดๆ อยู่ในเครือข่าย P-Server สามารถกำหนดเลขหมายเท่าจำนวน pool ได้เลย แต่ถ้าเป็นการเพิ่ม P-Server เข้าไปกับเครือข่ายที่มีการทำงานอยู่แล้ว P-Server ต้องทดสอบเลขหมาย IPv6 จนครบจำนวน pool แต่สามารถให้บริการจ่ายเลขหมาย IPv6 ได้ขณะที่ยังไม่เต็ม pool FR-DAD นี้ไม่ใช้การทำนาย Layer 2 handover เพราะออกแบบให้ทำงานได้ทั้งระบบ LAN และ Wireless LAN

ขั้นตอนของการสื่อสารของ FR-DAD คือ เมื่อ MN ย้ายเข้ามาในระบบจะส่งข้อความร้องขอเลขหมาย (Address Request) มาที่ P-Server และยังเป็นข้อความเดียวกันกับที่ MN ส่งถามใน

ขั้นตอน DAD แบบมาตรฐาน จากนั้นเมื่อ P-Server ได้รับข้อความนั้น จะส่งข้อความพร้อมเลขหมาย IPv6 ที่ยังไม่มีอุปกรณ์ใดใช้แน่นอนหรือข้อความตอบการร้องขอเลขหมาย (Address Reply) มาให้ MN พร้อมทั้งรอฟังเลขหมาย IPv6 ที่ MN กำลังทดสอบตามขั้นตอน DAD แบบมาตรฐานอยู่ เมื่อ MN ได้รับข้อความที่มีเลขหมาย IPv6 ที่ยังไม่มีอุปกรณ์ใดใช้แน่นอน MN จะนำเลขหมายนี้เริ่มใช้งานเป็นเลขหมาย CoA ใหม่และส่งข้อความยืนยันการใช้เลขหมายที่ได้จากข้อความ Address Reply (Address Acknowledgment) ซึ่งในการทำงานของวิธีใหม่นี้ใช้ลักษณะการสื่อสารของข้อความเป็นแบบสามทาง (Three-way handshake) เพื่อเพิ่มทางเลือกของ MN ในกรณีที่ไม่ได้รับข้อความ Address Reply จะใช้เลขหมายที่ได้ทดสอบโดยวิธี DAD แบบมาตรฐานทำให้เลขหมายในข้อความ Address Acknowledgment เป็นเลขหมายที่ทดสอบจากวิธี DAD แบบมาตรฐานซึ่งลดเวลาการรอข้อความหรือ timeout จากการทำงานของวิธีที่เพิ่มขึ้นมา จากนั้นเมื่อ P-Server ได้รับข้อความ Address Acknowledgment และครบการรอเป็นเวลา 1000 มิลลิวินาที P-Server จะเก็บเลขหมาย IPv6 ที่ MN ไม่ได้ใช้สำหรับเลขหมาย CoA ลงไปในส่วนสำรองเลขหมาย IPv6 ที่ยังไม่มีอุปกรณ์ใดใช้ต่อไป

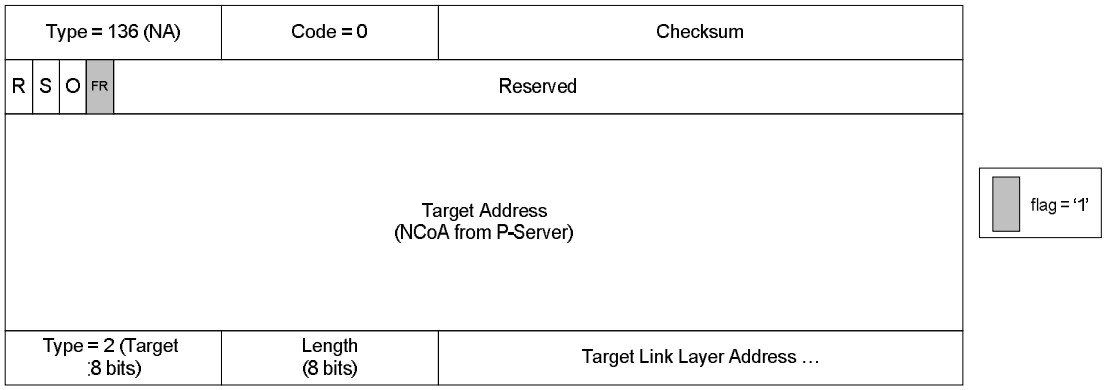
ลักษณะของข้อความใหม่ในวิธี FR-DAD เป็นข้อความที่ปรับจากข้อความมาตรฐาน Neighbor Discovery สำหรับเลขหมาย IPv6 เป็นข้อความ ICMPv6 ที่ไม่มีผลกระทบกับการสื่อสารมาตรฐานใช้การเพิ่ม ‘FR’ flag และ option ใหม่



ภาพที่ 19 แสดงโครงสร้างของข้อความ Address Request

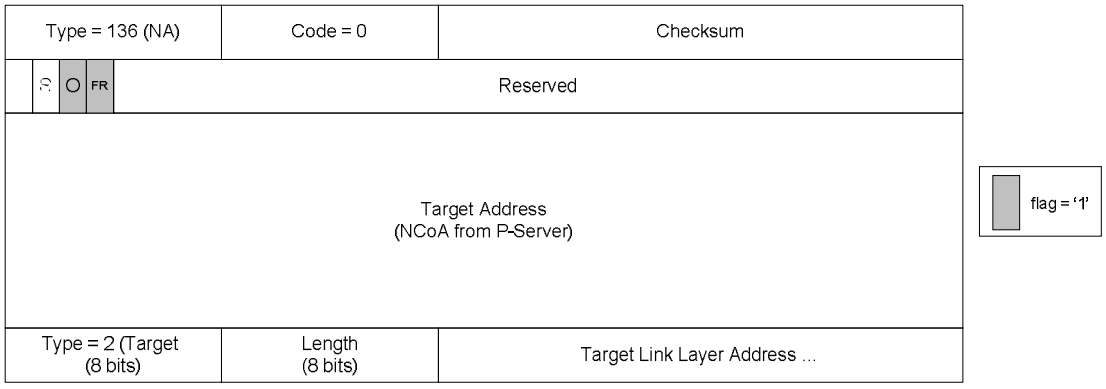
จากภาพที่ 19 เป็นข้อความ Address Request ซึ่งเป็นข้อความ NS ที่มีโครงสร้างคล้ายกับข้อความ NS ของ DAD แบบมาตรฐานที่ Target address เป็นเลขหมาย IPv6 ที่ MN คำนวณขึ้นเพื่อใช้ทดสอบ DAD แบบมาตรฐานไปพร้อมกับการร้องขอเลขหมายจาก P-Server การเพิ่ม ‘FR’ flag

ในบิตที่ 4 ของส่วน Reserved และมี option ใหม่ที่บรรจุ link-layer address ของ MN ช่วยในการสื่อสารกับ P-Server และส่วนที่เพิ่มขึ้นมานี้ไม่มีผลกระทบกับการทำงานของ DAD แบบมาตรฐาน



ภาพที่ 20 แสดงโครงสร้างของข้อความ Address Reply

จากภาพที่ 20 เป็นข้อความ Address Reply ซึ่งเป็นข้อความ NA ที่ส่งออกจาก P-Server เพื่อส่งเลขหมาย IPv6 ใน pool เพื่อให้ MN ใช้เป็นเลขหมาย CoA ใหม่ (NCoA) ในเครือข่ายปัจจุบัน โดยมี Target link-layer address เป็น MN link-layer address และ 'FR' flag เป็นส่วนที่จะระบุให้ MN ทราบว่าเป็นข้อความที่ส่งถึง MN จาก P-Server เท่านั้น

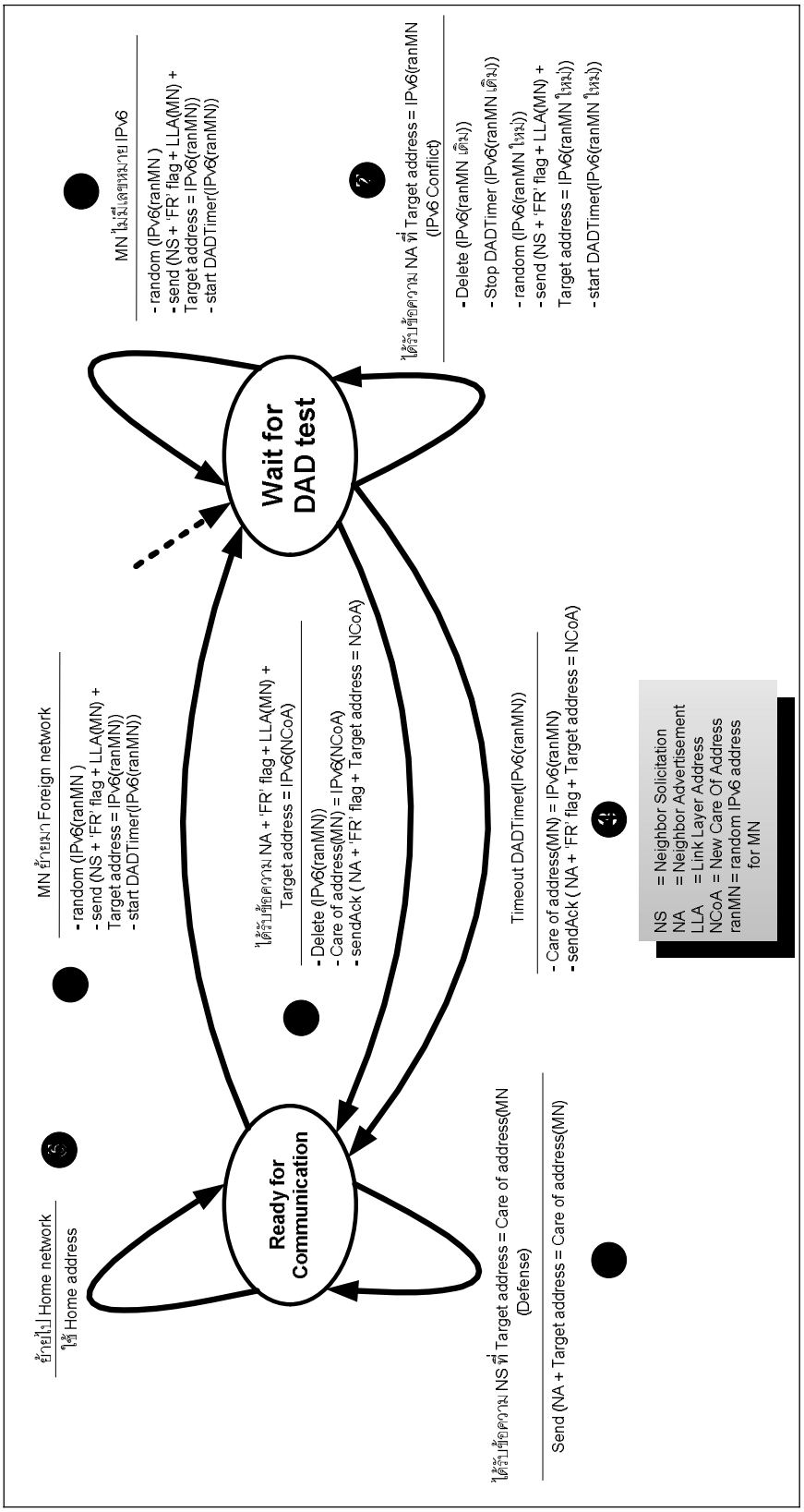


ภาพที่ 21 แสดงโครงสร้างของข้อความ Address Acknowledgment

จากภาพที่ 21 เป็นข้อความ Address Acknowledgment ซึ่งเป็นข้อความ NA ที่ส่งออกจาก MN เพื่อยืนยันการใช้เลขหมาย CoA ของ MN ว่าจะใช้เลขหมายที่ส่งจาก P-Server (NCoA) หรือเป็นเลขหมายที่เกิดจากวิธี DAD แบบมาตรฐาน ทั้งนี้ขึ้นอยู่กับความผิดพลาดของการสื่อสารที่

เกิดขึ้นซึ่งโพรโตคอลได้ออกแบบให้ไม่สูญเสียเวลาเมื่อมีการสูญหายของข้อความหรือการไม่มีอยู่ของ P-Server ในบางเครือข่าย ในข้อความ Address Acknowledgment ใช้ Target link-layer address เป็น MN link-layer address และ ‘FR’ flag กับ ‘O’ flag เป็นลักษณะของข้อความที่ส่งให้กับ P-Server เข้าใจการตัดสินใจการใช้งานเลขหมาย CoA ของ MN และยังเป็นข้อความที่ส่งกับ neighbor ในครั้งสุดท้ายที่จะปรับการจับคู่ระหว่างเลขหมาย IPv6 กับ link-layer address ใน neighbor entry ของ neighbor ในเครือข่ายเดียวกัน โดยมี ‘O’ flag หรือ override flag เป็นการบอกถึงการแก้ไข ซึ่ง P-Server จะรอข้อความนี้ในช่วงเวลา Acknowledgment timeout โดยต้องกำหนดให้มากกว่า 1000 มิลลิวินาทีเพราะในกรณีที่ MN ใช้เลขหมาย CoA จากการทดสอบ DAD แบบมาตรฐานจะต้องใช้เวลาอย่างน้อย 1000 มิลลิวินาที ซึ่งในการทดลองนี้กำหนดให้ Acknowledgment timeout อยู่ที่ 1500 มิลลิวินาที

จากแนวคิดนี้ได้ออกแบบโพรโตคอลมาในรูปแบบของ Finite State Machine (FSM) ที่แบ่งเป็น 3 ส่วนคือ การทำงานของ MN (ภาพที่ 22) การทำงานของ Server กรณีเก็บเลขหมาย IPv6 (ภาพที่ 23) และการทำงานของ P-Server กรณีจ่ายเลขหมาย IPv6 (ภาพที่ 24)



ภาพที่ 22 FSM การทำงานของ MN ตามวิธี FR-DAD

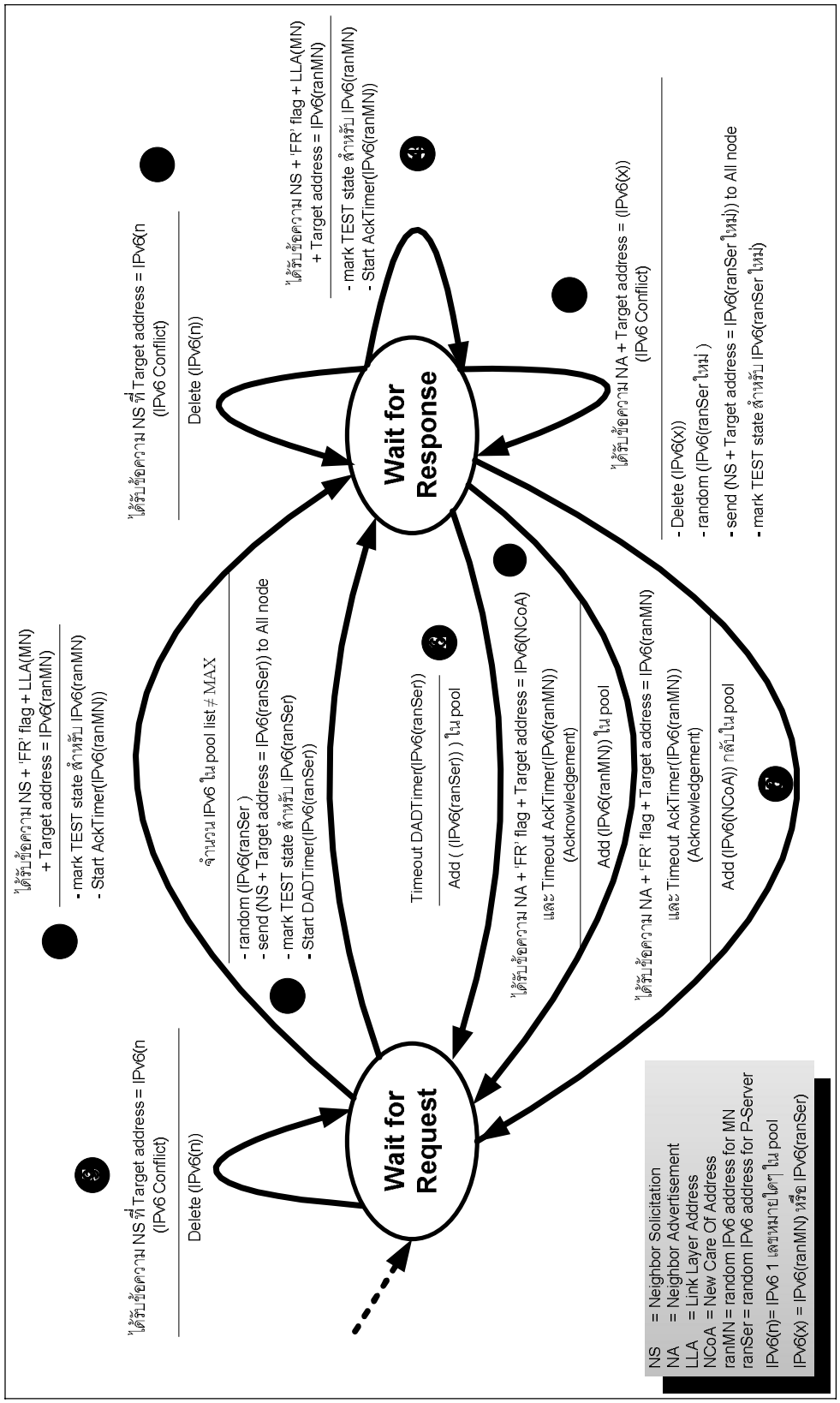
จากภาพที่ 22 เป็นสถานการณ์ทั้งหมดของ MN ที่ใช้ในการทำงานตามวิธีใหม่มีอยู่ 2 สถานะคือสถานะ MN มีเลขหมาย IPv6 ที่ใช้ในเครือข่ายใดๆ ที่สามารถใช้งานติดต่อสื่อสารได้ตามปกติ (Ready for Communication) และสถานะรอการตอบรับหรือระหว่างทำขั้นตอน DAD (Wait for test DAD) ขั้นตอนการทำงานเริ่ม MN ยังไม่มีเลขหมาย IPv6 ใช้งานในเครือข่ายจะทำตามขั้นตอน (1) ในภาพที่ 22 คือการส่งข้อความ NS ถามว่าเลขหมายที่ MN สุ่มขึ้นหรือ IPv6(ranMN) มีใครใช้งานหรือยังตามขั้นตอน DAD แบบมาตรฐาน แต่เพิ่ม 'FR' flag และ option ใหม่ตาม ภาพที่ 19 ไปเป็นลักษณะของข้อความ Address Request ที่ติดต่อกับ P-Server ด้วย ถ้าเป็นเครือข่ายที่เป็นวิธีใหม่เมื่อ P-Server ได้รับข้อความ Address Request จะเข้าใจถึงการร้องขอเลขหมาย IPv6 ที่ยังไม่มีการใช้งานหรือ IPv6(NCoA) P-Server จะส่งข้อความ Address Reply ตามภาพที่ 20 ที่ Target address เป็นเลขหมาย IPv6(NCoA) และ Link Layer address option ของ MN เป็นการระบุการรับข้อความ ซึ่งเมื่อ MN ได้รับข้อความ Address Reply จะนำเลขหมาย IPv6(NCoA) เป็นเลขหมาย CoA ของ MN และส่งข้อความ Address Acknowledgment ตามภาพที่ 21 ที่ Target address เป็นเลขหมาย IPv6(NCoA) ตามขั้นตอน (3) ในภาพที่ 22 แต่ถ้าในเครือข่ายไม่มีการทำงานของวิธีใหม่ MN จะรอคำตอบภายใน 1000 มิลลิวินาทีและใช้เลขหมาย IPv6(ranMN) เป็นเลขหมาย CoA ของ MN และส่งข้อความ Acknowadgement เป็นข้อความ NA ที่ Target address เป็นเลขหมาย IPv6(ranMN) เพิ่ม flag ตามขั้นตอน (4) ในภาพที่ 22

กรณีที่ MN จะย้ายจาก Home network ไปยัง Foreign network ซึ่ง MN ต้องการเลขหมาย CoA ไว้ใช้งานใน Foreign network เมื่อมาถึง Foreign network ยังไม่มีเลขหมาย CoA ใช้งานและไม่ทราบว่าที่ที่เครือข่ายนี้มีการใช้วิธี FR-DAD จะทำตามขั้นตอน (7) ในภาพที่ 22 คือการส่งข้อความ NS ถามว่าเลขหมายที่ MN สุ่มขึ้นหรือ IPv6(ranMN) มีใครใช้งานหรือยังตามขั้นตอน DAD แบบมาตรฐาน แต่เพิ่ม 'FR' flag และ option ใหม่ตาม ภาพที่ 19 ไปเป็นลักษณะของข้อความ Address Request ที่ติดต่อกับ P-Server ด้วย ถ้าเป็นเครือข่ายที่เป็นวิธีใหม่เมื่อ P-Server ได้รับข้อความ Address Request จะเข้าใจถึงการร้องขอเลขหมาย IPv6 ที่ยังไม่มีการใช้งานหรือ IPv6(NCoA) P-Server จะส่งข้อความ Address Reply ตามภาพที่ 20 ที่ Target address เป็นเลขหมาย IPv6(NCoA) และ Link Layer address option ของ MN เป็นการระบุการรับข้อความ ซึ่งเมื่อ MN ได้รับข้อความ Address Reply จะนำเลขหมาย IPv6(NCoA) เป็นเลขหมาย CoA ของ MN และส่งข้อความ Address Acknowledgment ตามภาพที่ 21 ที่ Target address เป็นเลขหมาย IPv6(NCoA) ตามขั้นตอน (3) ในภาพที่ 22 แต่ถ้าในเครือข่ายไม่มีการทำงานของวิธีใหม่ MN จะรอคำตอบภายใน 1000 มิลลิวินาทีและใช้เลขหมาย IPv6(ranMN) เป็นเลขหมาย CoA ของ MN และส่งข้อความ

Acknowadgement เป็นข้อความ NA ที่ Target address เป็นเลขหมาย IPv6(ranMN) เพิ่ม flag ตามขั้นตอน (4) ในภาพที่ 22

ในระหว่างการรอคำตอบนั้นมีข้อความ NA ที่ Target address เป็นเลขหมาย IPv6(ranMN) เป็นการบอกว่าเลขหมาย IPv6(ranMN) มีการใช้งานแล้ว (IP conflict) MN ต้องส่งข้อความ Address Request ที่ใช้เลขหมาย IPv6(ranMN) ที่สุ่มขึ้นมาใหม่ รอการตอบรับอีกครั้งตามขั้นตอน (2) ในภาพที่ 22

ในกรณีที่ MN มีการใช้งานเลขหมาย IPv6 อยู่นั้น ได้รับข้อความ NS ที่ Target address เป็นเลขหมาย CoA ของ MN เป็นการถามตามขั้นตอน DAD แต่ MN มีการใช้งานอยู่แล้ว MN ต้องส่งข้อความ NA ที่ Target address เป็นเลขหมาย CoA ของ MN เป็นการบอกว่า MN ใช้เลขหมาย IPv6 นี้อยู่ตามขั้นตอน (5) ในภาพที่ 22 และกรณีที่ MN ย้ายกลับไป Home network สามารถนำ Home address มาใช้ได้เลยตามขั้นตอน (6) ในภาพที่ 22



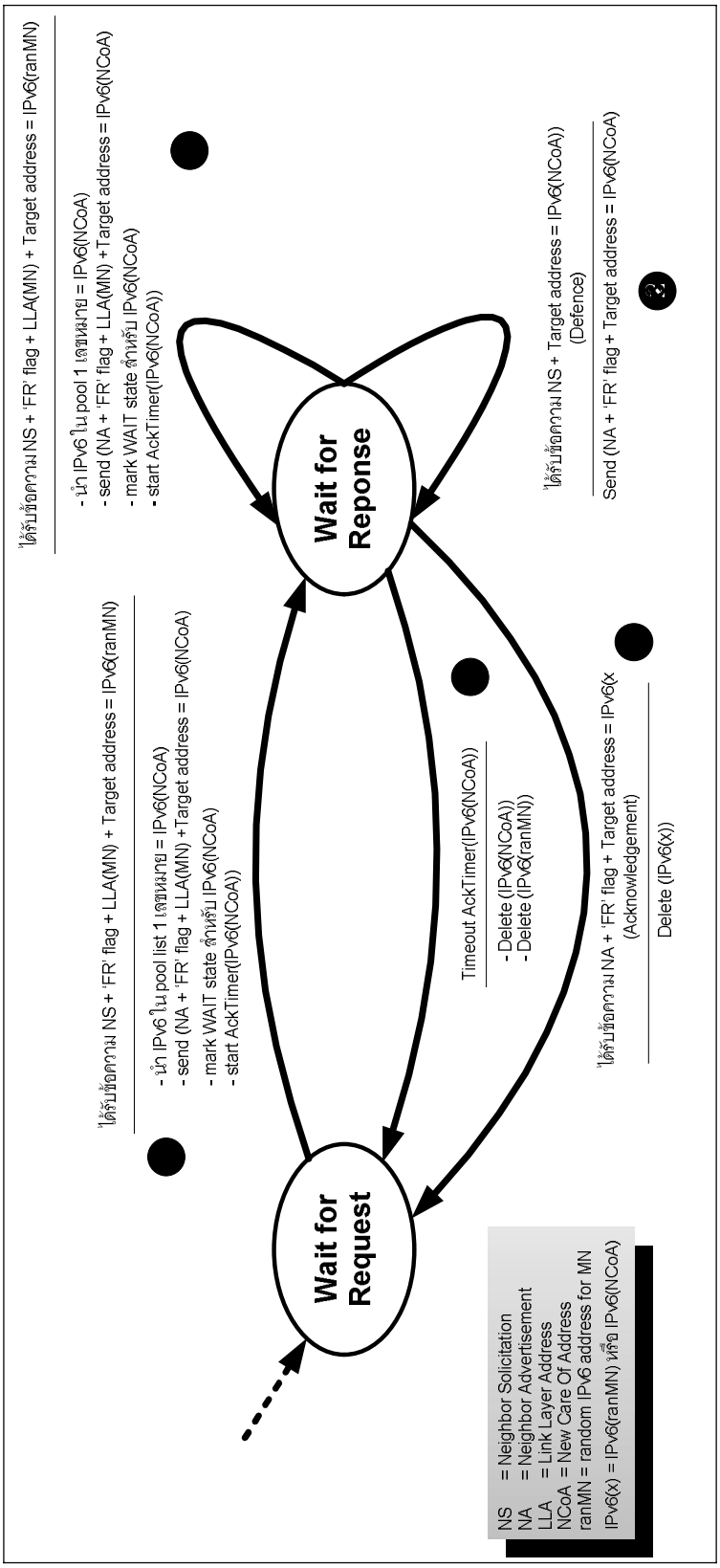
ภาพที่ 23 FSM การทำงานของ P-Server ในการเก็บเลขหมาย IPv6

ภาพที่ 23 เป็น FSM ของ P-Server กรณีเก็บเลขหมาย IPv6 สำรองไว้ (pool) มี 2 สถานะคือ รอการร้องขอ (Wait for Request) เป็นสถานะที่ pool มีเท่ากับจำนวนสูงสุดที่ได้กำหนดไว้ และสถานะรอการตอบรับ (Wait for Response) เป็นสถานะที่รอการตอบรับของขั้นตอนการตรวจสอบขั้นตอน DAD การทำงานจะเปลี่ยนสถานะได้นั้นเริ่มจากเลขหมาย IPv6 ที่ผ่านขั้นตอน DAD แล้วใน pool มีจำนวนน้อยกว่าที่กำหนดไว้จึงต้องตรวจสอบเลขหมายเพื่อเก็บเพิ่มโดยมี 2 กรณีในการเก็บเลขหมายคือขั้นตอน (1) ในภาพที่ 23 ได้รับข้อความร้องขอเลขหมาย IPv6(NCoA) ทำให้ใช้เลขหมาย IPv6 ใน pool ไปหนึ่งเลขหมายแต่การส่งข้อความร้องขอนี้เป็นข้อความ NS หรือข้อความ Address Request ที่เป็นการตรวจสอบตามขั้นตอน DAD อยู่แล้วโดยมีเลขหมาย IPv6 ที่ MN สุ่มมาเป็น IPv6(ranMN) P-Server จึงทำหน้าที่รอการตอบรับของข้อความทดสอบนี้ด้วยภายในเวลา timeout ของข้อความ Address Acknowledgment ถ้าได้รับข้อความ NA ที่ Target address = IPv6(ranMN) แสดงว่าเลขหมายนี้มีการใช้แล้ว P-Server จะหยุดการตรวจสอบและลบเลขหมาย IPv6(ranMN) ทิ้งและสุ่มเลขหมาย IPv6(ranSer) ขึ้นมาใหม่ นำไปทดสอบตามขั้นตอน DAD อีกครั้งรอการตรวจสอบ 1000 มิลลิวินาทีตามขั้นตอน (5) ในภาพที่ 23 และในขณะรอการตอบรับมี MN ร้องขอเลขหมาย IPv6(NCoA) มาอีก node ก็รอการตอบรับของข้อความทดสอบนี้ด้วยภายในเวลา timeout ของข้อความ Address Acknowledgment อีกครั้งตามขั้นตอน (4) ในภาพที่ 23

และกรณีที่สอง pool มีจำนวนน้อยกว่าที่กำหนด P-Server จะสุ่มเลขหมาย IPv6(ranSer) ขึ้นมาและส่ง NS ทดสอบตามขั้นตอน DAD แบบมาตรฐานรอภายในเวลา 1000 มิลลิวินาทีตามขั้นตอน (2) ในภาพที่ 23 และถ้ามีข้อความ NA ที่ Target address = IPv6(ranSer) แสดงว่าเลขหมายนี้มีการใช้แล้ว P-Server จะหยุดการตรวจสอบและลบเลขหมาย IPv6(ranSer) เดิมทิ้งและสุ่มเลขหมาย IPv6(ranSer) ขึ้นมาใหม่ นำไปทดสอบตามขั้นตอน DAD อีกครั้งรอการตรวจสอบ 1000 มิลลิวินาทีตามขั้นตอน (5) ในภาพที่ 23 และเมื่อครบเวลา 1000 มิลลิวินาทีจะนำเลขหมาย IPv6(ranSer) เก็บเข้าไปใน pool ตามขั้นตอน (6) ในภาพที่ 23

จากกรณีที่รอเลขหมาย IPv6(ranMN) เมื่อเวลา timeout ของข้อความ Address Acknowledgment หมดลงและได้รับข้อความ Address Acknowledgment ที่ตอบกลับมาด้วยเลขหมาย IPv6(NCoA) ก็นำเลขหมาย IPv6(ranMN) เก็บเข้าไปใน pool ตามขั้นตอน (7) ในภาพที่ 23 แต่ถ้าในกรณีเดียวกันรับข้อความ Address Acknowledgment ที่ตอบกลับมาด้วยเลขหมาย IPv6(ranMN) ก็นำเลขหมาย IPv6(NCoA) เก็บกลับเข้าไปใน pool ตามขั้นตอน (8) ในภาพที่ 23

ถ้ามีข้อความ NS ถามจากขั้นตอน DAD จากอุปกรณ์ในเครือข่ายที่ Target address ตรงกับเลขหมาย IPv6 ใดๆใน pool ต้องลบเลขหมายนั้นออกจาก P-Server ทันทีไม่ว่าจะอยู่ที่สถานะใดก็ตามในขั้นตอน (3) และ (9) ในภาพที่ 23



ภาพที่ 24 FSM การทำงานของ P-Server ในการจ่ายเลขหมาย IPv6

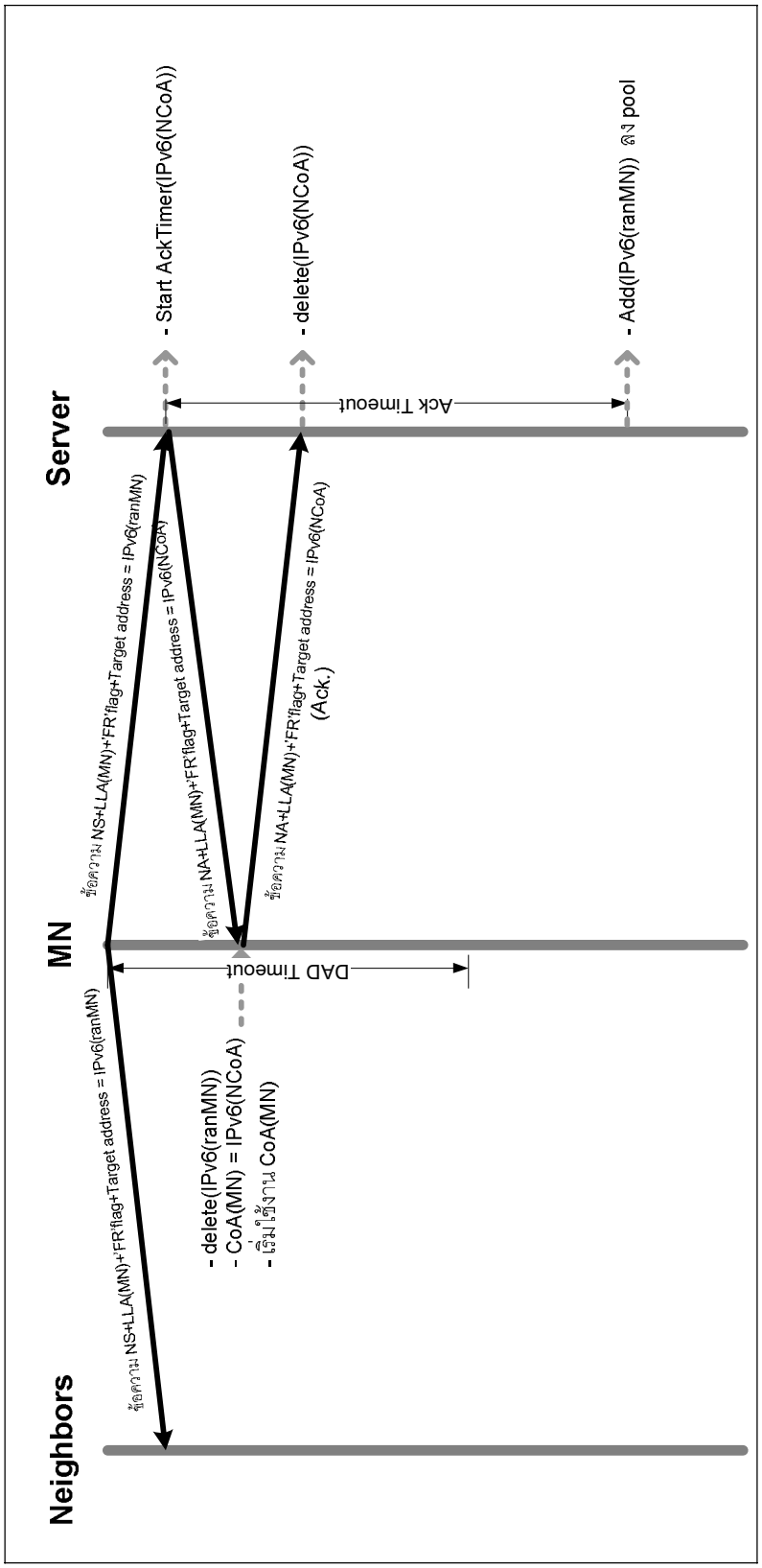
ภาพที่ 24 เป็น FSM ของ P-Server กรณีจ่ายเลขหมาย IPv6 มี 2 สถานะคือ รอการร้องขอ (Wait for Request) เป็นสถานะที่รอข้อความร้องขอเลขหมาย IPv6 และสถานะรอการตอบรับ (Wait for Response) เป็นสถานะที่รอการตอบกลับข้อความ Address Acknowledgment

การได้รับเลขหมาย IPv6(NCoA) ที่ P-Server จ่ายไปให้กับ MN ที่ต้องการ การทำงานของกรณีนี้เริ่มจากได้รับข้อความข้อความ Address Request ของ MN ที่ต้องการเป็นข้อความ NS ที่มี Target address เป็นเลขหมาย IPv6(ranMN) หลังจากนั้น P-Server จะส่งข้อความ Address Reply ที่ Target address เป็นเลขหมาย IPv6(NCoA) ที่อยู่ใน pool ที่มี Target Link Layer address option เป็นของ MN เป็นการระบุการรับข้อความ และรอการตอบกลับของ MN ว่าได้รับเลขหมาย IPv6(NCoA) แล้วตามขั้นตอน (1) และ (2) ในภาพที่ 24

ในขณะที่อยู่สถานะรอการตอบรับ ถ้ามีข้อความ NS ที่ Target address เป็นเลขหมาย IPv6(NCoA) P-Server จะส่งข้อความ NA ที่ Target address เป็นเลขหมาย IPv6(NCoA) และมี ‘FR’ flag เป็นการป้องกันการชนกันของเลขหมายโดยการส่งข้อความว่ามีการใช้งานอยู่แล้วตามขั้นตอน (3) ในภาพที่ 24

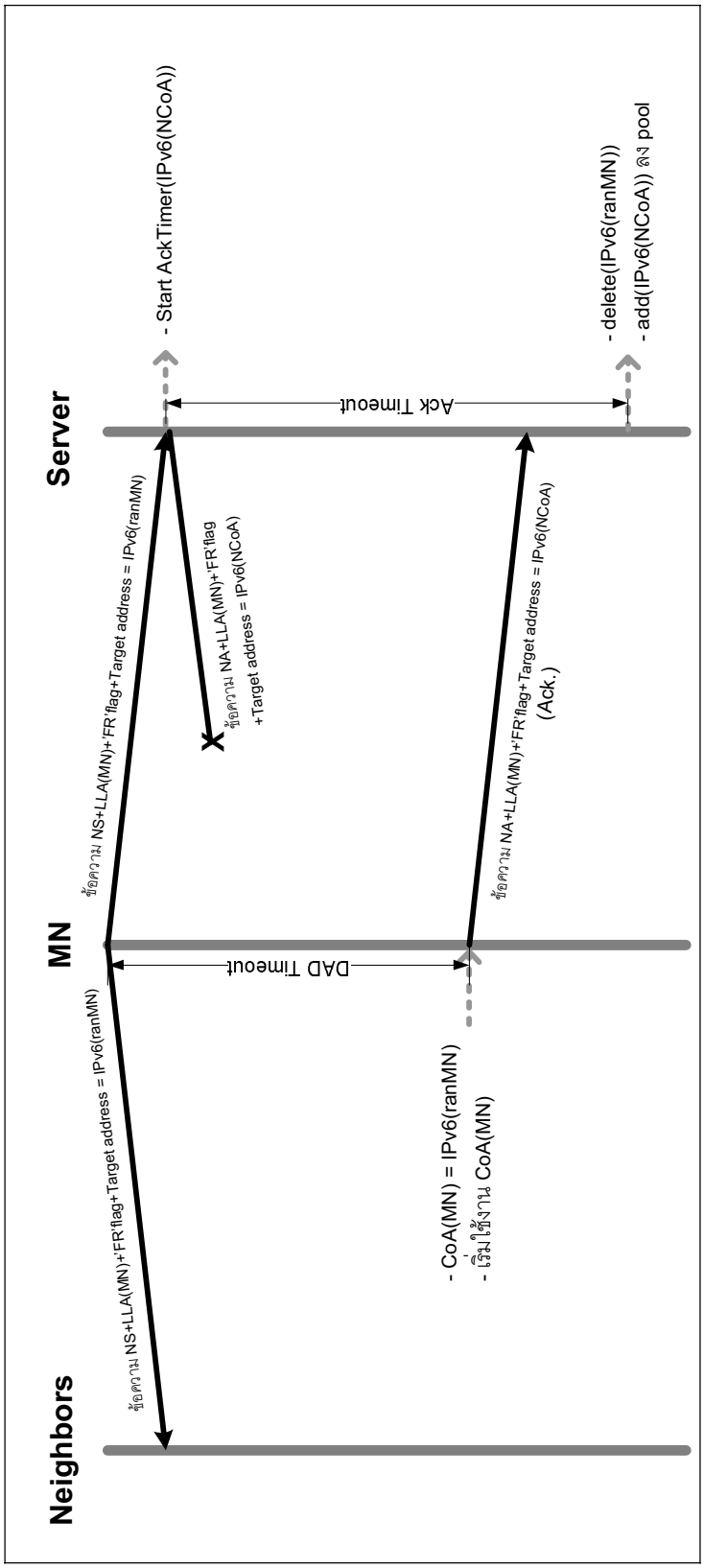
จากนั้นถ้า P-Server ได้รับข้อความ Address Acknowledgment ที่ Target address เป็นเลขหมาย IPv6(NCoA) เป็นข้อความที่บอกว่า MN ได้ใช้เลขหมาย IPv6(NCoA) เป็นเลขหมาย CoA แล้ว P-Server จะลบเลขหมาย IPv6(NCoA) ออกจาก pool และในลักษณะเดียวกัน ถ้า P-Server ได้รับข้อความ Address Acknowledgment ที่ Target address เป็นเลขหมาย IPv6(ranMN) เป็นข้อความที่บอกว่า MN ได้ใช้เลขหมาย IPv6(ranMN) เป็นเลขหมาย CoA แล้ว P-Server จะลบเลขหมาย IPv6(ranMN) ออกจาก pool ตามขั้นตอน (5) ในภาพที่ 24 และถ้าที่ไม่ได้รับข้อความ Address Acknowledgment เลยและเวลาการรอ Timeout ของข้อความ Address Acknowledgment หมดลง P-Server จะลบเลขหมาย IPv6(NCoA) และ IPv6(ranMN) ทันทีตามขั้นตอน (4) ในภาพที่ 24

จาก FSM ที่แสดงไปนั้น เพื่อให้เข้าใจการทำงานของวิธีใหม่เพิ่มขึ้นได้จำลองสถานการณ์ออกเป็น 4 สถานการณ์หลักเป็น Timing diagram ดังนี้



ภาพที่ 25 Timing diagram กรณีที่ไม่มีความผิดพลาด (Ideal case)

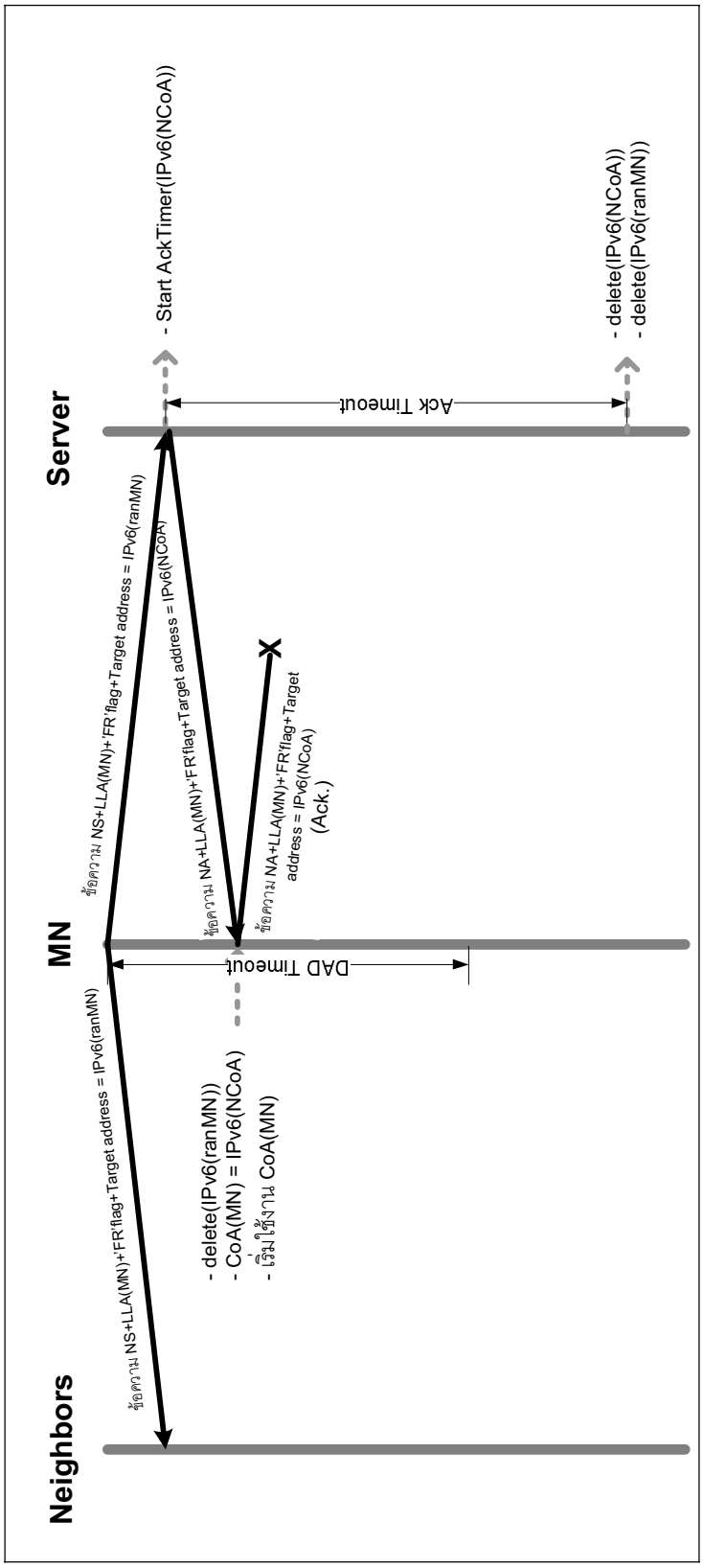
กรณีแรกเป็นกรณีที่ไม่มีความผิดพลาดของขั้นตอนการจ่ายเลยหมาย IPv6(NCoA) ตามภาพที่ 25 เริ่มจาก MN ย้ายเข้ามาใน Foreign network จะส่งข้อความ Address Request และทำขั้นตอน DAD ด้วยเลขหมาย IPv6(ranMN) ไปด้วยคือข้อความ NS ที่มี ‘FR’ flag MN Link Layer address option และ Target address เป็นเลขหมาย IPv6(ranMN) ซึ่งใน Foreign network นี้มี P-Server ที่เตรียมเลขหมาย IPv6(NCoA) ไว้ใน pool เรียบร้อยแล้ว เมื่อ P-Server ได้รับข้อความ Address Request ที่เป็นการร้องขอเลขหมาย IPv6 จะส่งข้อความ Address Reply ที่ Target address เป็นเลขหมาย IPv6(NCoA) เป็นการจ่ายเลขหมาย IPv6(NCoA) ไปและเริ่มรอการตรวจสอบขั้นตอน DAD เลขหมาย IPv6(ranMN) ไปพร้อมกัน เมื่อ MN ได้รับข้อความข้อความ Address Reply จะนำเลขหมาย IPv6(NCoA) จะนำใช้เป็นเลขหมาย CoA ของ MN และลบเลขหมาย IPv6(ranMN) ออกจากการตรวจสอบและส่งข้อความ Address Acknowledgment เป็นข้อความ NA ที่ Target address เป็นเลขหมาย IPv6(NCoA) ในขั้นตอนนี้ MN สามารถเริ่มการติดต่อสื่อสารได้เลย จากนั้นเมื่อ P-Server ได้รับข้อความ Address Acknowledgment จาก MN จะลบเลขหมาย IPv6(NCoA) ออกจาก pool และเมื่อ timeout ของการรอข้อความ Address Acknowledgment หมดลง P-Server จะนำเลขหมาย IPv6(ranMN) เข้า pool จึงจบขั้นตอน



ภาพที่ 26 Timing diagram กรณีที่ไม่ได้รับเลขหมาย IPv6(NCoA)

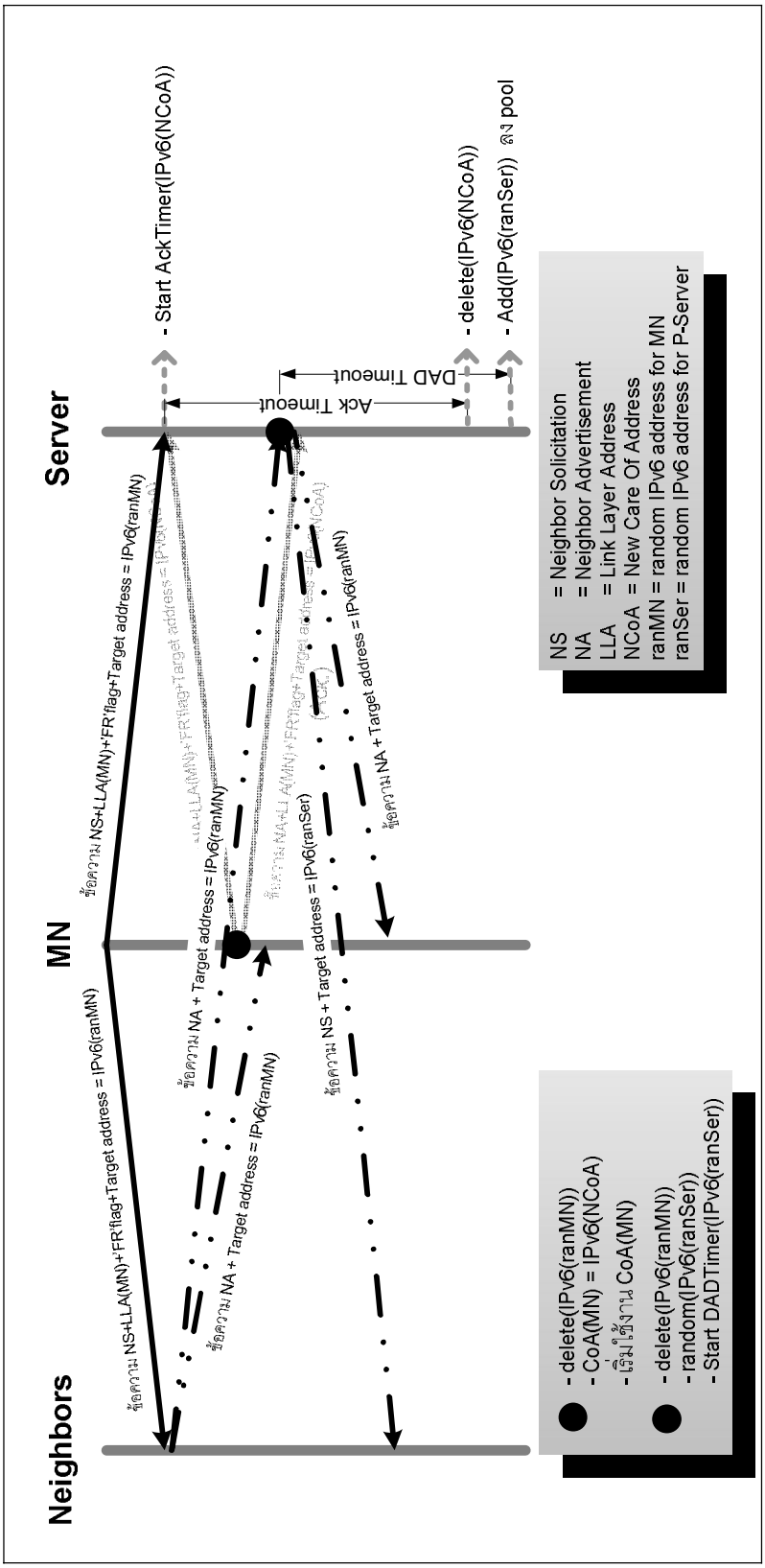
กรณีที่สองเป็นกรณีที่ไม่ได้รับเลขหมาย IPv6(NCoA) ตามภาพที่ 26 เริ่มจาก MN ย้ายเข้ามาใน Foreign network จะส่งข้อความ Address Request และทำขั้นตอน DAD ด้วยเลขหมาย IPv6(ranMN) ไปด้วยคือข้อความ NS ที่มี 'FR' flag MN Link Layer address option และ Target address เป็นเลขหมาย IPv6(ranMN) ซึ่งใน Foreign network นี้มี P-Server ที่เตรียมเลขหมาย IPv6(NCoA) ไว้ใน pool เรียบร้อยแล้ว เมื่อ P-Server ได้รับข้อความ Address Request ที่เป็นการร้องขอเลขหมาย IPv6 จะส่งข้อความ Address Reply ที่ Target address เป็นเลขหมาย IPv6(NCoA) เป็นการจ่ายเลขหมาย IPv6(NCoA) ไปและเริ่มรอการตรวจสอบขั้นตอน DAD เลขหมาย IPv6(ranMN) ไปพร้อมกัน แต่ข้อความ Address Reply ที่มีเลขหมาย IPv6(NCoA) นั้น ไปไม่ถึง MN ซึ่ง MN จะรอข้อความนี้จนกระทั่ง 1000 มิลลิวินาที เลขหมาย IPv6(ranMN) ที่ได้ทำงานตรวจสอบนั้นเรียบร้อยพร้อมนำไปใช้ MN จะนำเลขหมาย IPv6(ranMN) เป็นเลขหมาย CoA ทันทีและเริ่มการติดต่อสื่อสารได้ พร้อมทั้งส่งข้อความ Address Acknowledgment ที่ Target address เป็นเลขหมาย IPv6(ranMN) เมื่อ P-Server ได้รับข้อความ Address Acknowledgment ดังกล่าวจะทราบว่า MN ได้ใช้เลขหมาย IPv6(ranMN) แทน เมื่อ timeout ของการรอข้อความ Address Acknowledgment หมดลง P-Server จะลบเลขหมาย IPv6(ranMN) ออกไปและนำเลขหมาย IPv6(NCoA) ที่ยังไม่ได้ใช้งานเก็บไว้ใน pool ตามเดิม ซึ่งถ้าเป็นกรณีที่ไม่มี P-Server ใน Foreign network จะไม่มีขั้นตอนของ P-Server ใน Timing diagram ตามภาพที่ 26 แต่การทำงานของ MN จะเหมือนกัน

กรณีที่ไม่ได้รับเลขหมาย IPv6(NCoA) เนื่องจากการส่งข้อความ Address Reply ผิดพลาดหรือเนื่องจากไม่มี P-Server ในเครือข่ายนั้น การทำงานของ MN ยังคงใช้เวลาไม่เกิน 1000 มิลลิวินาที เพราะในวิธี FR-DAD นี้ออกแบบให้ใช้ข้อความ Address Request เป็นข้อความเดียวกับข้อความ NS ในขั้นตอน DAD แบบมาตรฐาน จึงไม่เพิ่มเวลาการทำงานไม่ว่าจะมีการผิดพลาดดังกล่าว



ภาพที่ 27 Timing diagram กรณีที่ P-Server ไม่ได้รับข้อความ Address Acknowledgment

กรณีที่สามเป็นกรณีที่ P-Server ไม่ได้รับข้อความ Address Acknowledgment ตามภาพที่ 27 เริ่มจาก MN ย้ายเข้ามาใน Foreign network จะส่งข้อความ Address Request และทำขั้นตอน DAD ด้วยเลขหมาย IPv6(ranMN) ไปด้วยคือข้อความ NS ที่มี 'FR' flag MN Link Layer address option และ Target address เป็นเลขหมาย IPv6(ranMN) ซึ่งใน Foreign network นี้มี P-Server ที่เตรียมเลขหมาย IPv6(NCoA) ไว้ใน pool เรียบร้อยแล้ว เมื่อ P-Server ได้รับข้อความ Address Request ที่เป็นการร้องขอเลขหมาย IPv6 จะส่งข้อความ Address Reply ที่ Target address เป็นเลขหมาย IPv6(NCoA) เป็นการจ่ายเลขหมาย IPv6(NCoA) ไปและเริ่มรอการตรวจสอบขั้นตอน DAD เลขหมาย IPv6(ranMN) ไปพร้อมกัน เมื่อ MN ได้รับข้อความข้อความ Address Reply จะนำเลขหมาย IPv6(NCoA) จะนำใช้เป็นเลขหมาย CoA ของ MN และลบเลขหมาย IPv6(ranMN) ออกจากการตรวจสอบและส่งข้อความ Address Acknowledgment ที่ Target address เป็นเลขหมาย IPv6(NCoA) ในขั้นตอนนี้ MN สามารถเริ่มการติดต่อสื่อสารได้เลย แต่เนื่องจาก P-Server ไม่ได้รับข้อความ Address Acknowledgment เพื่อลดความยุ่งยากของการจัดการเลขหมาย IPv6 เมื่อไม่ได้รับข้อความ Address Acknowledgment เมื่อ timeout ของการรอข้อความ Address Acknowledgment หมดลงและ P-Server ยังไม่ได้รับข้อความ Address Acknowledgment P-Server จะลบเลขหมายที่เกี่ยวข้องกับ MN ทั้งหมดคือเลขหมาย IPv6(ranMN) และ IPv6(NCoA) ซึ่ง MN ยังสามารถทำงานได้อย่างปกติ จากเหตุการณ์นี้ทำให้ P-Server เสียเลขหมายใน pool ไปหนึ่งเลขหมาย P-Server จึงมีทำขั้นตอน DAD แบบมาตรฐานเพื่อทดแทนเลขหมายที่หายไปหนึ่งเลขหมาย



ภาพที่ 28 Timing diagram กรณีที่เลขหมาย IPv6(ranMN) มีการใช้งานแล้ว

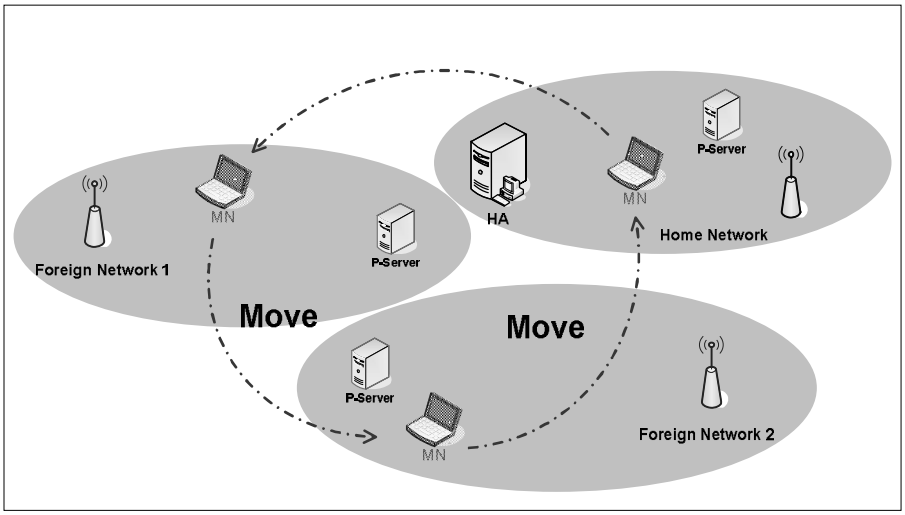
กรณีที่สุดท้ายเป็นกรณีที่เลขหมาย IPv6(ranMN) มีการใช้งานแล้ว ตามภาพที่ 28 เริ่มจาก MN ย้ายเข้ามาใน Foreign network จะส่งข้อความ Address Request และทำขั้นตอน DAD ด้วยเลขหมาย IPv6(ranMN) ไปด้วยคือข้อความ NS ที่มี 'FR' flag MN Link Layer address option และ Target address เป็นเลขหมาย IPv6(ranMN) ซึ่งใน Foreign network นี้มี P-Server ที่เตรียมเลขหมาย IPv6(NCoA) ไว้ใน pool เรียบร้อยแล้ว เมื่อ P-Server ได้รับข้อความ Address Request ที่เป็นการร้องขอเลขหมาย IPv6 จะส่งข้อความ Address Reply ที่ Target address เป็นเลขหมาย IPv6(NCoA) เป็นการจ่ายเลขหมาย IPv6(NCoA) ไปและเริ่มรอการตรวจสอบขั้นตอน DAD เลขหมาย IPv6(ranMN) ไปพร้อมกัน เมื่อ MN ได้รับข้อความข้อความ Address Reply จะนำเลขหมาย IPv6(NCoA) จะนำใช้เป็นเลขหมาย CoA ของ MN และลบเลขหมาย IPv6(ranMN) ออกจากการตรวจสอบและส่งข้อความ Address Acknowledgment ที่ Target address เป็นเลขหมาย IPv6(NCoA) ในขั้นตอนนี้ MN สามารถเริ่มการติดต่อสื่อสารได้เลย ในขณะเดียวกันที่อุปกรณ์อื่นในเครือข่ายเดียวกัน (Neighbor) ก็ได้รับข้อความ NS ที่ Target address เป็นเลขหมาย IPv6(ranMN) จาก MN แต่มี Neighbor อุปกรณ์หนึ่งใช้งานเลขหมายนี้อยู่แล้วจึงส่งข้อความ NA ที่ Target address เป็นเลขหมาย IPv6(ranMN) ให้กับทุกอุปกรณ์ในเครือข่ายเมื่อ P-Server ได้รับข้อความ NA นี้จะลบเลขหมาย IPv6(ranMN) จากการรอการตรวจสอบขั้นตอน DAD และสุ่มเลขหมาย IPv6(ranSer) แล้วทำขั้นตอน DAD ด้วยเลขหมาย IPv6(ranSer) โดยส่งข้อความ NS ที่ Target address เป็นเลขหมาย IPv6(ranSer) แทนการสูญเสียเลขหมาย IPv6(ranMN) ในส่วนของ MN ที่ได้รับข้อความ NA จาก Neighbor อุปกรณ์หนึ่ง MN ได้ใช้เลขหมาย IPv6(NCoA) เป็นเลขหมาย CoA แล้วจึงไม่มีผลกระทบใดๆ ต่อการสื่อสารของ MN

วิธีและขั้นตอนการทดลอง

ในการทดลองการทำงานของ FR-DAD เพื่อเปรียบเทียบประสิทธิภาพการทำงานโดยเปรียบเทียบกับวิธี Standard DAD และ A-DAD ซึ่งมีความใกล้เคียงในลักษณะของขั้นตอน โดยการทดลองนี้จะใช้การจำลองเหตุการณ์ด้วยซอฟต์แวร์ Omnet ++ Simulator version 3.3 ที่มี IPv6 Suite for INET framework (Anonymous, 2006) ทำงานระบบปฏิบัติการ Kubuntu Linux version 7.04

ขั้นตอนการเปิด Simulation หลังจากได้ติดตั้งโปรแกรมเรียบร้อยแล้ว ต้องสร้างโครงสร้างของเครือข่ายของการทดลองตามภาพที่ 29 ด้วย GNED Editor ที่เป็นโปรแกรมย่อยของ Omnet ++ Simulator เป็นไฟล์สกุล ned ที่มีชื่อเดียวกับโฟล์เดอร์ และไฟล์สกุล xml ที่มีชื่อเดียวกับโฟล์เดอร์

เป็นไฟล์ที่ระบุ parameter ย่อยของการทดลองรวมถึงการเคลื่อนที่ตามเวลาของ MN สัมพันธ์กับไฟล์สกุล ned และมีไฟล์ executable ที่มีชื่อเดียวกับโฟล์เดอร์เป็นไฟล์ที่ใช้เปิดโครงงานที่ได้ออกแบบไว้ เมื่อเปิดโปรแกรม Omnet++ Simulator สามารถสั่ง run ตามขั้นตอนที่ได้สร้างไว้ในไฟล์สกุล xml ที่มีชื่อเดียวกับโฟล์เดอร์ได้ นอกจากนี้ยังมีไฟล์ default2.ini ที่อยู่ใน \omnet\IPv6Suite\Etc\ เป็น parameter ของสภาพแวดล้อม และ omnet.ini เป็นไฟล์ที่บอกรายละเอียดเฉพาะภายในงานที่สร้างขึ้น ในรายละเอียดของไฟล์ต่างๆ ที่ได้กล่าวนั้น ได้รวบรวมไว้ในภาคผนวก ข.



ภาพที่ 29 แสดงลักษณะโครงสร้างของเครือข่ายสำหรับการทดลอง

จากภาพที่ 29 เป็นลักษณะโครงสร้างของเครือข่ายสำหรับการทดลอง โดยกำหนดให้มี 3 เครือข่ายที่ MN สามารถเคลื่อนที่ไปได้ซึ่งแต่ละเครือข่ายใช้ subnet ที่แตกต่างกัน 3 subnet การเคลื่อนที่ของ MN จะเริ่มจาก Home network ไปยัง Foreign network 1 และ Foreign network 2 ตามลำดับและกลับมาที่ Home network เป็นการจบขั้นตอนการเคลื่อนย้าย MN

ในการทดลองแบ่งเป็น 3 วิธี 5 กรณี คือ

(1) วิธี DAD แบบมาตรฐาน

(2) วิธี A-DAD แบบไม่มีความผิดพลาด (A-DAD Ideal case) เป็นการทำงานโดยทุกเครือข่ายมีการทำงานของ A-DAD และพร้อมที่จะจ่ายเลขหมาย IPv6 ให้โดยไม่มีการสูญเสียข้อความใดๆ

(3) วิธี A-DAD แบบไม่ได้รับเลขหมาย NCoA (A-DAD missing NCoA case) หรือแบบไม่มีวิธี A-DAD ในเครือข่าย (No A-DAD method case) วิธีนี้ทำโดยการปิดการทำงาน A-DAD ที่ access router ของทุกเครือข่ายจึงไม่มีการตอบกลับของ access router เมื่อมีการร้องขอเลขหมาย

(4) วิธี FR-DAD แบบไม่มีความผิดพลาด (FR-DAD Ideal case) เป็นการทำงานโดยทุกเครือข่ายมีการทำงานของ FR-DAD และพร้อมที่จะจ่ายเลขหมาย IPv6 ให้โดยไม่มีการสูญเสียข้อความใดๆ

(5) วิธี FR-DAD แบบไม่ได้รับเลขหมาย NCoA (FR-DAD missing NCoA case) หรือแบบไม่มี P-Server ในเครือข่าย (No P-Server case) วิธีนี้ทำโดยการปิดการทำงาน FR-DAD ของทุกเครือข่าย MN จึงทำงานโดยใช้ standard DAD

วิธีการทำงานของ A-DAD และ FR-DAD ได้เพิ่มและแก้ไขโปรแกรมด้วยภาษา C++ เข้าไปในส่วน library ตามไฟล์ดังนี้

- NDStatehost.cc , NDStatehost.h
- MIPv6CDSMobileNode.cc
- MIPv6NDStatehost.cc, MIPv6NDStatehost.h
- Netconf2.dtd
- XMLOmnetParser.cc
- RoutingTable6.h
- ICMPv6NDMessage.cc, ICMPv6NDMessage.h
- IPv6Forword.cc
- IPv6Interfacedata.cc, IPv6Interfacedata.h
- Omnet.ini

โดยส่วนของการเพิ่มโคดในไฟล์ที่ได้กล่าวจะอยู่ในภาคผนวก ข. ค่าที่ได้จากการทดลองเป็นรูปแบบของเวลาที่เกิดเหตุการณ์ตามขั้นตอนได้ดึงออกมาจาก log ที่บันทึกไว้มีลักษณะคือ

ตัวอย่าง log ที่บันทึก

1- XServer1 MIPv6 -:Class processRtrAd of MIPv6 at 45.9543

2- XServer1 -:Class processRtrAd of NDState at 45.9543

3- XServer2 MIPv6 -:Class processRtrAd of MIPv6 at 46.4313

4- XServer2 -:Class processRtrAd of NDState at 46.4313

5- 46.5869 MosNetwork.client1.linkLayers[0].networkInterface associated to: 55:d:86:ba:f:78 on chan: 1 sig strength: -89.9733

6- client1 MIPv6 -:Class processRtrAd of MIPv6 at 46.6075

7- client1 -:Class processRtrAd of NDState at 46.6075

8- client1 -:Class prefixAddrConf at 46.6075

9- client1 -:Class prefixAddrConf IPv6= 3019:ffff:0:0:ecee:b1ff:fe42:77d5/64

10- client1 INet 0 = fe80:0:0:0:ecee:b1ff:fe42:77d5/64

11- client1 INet 1 = 3018:ffff:0:0:ecee:b1ff:fe42:77d5/64

12- client1 -:Class detectDupAddress at 46.6075

13- client1 -:Class dupAddrDetection IPv6 = 3019:ffff:0:0:ecee:b1ff:fe42:77d5/64 at 46.6075

14- client1 MIPv6 -:Class processRtrAd of MIPv6 at 46.6086

15- client1 -:Class processRtrAd of NDState at 46.6086

16- client1 MIPv6 -:Class processRtrAd of MIPv6 at 46.6098

17- client1 -:Class processRtrAd of NDState at 46.6098

18- PCconnect -:Class processRtrAd of NDState at 46.7816

19- XServer3 MIPv6 -:Class processRtrAd of MIPv6 at 46.8651

20- XServer3 -:Class processRtrAd of NDState at 46.8651

21- XServer2 MIPv6 -:Class processRtrAd of MIPv6 at 46.9777

22- XServer2 -:Class processRtrAd of NDState at 46.9777

23- client1 MIPv6 -:Class processRtrAd of MIPv6 at 46.9777

24- client1 -:Class processRtrAd of NDState at 46.9777

25- XServer1 MIPv6 -:Class processRtrAd of MIPv6 at 47.1212

26- XServer1 -:Class processRtrAd of NDState at 47.1212

27- client1 -:Class dupAddrDetection IPv6 = 3019:ffff:0:0:ecee:b1ff:fe42:77d5/64 at 47.6075

```
28- client1 -:Class dupAddrDetSuccess IPv6 Success = 3019:ffff:0:0:ecee:b1ff:fe42:77d5/64
    at 47.6075
29- client1 -:Class dupAddrDetSuccess potentialCoa = 3019:ffff:0:0:ecee:b1ff:fe42:77d5/64
    Scope: Global at 47.6075
30- client1 -:Class SendBU coa!=hoa at 47.6075
31- client1 -:Class SendBU at 47.6075
32- PCconnect -:Class processRtrAd of NDState at 48.0005
33- XServer3  MIPv6 -:Class processRtrAd of MIPv6 at 48.2458
34- XServer3 -:Class processRtrAd of NDState at 48.2458
35- XServer2  MIPv6 -:Class processRtrAd of MIPv6 at 48.2983
36- XServer2 -:Class processRtrAd of NDState at 48.2983
37- client1  MIPv6 -:Class processRtrAd of MIPv6 at 48.2983
38- client1 -:Class processRtrAd of NDState at 48.2983
39- client1 -:Class Handover at 48.2983
40- -:Class formCareOfAddress CoA= 3019:ffff:0:0:ecee:b1ff:fe42:77d5 At 48.2983 sec,
    client1
41- XServer1  MIPv6 -:Class processRtrAd of MIPv6 at 48.4873
42- XServer1 -:Class processRtrAd of NDState at 48.4873
43- PCconnect -:Class processRtrAd of NDState at 49.106
44- haa -:Class process NS at 49.1278
45- router2 -:Class process NS at 49.158
46- client1 -:Class process NS at 49.1888
47- client1 -:Class processBA Binding Ack at 49.2093
```

ในการทดลองได้กำหนดพารามิเตอร์ในการทดลองตามตารางที่ 1 มีรายละเอียดดังนี้ RetransTimer เป็นเวลาที่ใช้ในการทดสอบ DAD มีค่ามาตรฐานที่ 1000 มิลลิวินาที DupAddrDetectTransmits เป็นจำนวนครั้งที่ใช้ทดสอบ DAD ต่อหนึ่งเลขหมาย Router Advertisement interval เป็นค่าเวลาของรอบที่ access router ส่งข้อความ RA ที่มี prefix information ของเครือข่ายที่ access router ดูแลอยู่โดยกำหนดให้เป็นค่าสุ่มอยู่ในช่วง 1 วินาทีถึง 1.5 วินาที P-Server's Acknowledgement Timer เป็นเวลาในการรอข้อความ Address Ackledgement ของ P-

Server ในวิธี FR-DAD กำหนดที่ 1.5 วินาที และสุดท้ายเป็น A-DAD's NCoA Timeout เป็นเวลาในการรอรับข้อความ reply ที่มี NCoA ของ MN ตามวิธี A-DAD โดยกำหนดที่เวลา 300 มิลลิวินาที

ตารางที่ 1 แสดงพารามิเตอร์ที่ใช้ในการทดลอง

ชื่อพารามิเตอร์	ค่าที่ใช้ในการทดลอง
RetransTimer	1000 ms
DupAddrDetectTransmits	1
Router Advertisement interval	(1 s, 1.5 s)
P-Server's Acknowledgement Timer	1500 ms
A-DAD's NCoA Timeout	300 ms

ซึ่งบันทึกเฉพาะขั้นตอนสำคัญ 4 ขั้นตอนคือ

(1) เวลาที่สัญญาณของ Access point ติดต่อกับ MN เรียบร้อยแล้ว (Signal up) ตาม log บรรทัดที่ 5

(2) เวลาที่ MN ได้รับข้อความ RA ที่มี Prefix information จาก access router และเริ่มทดสอบ DAD (Received RA message and test DAD) ตาม log บรรทัดที่ 6

(3) เวลาที่วิธี DAD สำเร็จหรือ MN สามารถใช้เลขหมาย CoA ในเครือข่ายใหม่ได้ และส่งข้อความ BU (DAD Success) ตาม log บรรทัดที่ 28

(4) เวลาที่ MN ได้รับข้อความ BU Acknowledgment (Received BU Acknowledgment message) ตาม log บรรทัดที่ 47

จากค่าที่บันทึกได้จะนำมาใช้วัดประสิทธิภาพมีด้วยกัน 2 ค่าคือ เวลาที่ใช้ในขั้นตอน DAD (DAD delay time) เป็นเวลาที่ใช้จากขั้นตอนที่ MN ได้รับข้อความ RA ที่มี Prefix information จาก access router และเริ่มทดสอบ DAD ถึงขั้นตอนที่วิธี DAD สำเร็จ (ขั้นตอนที่ (3) – ขั้นตอนที่ (2)) และเวลาที่ใช้ในขั้นตอน Handover layer-3 (Handover layer-3 delay time) เป็นเวลาที่ใช้จากขั้นตอนที่สัญญาณของ Access point ติดต่อกับ MN เรียบร้อยแล้วถึงขั้นตอนที่ MN ได้รับข้อความ BU Acknowledgment (ขั้นตอนที่ (4) – ขั้นตอนที่ (1)) นอกจากนี้จะใช้การคำนวณจำนวนข้อความที่ใช้ในแต่ละกรณี เพื่อเปรียบเทียบมิติของปริมาณข้อความที่ใช้สื่อสารภายในเครือข่ายอีกด้วย

ผลและวิจารณ์

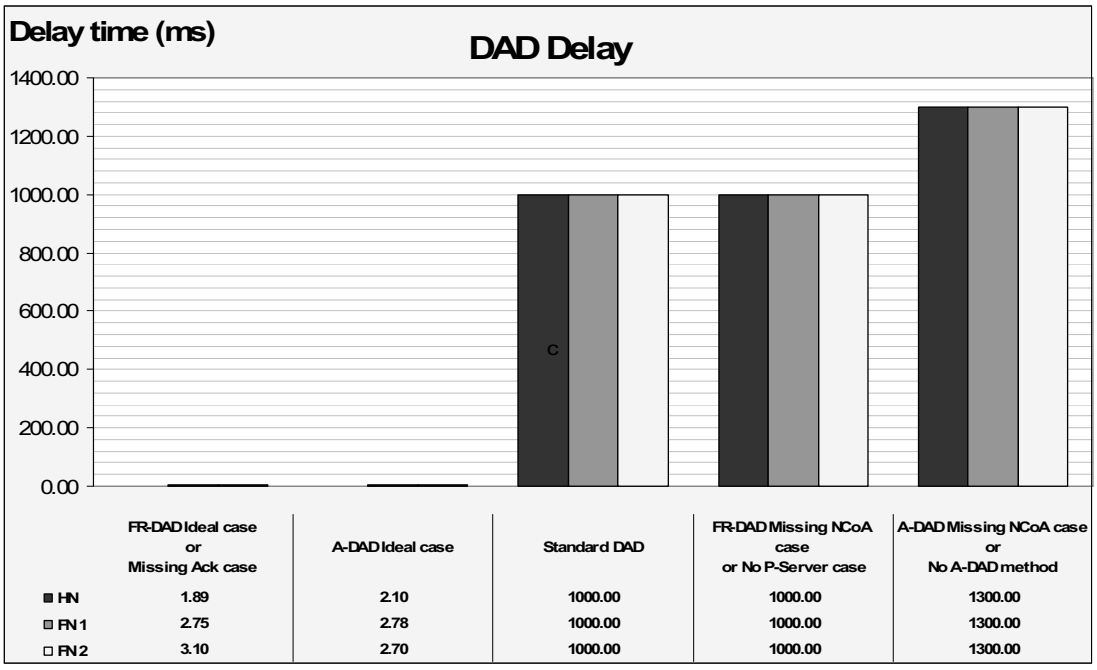
ผล

จากวิธีการทดลองที่ได้กล่าวไปในหัวข้อก่อนหน้านี้โดยใช้ซอฟต์แวร์ Omnet ++ Simulator จำลองเหตุการณ์ที่เพิ่มขั้นตอนและข้อความของโพรโตคอล FR-DAD กับโพรโตคอล A-DAD และทำการทดลองตามโครงสร้างของเครือข่ายที่ออกแบบไว้ในภาพที่ 29

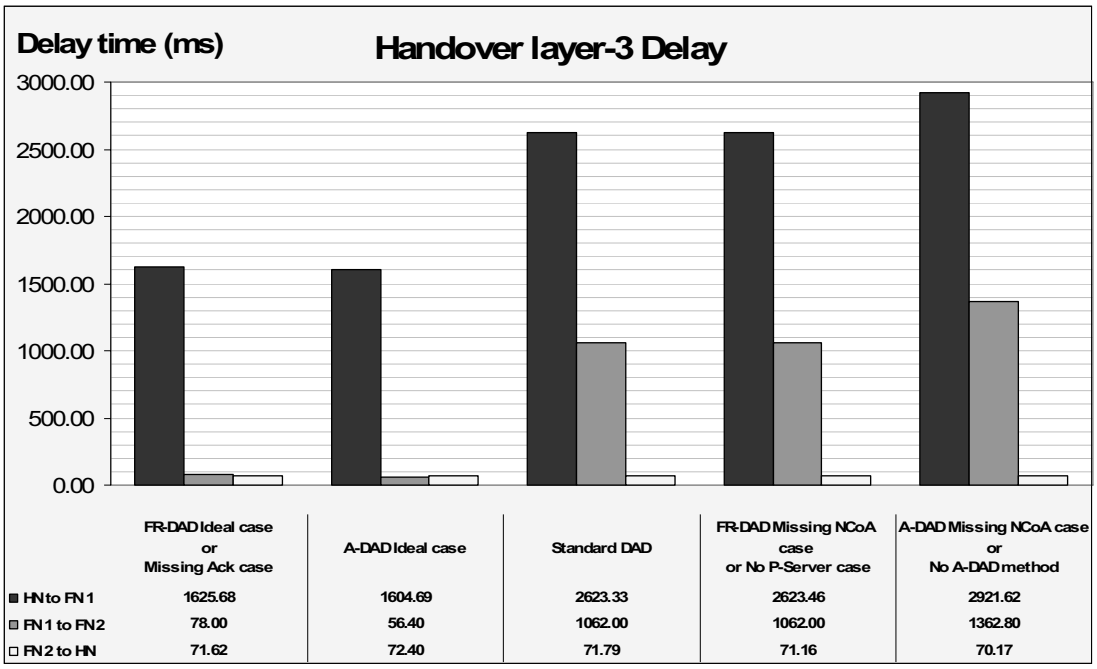
ในการทดลองจะเก็บเวลาของเหตุการณ์ที่ใช้คำนวณหา DAD delay time และ Handover layer-3 delay time ซึ่งจะทดลองทั้งหมด 10 ครั้งใน 1 กรณีและหาค่าเฉลี่ยของเวลาที่เกิดขึ้นก่อนคำนวณ delay time ได้ค่าตามตารางที่ 2 แสดงข้อมูลตาม log ที่ได้บันทึกไว้เป็นเวลาที่เกินเหตุการณ์ขึ้นในหน่วยวินาที และค่าเฉลี่ยที่ได้จากตารางที่ 2 จะนำไปคำนวณเป็น DAD delay time และ Handover layer-3 delay time ดังภาพที่ 30 และ 31

ตารางที่ 2 แสดงผลการทดลองค่าเฉลี่ยของเวลาตามสถานการณ์ในหน่วยวินาทีใน 5 กรณี

ขั้นตอน	Standard DAD	FR-DAD Ideal case	FR-DAD Missing NCoA case	A-DAD Ideal case	A-DAD missing NcoA case
Home Network:					
Signal up	0.605892 s	0.602862 s	0.643859 s	0.604862 s	0.603859 s
Receive RA + prefix and test DAD	1.024975 s	1.017524 s	1.016798 s	1.320732 s	1.319017 s
Foreign network 1:					
Signal up	46.617810 s	46.630600 s	46.582580 s	46.589200 s	46.585760 s
Receive RA + prefix and test DAD	46.638870 s	46.651580 s	46.603750 s	46.604190 s	46.605150 s
DAD Success	47.638870 s	46.654330 s	47.603750 s	46.606970 s	47.905150 s
Receive BU Ack	49.241140 s	48.256280 s	49.206040 s	48.208880 s	49.507380 s
Foreign network 2:					
Signal up	142.957900 s	142.943900 s	142.944400 s	142.931800 s	142.926800 s
Receive RA + prefix and test DAD	142.979100 s	142.964900 s	142.964500 s	142.953000 s	142.948000 s
DAD Success	143.979100 s	142.968000 s	143.964500 s	142.955700 s	144.248000 s
Receive BU Ack	144.019900 s	143.021900 s	144.006400 s	143.009400 s	144.289600 s
Return to Home Network:					
Signal up	239.794300 s	239.767000 s	239.770300 s	239.741100 s	239.763500 s
Receive RA + prefix and send BU	239.815200 s	239.787900 s	239.790900 s	239.762100 s	239.782700 s
Receive BU Ack	239.866094 s	239.838616 s	239.841464 s	239.813499 s	239.833669 s



ภาพที่ 30 แสดง DAD delay time เฉลี่ยของ 5 กรณี



ภาพที่ 31 แสดง Handover layer-3 delay time เฉลี่ยของ 5 กรณี

วิจารณ์

ลักษณะของเวลาที่ใช้ในขั้นตอน DAD มี delay time เกิดขึ้นดังภาพที่ 30 นั้น เห็นได้ว่า FR-DAD Ideal case และ A-DAD Ideal case ใช้เวลาประมาณ 2 – 3 มิลลิวินาที ซึ่งเมื่อเทียบกับการทำงานของ Standard DAD ที่ใช้เวลารอการตรวจสอบที่ 1000 มิลลิวินาที สามารถลดเวลาการทดสอบไปได้มาก ในกรณี FR-DAD Missing NCoA หรือ No P-Server จะใช้เวลาการตรวจสอบเท่ากับการทำงานปกติของ Standard DAD เพราะ MN ไม่ได้รับข้อความ Address Reply ที่มีเลขหมาย IPv6(NCoA) เมื่อครบการรอที่ 1000 มิลลิวินาที MN จะใช้เลขหมาย IPv6(ranMN) เป็นเลขหมาย CoA ที่ได้ทำงานตรวจสอบแทนทันทีตามขั้นตอนที่ได้กล่าวไว้ใน Timing diagram กรณีที่ไม่ได้รับเลขหมาย IPv6(NCoA) ตามภาพที่ 26

ในกรณี A-DAD Missing NCoA หรือ No A-DAD method ของวิธี A-DAD ใช้ DAD Delay time 1300 มิลลิวินาที เกิดขึ้นจากเมื่อ MN ได้ส่งข้อความ RS ที่มี Duplication-free NCoA Request Option เป็นการร้องขอเลขหมาย NCoA แต่เนื่องจากเป็นกรณี A-DAD Missing NCoA หรือ No A-DAD method ทำให้ MN ไม่ได้รับ NCoA ซึ่งจะใช้เวลารอ 300 มิลลิวินาที (กำหนดให้ A-DAD's NCoA Timeout เป็น 300 มิลลิวินาทีตามตารางที่ 1) จึงทำการทดสอบด้วยวิธี Standard DAD ที่ใช้เวลา 1000 มิลลิวินาที ถึงได้เลขหมาย CoA มาใช้ในการสื่อสารต่อไป

เวลาที่ใช้ในขั้นตอน Handover layer-3 มี delay time เกิดขึ้นดังภาพที่ 30 เป็น delay time ของ3 ขั้นตอนคือ ขั้นตอนการรอ prefix information, ขั้นตอน DAD และขั้นตอนการทำ Binding updateลักษณะของเวลาที่เกิดขึ้นจะ เป็นการใช้งานของขั้นตอนมาตรฐานยกเว้นการทำงานของ DAD ดังนั้นค่าที่เกิดขึ้นจะมีแนวโน้มของการใช้เวลาแปรตามวิธีขั้นตอน DAD ที่ต่างออกไป แต่จะเห็นว่าเมื่อ MN เคลื่อนย้ายจาก home network ไป foreign network 1 จะใช้ Handover layer-3 delay time มากกว่าจากที่อื่นเพราะเสียเวลาไปกับการทำงานขั้นตอน binding update หรือการลงทะเบียนกับ home agent ในครั้งแรก ซึ่งเป็นครั้งแรกที่ home agent ต้องตรวจสอบ MN และสร้าง binding cache สำหรับ MN ที่จำคู่ระหว่าง home address กับ care-of address ตามผลการทดลองตาม
ตารางที่ 2

นอกจากนี้ในกรณี MN เคลื่อนย้ายกลับบ้านหรือจาก foreign network 2 ไป home network จะไม่ต้องทดสอบ DAD ไม่ว่ากรณีของวิธีใดก็ตาม MN จะกลับมาใช้ home address โดยแจ้งให้ home agent รู้ว่าได้กลับมาใช้ home address แล้ว

เห็นได้ว่าวิธี FR-DAD ใช้เวลาที่มากที่สุดสำหรับกรณีที่มีความผิดพลาดมากที่สุดเท่ากับการทำงานของ Standard DAD แต่วิธี A-DAD เมื่อเกิดกรณีที่มีความผิดพลาดมากที่สุดจะใช้เวลามากกว่าการทำงานของ Standard DAD 300 มิลลิวินาทีหรือตามค่า A-DAD's NCoA Timeout

นอกจากการเปรียบเทียบด้านความเร็วในสภาวะที่ดีที่สุด และร้ายที่สุดในเรื่องของเวลานั้น ยังสามารถวิเคราะห์ได้จาก จำนวนของข้อความที่ใช้ในกรณีต่างๆ เพื่อเพิ่มมุมมองในการเปรียบเทียบแต่ละวิธี ในการวิเคราะห์จำนวนของข้อความนี้จะคำนวนออกมาเป็นสมการซึ่งมีตัวแปรเป็นชนิดของข้อความต่างๆ โดยกำหนดให้

M_x เป็นจำนวนของข้อความชนิด x

d_1 เป็นจำนวนครั้งของเลขหมาย IPv6 ที่ใช้ในการทดสอบแล้วเกิดการซ้ำกันโดยเป็นเลขหมายที่สร้างโดย MN

d_2 เป็นจำนวนครั้งของเลขหมาย IPv6 ที่ใช้ในการทดสอบแล้วเกิดการซ้ำกันโดยเป็นเลขหมายที่สร้างโดย P-Server ของวิธี FR-DAD หรือ access router ของวิธี A-DAD

N_y เป็นจำนวนข้อความที่ใช้สำหรับกรณีและวิธี y

เริ่มจากกรณีใช้วิธี DAD แบบมาตรฐาน จำนวนของข้อความในวิธี DAD แบบมาตรฐาน (N_{StdDAD}) ซึ่งกรณีที่ทำให้จำนวนข้อความมากขึ้นนั้นขึ้นอยู่กับจำนวนครั้งของการซ้ำของเลขหมาย IPv6 ที่ใช้ในการทดสอบ ทำให้มีจำนวนของข้อความเพิ่มขึ้นดังสมการ (1)

$$N_{StdDAD} = M_{NS\text{-}DAD} \cdot DupAddrDetectTransmits + (M_{NA} + M_{NS\text{-}DAD})d_1$$
$$N_{StdDAD} = 2d_1 + DupAddrDetectTransmits \quad (1)$$

ถัดมาเป็นกรณีของวิธี A-DAD แบบไม่มีความผิดพลาด (A-DAD Ideal case) จำนวนของข้อความที่ใช้ในกรณีนี้ ($N_{A\text{-}DAD\text{-}1}$) เกิดจากข้อความ Request (ข้อความ RS ที่มี Duplication-free NCoA Request Option) และเมื่อ access router ได้รับข้อความ Request แล้วจะส่งข้อความ Reply

(ข้อความ RA ที่มี Duplication-free NCoA Reply Option) ที่มีเลขหมาย NCoA กลับมาเป็นการจบกระบวนการด้านการจ่ายเลขหมายของ access router แต่เมื่อ access router เสีย NCoA ไปเลขหมายหนึ่งจะต้องทำการสุ่มทดสอบเลขหมาย NCoA เก็บไว้ใน pool เท่ากับจำนวนที่ต้องการ ดังนั้นจึงมีการทำขั้นตอน DAD แบบมาตรฐานขึ้นสำหรับเลขหมายที่สูญเสียไปจึงเกิดเป็นผลกระทบของจำนวนข้อความที่ใช้โดนเพิ่มขึ้นจากจำนวนครั้งของเลขหมาย IPv6 ที่ใช้ในการทดสอบแล้วเกิดการซ้ำกันโดยเป็นเลขหมายที่สร้างโดย access router ของวิธี A-DAD ตามสมการ (2)

$$
\begin{aligned}
N_{\text{A-DAD-1}} &= M_{\text{RS(req)}} + M_{\text{RA(reply)}} + N_{\text{StdDAD-AR}} \\
&= 1 + 1 + 2d_2 + \text{DupAddrDetectTransmits} \\
N_{\text{A-DAD-1}} &= 2d_2 + \text{DupAddrDetectTransmits} + 2 \qquad (2)
\end{aligned}
$$

ถัดมาเป็นกรณีวิธี A-DAD แบบไม่ได้รับเลขหมาย NCoA (A-DAD missing NCoA case) จำนวนของข้อความที่ใช้ในกรณีนี้ ($N_{\text{A-DAD-2}}$) เกิดจากการส่งข้อความ Reply (ข้อความ RA ที่มี Duplication-free NCoA Reply Option) จาก access router ไม่ถึง MN และเมื่อหมดเวลาในการรอรับเลขหมาย NCoA MN จะต้องทำขั้นตอน DAD แบบมาตรฐานซึ่งผลกระทบของจำนวนข้อความที่ใช้โดนเพิ่มขึ้นจากจำนวนครั้งของเลขหมาย IPv6 ที่ใช้ในการทดสอบแล้วเกิดการซ้ำกันโดยเป็นเลขหมายที่สร้างโดย MN และ access router ได้สูญเสีย NCoA ไปเลขหมายหนึ่งจึงต้องทำการสุ่มทดสอบเลขหมาย NCoA เก็บไว้ใน pool เท่ากับจำนวนที่ต้องการซึ่งจะเกิดผลกระทบของจำนวนข้อความที่ใช้โดนเพิ่มขึ้นจากจำนวนครั้งของเลขหมาย IPv6 ที่ใช้ในการทดสอบแล้วเกิดการซ้ำกันโดยเป็นเลขหมายที่สร้างโดย access router ของวิธี A-DAD ดังสมการ (3)

$$
\begin{aligned}
N_{\text{A-DAD-2}} &= M_{\text{RS(req)}} + M_{\text{RA(reply-lost)}} + N_{\text{StdDAD-MN}} + N_{\text{StdDAD-AR}} \\
&= 1 + 1 + (2d_1 + \text{DupAddrDetectTransmits}) \\
&\quad + (2d_2 + \text{DupAddrDetectTransmits}) \\
N_{\text{A-DAD-2}} &= 2d_1 + 2d_2 + 2\text{DupAddrDetectTransmits} + 2 \qquad (3)
\end{aligned}
$$

กรณีวิธี A-DAD แบบไม่มีวิธี A-DAD ในเครือข่าย (No A-DAD method case) จำนวนของข้อความที่ใช้ในกรณีนี้ ($N_{\text{A-DAD-3}}$) มีขั้นตอนจากที่ MN ส่งข้อความ Request (ข้อความ RS ที่มี Duplication-free NCoA Request Option) ไปแล้วไม่มีการตอบกลับเพราะไม่มีวิธี A-DAD อยู่ในเครือข่ายนี้ เมื่อหมดเวลารอจะเป็นการทำงานการทำขั้นตอน DAD แบบมาตรฐานขึ้นสำหรับเลข

หมายที่สูญเสียไปจึงเกิดเป็นผลกระทบของจำนวนครั้งของเลขหมาย IPv6 ที่ใช้ในการทดสอบแล้วเกิดการซ้ำกันโดยเป็นเลขหมายที่สร้างโดย MN ดังสมการ (4)

$$N_{A\text{-}DAD\text{-}3} = M_{RS(req)} + N_{StdDAD\text{-}MN}$$
$$N_{A\text{-}DAD\text{-}3} = 2d_1 + DupAddrDetectTransmits + 1 \quad (4)$$

ถัดมาเป็นกรณีในส่วนของวิธี FR-DAD เริ่มจากกรณีวิธี FR-DAD แบบไม่มีความผิดพลาด (FR-DAD Ideal case) จำนวนของข้อความที่ใช้ในกรณีนี้ ($N_{FR\text{-}DAD\text{-}1}$) เป็นขั้นตอนการทำงานของสามข้อความที่ใช้ในการร้องขอเลขหมาย NCoA คือข้อความ Address Request, Address Reply และ Address Acknowledgment โดยไม่มีความผิดพลาด MN สามารถทำงานได้เลย และหลังจากนั้นในส่วนของ P-Server มีความเสี่ยงกรณีที่เลขหมาย IPv6(ranMN) มีโอกาสซ้ำซึ่งทำให้ P-Server ต้องสุ่มเลขหมายขึ้นมาใหม่เพื่อทดสอบอีก (IPv6(ranSer)) เป็นผลกระทบของจำนวนครั้งของเลขหมาย IPv6 ที่ใช้ในการทดสอบแล้วเกิดการซ้ำกันโดยเป็นเลขหมายที่สร้างโดย P-Server ของวิธี FR-DAD ดังสมการ (5)

$$N_{FR\text{-}DAD\text{-}1} = M_{NS(req)} + M_{NA(reply)} + M_{NA(ack)} + (N_{StdDAD\text{-}Pserver} - M_{NS(req)})$$
$$N_{FR\text{-}DAD\text{-}1} = 2d_2 + DupAddrDetectTransmits + 2 \quad (5)$$

ถัดมาเป็นกรณีวิธี FR-DAD แบบไม่ได้รับข้อความ Address Acknowledgment (FR-DAD Address Acknowledgment missing case) จำนวนของข้อความที่ใช้ในกรณีนี้ ($N_{FR\text{-}DAD\text{-}2}$) ต่างจากกรณีวิธี FR-DAD แบบไม่มีความผิดพลาดที่เมื่อ P-Server ไม่ได้รับข้อความ จะลบเลขหมาย IPv6(NCoA) และ IPv6(ranMN) ทิ้ง ทำให้เลขหมาย IPv6 ใน pool ลดลงหนึ่งเลขหมาย P-Server จึงต้องทำขั้นตอน DAD แบบมาตรฐานขึ้นสำหรับเลขหมายที่สูญเสียไปจึงเกิดเป็นผลกระทบของจำนวนครั้งของเลขหมาย IPv6 ที่ใช้ในการทดสอบแล้วเกิดการซ้ำกันโดยเป็นเลขหมายที่สร้างโดย P-Server ของวิธี FR-DAD ดังสมการ (6)

$$N_{FR\text{-}DAD\text{-}2} = M_{NS(req)} + M_{NA(reply)} + M_{NA(ack\text{-}lost)} + N_{StdDAD\text{-}Pserver}$$
$$N_{FR\text{-}DAD\text{-}2} = 2d_2 + DupAddrDetectTransmits + 3 \quad (6)$$

ถัดมาเป็นกรณีวิธี FR-DAD แบบไม่ได้รับเลขหมาย NCoA (FR-DAD missing NCoA case) เกิดขึ้นได้จากการที่ข้อความ Address Reply ส่งไปไม่ถึง MN และเกิดการซ้ำของเลขหมาย IPv6(ranMN) ไปพร้อมๆ กัน ทำให้ MN ต้องสุ่มเลขหมาย IPv6(ranMN) ขึ้นมาใหม่และทดสอบด้วยข้อความ Address Request ซึ่งเป็นการร้องขอเลขหมายกับ P-Server ไปพร้อมๆกัน กรณีที่ทำให้ MN ได้รับเลขหมาย CoA มานั้นจะเกิดได้ 2 กรณี คือกรณีแรก MN จะได้รับข้อความ Address Reply ในครั้งสุดท้ายทำให้ได้ NCoA จาก P-Server มาใช้เป็น CoA ของ MN และP-Server มีความเสี่ยงกรณีที่เลขหมาย IPv6(ranMN) มีโอกาสซ้ำจึงเป็นผลกระทบแบบ d_2 อีก จำนวนของข้อความที่ใช้ในกรณีนี้ ($N_{FR\text{-}DAD\text{-}3}$) เป็นไปดังสมการ (7)

$$\begin{aligned} N_{FR\text{-}DAD\text{-}3} &= (M_{NS(req)} + M_{NA(reply\text{-}lost)} + M_{NA})d_1 + M_{NS(req)} + M_{NA(reply)} + M_{NA(ack)} \\ &\quad + (N_{StdDAD\text{-}Pserver} - M_{NS(req)}) \\ N_{FR\text{-}DAD\text{-}3} &= 3d_1 + 2d_2 + DupAddrDetectTransmits + 2 \end{aligned} \tag{7}$$

กรณีที่สองเกิดจากการไม่ซ้ำกันของเลขหมาย IPv6(ranMN) และ MN ไม่ได้รับข้อความ Address Reply ในครั้งสุดท้าย ทำให้ MN ใช้เลขหมาย IPv6(ranMN) เป็น CoA และ P-Server ได้เก็บ NCoA กลับไปใน pool อย่างเดิม จำนวนของข้อความที่ใช้ในกรณีนี้ ($N_{FR\text{-}DAD\text{-}4}$) เป็นไปดังสมการ (8)

$$\begin{aligned} N_{FR\text{-}DAD\text{-}4} &= (M_{NS(req)} + M_{NA(reply\text{-}lost)} + M_{NA})d_1 \\ &\quad + (M_{NS(req)} + M_{NA(reply\text{-}lost)})\, DupAddrDetectTransmits + M_{NA(ack)} \\ N_{FR\text{-}DAD\text{-}4} &= 3d_1 + 2DupAddrDetectTransmits + 1 \end{aligned} \tag{8}$$

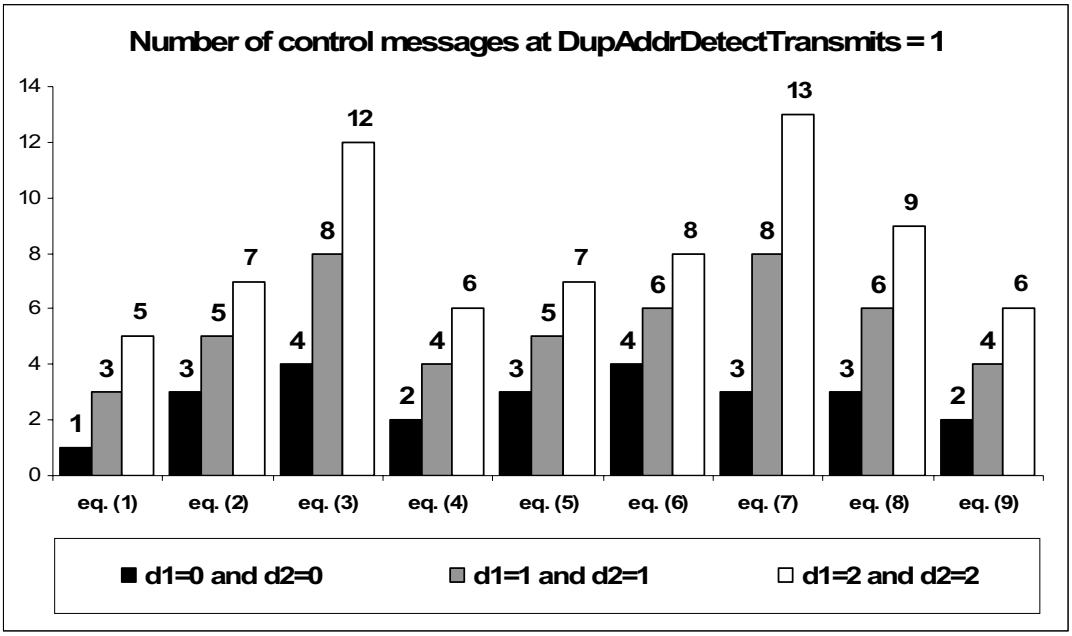
กรณีสุดท้ายเป็นกรณีวิธี FR-DAD แบบไม่มี P-Server ในเครือข่าย (No P-Server case) จำนวนของข้อความที่ใช้ในกรณีนี้ ($N_{FR\text{-}DAD\text{-}5}$) เป็นเหมือนการทำขั้นตอน DAD แบบมาตรฐานและมีการส่งข้อความ Address Acknowledgment เป็นไปดังสมการ (9)

$$\begin{aligned} N_{FR\text{-}DAD\text{-}5} &= N_{StdDAD\text{-}MN} + M_{NA(ack\text{-}lost)} \\ N_{FR\text{-}DAD\text{-}5} &= 2d_1 + DupAddrDetectTransmits + 1 \end{aligned} \tag{9}$$

จากสมการทั้งหมดสามารถสรุปได้เป็นตารางที่ 3 ดังนี้

ตารางที่ 3 สรุปสมการการคำนวณการใช้ข้อความทั้ง 9 กรณี

สมการที่	รายละเอียด	สมการ
1	Standard DAD	$N_{StdDAD} = 2d_1 + DupAddrDetectTransmits$
2	A-DAD: Ideal case	$N_{A\text{-}DAD\text{-}1} = 2d_2 + DupAddrDetectTransmits + 2$
3	A-DAD: Missing NCoA case	$N_{A\text{-}DAD\text{-}2} = 2d_1 + 2d_2 + 2DupAddrDetectTransmits + 2$
4	A-DAD: No A-DAD Method in Network case	$N_{A\text{-}DAD\text{-}3} = 2d_1 + DupAddrDetectTransmits + 1$
5	FR-DAD: Ideal case	$N_{FR\text{-}DAD\text{-}1} = 2d_2 + DupAddrDetectTransmits + 2$
6	FR-DAD: Missing Address Acknowledgment message case	$N_{FR\text{-}DAD\text{-}2} = 2d_2 + DupAddrDetectTransmits+ 3$
7	FR-DAD: Missing NCoA case, MN จะได้รับข้อความ Address Reply ในครั้งสุดท้าย	$N_{FR\text{-}DAD\text{-}3} = 3d_1 + 2d_2 + DupAddrDetectTransmits + 2$
8	FR-DAD: Missing NCoA case, MN ไม่ได้รับข้อความAddress Reply ในครั้งสุดท้าย	$N_{FR\text{-}DAD\text{-}4} = 3d_1 + 2DupAddrDetectTransmits + 1$
9	FR-DAD: No P-Server in Network case	$N_{FR\text{-}DAD\text{-}5} = 2d_1 + DupAddrDetectTransmits + 1$



ภาพที่ 32 แสดงกราฟแท่งเปรียบเทียบจำนวนของข้อความที่กำหนดค่า d_1 และ d_2 ที่ 3 ค่า

จากสมการทั้งหมดได้สร้างกราฟแท่งขึ้นมาเปรียบเทียบดังภาพที่ 32 โดยกำหนดให้ค่าของ d_1 และ d_2 มีค่าเท่ากันในสามกรณีคือที่ $d_1=d_2=0$, $d_1=d_2=1$ และ $d_1=d_2=2$ จะเห็นได้ว่าสมการ (1) คือ วิธี DAD แบบมาตรฐานใช้จำนวนของข้อความน้อยที่สุด สมการ (2) และ (5) ใช้จำนวนของข้อความเท่ากันซึ่งเป็นกรณีไม่มีความผิดพลาดของวิธี A-DAD และ FR-DAD ตามลำดับ เมื่อเปรียบเทียบกรณีที่มีความผิดพลาดของวิธี A-DAD และ FR-DAD จะเห็นว่าวิธี FR-DAD มีการใช้ข้อความมากกว่าวิธี A-DAD แต่เมื่อมองในด้านความเร็ววิธี FR-DAD จะช่วยให้การตรวจสอบเลขหมายเป็นไปได้เร็วขึ้น

เมื่อเปรียบเทียบการทำงานของ DHCPv6 กับ FR-DAD ในด้านการใช้ข้อความในกรณีที่ไม่มีความผิดพลาดของการสื่อสาร FR-DAD จะใช้ 3 ข้อความ DHCPv6 ใช้ 4 ข้อความตามภาพที่ 17 แต่ DHCPv6 ไม่มีความเสี่ยงกับการซ้ำกันของเลขหมายเพราะมี DHCP Server จัดการเลขหมายให้ สิ่งที่ทำให้ DHCPv6 เสียเวลาในการได้รับเลขหมายคือ กรณีที่ข้อความไปไม่ถึงผู้รับ ทำให้มีการร้องขอจนกว่าจะได้รับข้อความ ซึ่ง FR-DAD ใช้การแก้ปัญหาโดยมีทางเลือกให้กับ MN เพื่อให้ MN ไม่ต้องรอความผิดพลาดต่างๆ ให้ได้เลขหมาย CoA ช้าเกินไป

สรุปและข้อเสนอแนะ

สรุป

วัตถุประสงค์ของงานวิจัยนี้คือ การพัฒนาขั้นตอนการตรวจสอบการซ้ำกันของเลขหมาย IPv6 ให้ใช้เวลาน้อย เพื่อรองรับการทำงานของเทคโนโลยีที่ต้องการความต่อเนื่องของการส่ง-รับข้อมูลพร้อมกับการเคลื่อนย้ายไปด้วยกันตามจุดมุ่งหมายของเครือข่ายแบบเคลื่อนที่ที่ใช้เลขหมาย IPV6 (Mobile IPv6 network) จากการออกแบบและพัฒนาขั้นตอนวิธีการสามารถช่วยลดการทำงานขั้นตอน Standard DAD ได้มาก วิธี FR-DAD นี้ใช้การดัดแปลงข้อความ ICMPv6 มาตรฐานใน RFC4861 ทำให้ทำงานกับขั้นตอนมาตรฐานได้ ซึ่งง่ายต่อการเพิ่มวิธีนี้เข้าไปกับระบบเดิมให้กับ MN อุปกรณ์ที่จะเป็นที่ต้องมีทุกเครือข่ายสำหรับการใช้วิธี FR-DAD คือ P-Server ซึ่งช่วยสำรองเลขหมาย IPv6 ที่ผ่านการตรวจสอบการซ้ำกันเรียบร้อยแล้ว

จากผลการทดลองสามารถลดเวลาการตรวจสอบไปได้มากแต่เมื่อเปรียบเทียบกับการทำงานของวิธี A-DAD ที่ได้มีการพัฒนามาแล้วนั้น ให้ผลใกล้เคียงกันในด้านการลดเวลาการตรวจสอบสำหรับกรณีที่ไม่มีความผิดพลาดใช้เวลาประมาณ 1.89 ms นอกจากนี้ ในการพัฒนา FR-DAD ยังคำนึงถึงการออกแบบโพรโตคอลให้เข้ากันได้กับการทำงานของข้อความที่ใช้เป็นมาตรฐาน ซึ่งสามารถนำ FR-DAD เข้าไปใช้ในเครือข่ายมาตรฐานได้เลยโดยวิธีมาตรฐานสามารถทำงานได้ปกติ และการออกแบบการทำงานนี้ยังให้ความสำคัญในการรักษาความคงทนของการใช้เวลาในการตรวจสอบโดยสามารถใช้เวลาเท่ากับ Standard DAD ที่ 1000 ms ในกรณีที่มีความผิดพลาด ซึ่งได้พัฒนาเวลาให้ดีขึ้นเมื่อเทียบกับวิธี A-DAD โดย A-DAD ใช้เวลา 1300 ms กรณีที่มีความผิดพลาด เมื่อสามารถลดการตรวจสอบได้แล้ว เวลาในการทำงานโดยรวมของ Handover Layer-3 จะลดลงตามภาพที่ 30 และ 31 เห็นได้ว่าวิธี FR-DAD ที่ได้พัฒนาขึ้นนี้สามารถลดเวลาจากการทำงาน Standard DAD ได้ 99.74% (เทียบที่ 1.89 ms กับ 1000 ms)ในกรณีที่ไม่มีความผิดพลาดและกรณีที่มีความผิดพลาดใช้เวลาเท่ากับ Standard DAD และใช้เวลาน้อยกว่าวิธี A-DAD 23% (เทียบที่ 1000 ms กับ 1300 ms) เมื่อวิเคราะห์ในด้านของปริมาณการใช้ข้อความของแต่ละวิธี FR-DAD และ A-DAD ใช้จำนวนของข้อความใกล้เคียงกัน ซึ่งมากกว่าวิธี Standard DAD อยู่มากพอสมควร แต่เมื่อเทียบกับเวลาลดลงที่ได้รับทำให้เป็นค่าที่ยอมรับได้

ข้อเสนอแนะ

แนวทางการพัฒนาสำหรับวิธี FR-DAD นี้เป็นการพัฒนาให้นำไปใช้ได้กับเครือข่ายแบบเคลื่อนที่ที่ใช้เลขหมาย IPv6 ในเครือข่ายจริง จากการทดลองด้วยโปรแกรม Omnet++ สามารถทำงานได้ตามพารามิเตอร์ที่ได้กำหนดไว้ แต่ยังขาดการทดสอบวิธี FR-DAD กับระบบเครือข่าย Mobile IPv6 จริง ซึ่งในการวัดประสิทธิภาพการทำงานเมื่อใช้กับการเคลื่อนย้าย MN จริงนั้น อาจจะใช้เวลาในการทำงานมากขึ้น เนื่องจากเป็นการรับส่งข้อความจริงและผ่านสัญญาณสื่อสารจริงๆ

ปัญหาของวิธี FR-DAD เกิดขึ้นได้จากอุปกรณ์ P-Server ไม่ทำงานแต่เวลาที่สูญเสียนั้นจะเท่ากับเวลาที่ใช้ใน DAD แบบมาตรฐานคือ 1000 มิลลิวินาที เสมือนว่าไม่มี P-Server อยู่ในเครือข่าย และถ้า P-Server เกิดขึ้นในเครือข่ายที่มีการทำงานอยู่แล้ว การเก็บเลขหมายจำนวนหนึ่งลง pool จะเสียเวลาเท่ากับ 1000 มิลลิวินาทีต่อหนึ่งเลขหมาย IPv6 ซึ่งขึ้นกับความสามารถของอุปกรณ์ P-Server ว่าสามารถทดสอบเลขหมาย IPv6 พร้อมๆกันได้เท่าไร โดยที่ยังสามารถจ่ายเลขหมาย NCoA ให้กับ MN ที่ต้องการได้ เห็นได้ว่าปัญหาตรงจุดนี้เป็นความสามารถการรับการทดสอบ DAD หรือจ่ายเลขหมาย NCoA ได้สูงสุดโดยที่ P-Server ยังทำงานได้ โดยขึ้นอยู่กับประสิทธิภาพของอุปกรณ์ที่นำมาใช้เป็น P-Server

ในการออกแบบ FR-DAD นั้น ไม่ได้คำนึงถึงความปลอดภัยของการรับ-ส่งข้อความ มีความเป็นไปได้ที่จะมีผู้ที่ดักฟังข้อความของ MN ทำให้ทราบ MAC address ของ MN และนำไปปลอมตัวเป็น MN เพื่อรับข้อมูลสำคัญบางประการได้ และในกรณีที่ไม่ได้รับเลขหมาย NCoA สามารถเพิ่มเวลา timeout การรอข้อความ address reply เพื่อร้องขอเลขหมาย NCoA อีกครั้งด้วยข้อความ address request ก่อนที่การทำงานของ DAD แบบมาตรฐานจะเสร็จสิ้น เป็นส่วนงานที่ไม่ได้ทำการทดสอบถึงความเหมาะสม ซึ่งอาจจะช่วยแก้ไขการทำการส่งข้อความที่ไปไม่ถึงให้มีโอกาสส่งเพิ่มขึ้นหรืออาจเป็นการเพิ่มจำนวนข้อความในเครือข่ายโดยไม่จำเป็นก็เป็นได้

ในกรณีที่มีการใช้งานวิธีการ DAD หลายวิธีในเครือข่ายเดียวกันได้แก่ DHCPv6 A-DAD และ FR-DAD การส่งข้อความร้องขอเลขหมายจะไม่เกิดปัญหาใดๆ เพราะแต่ละวิธีมีการใช้ข้อความที่แตกต่างกันคือ DHCPv6 ใช้การสื่อสารด้วยข้อความของ DHCPv6 โดยเฉพาะ A-DAD จะใช้ข้อความ RS ที่มี option ใหม่และ FR-DAD และ FR-DAD ใช้ข้อความ NS มี ‘FR’ flag และ

option ใหม่ ซึ่งทำให้ระบบอุปกรณ์ที่สำรองเลขหมายแต่ละตัวสามารถติดต่อกับ MN ที่ต้องการเลขหมายตามวิธีนั้นๆ ได้อย่างไม่เกิดความสับสนของการสื่อสารแต่อย่างใด

จากปัญหาที่ P-Server เป็น single point of failure และข้อจำกัดการทำงานของ P-Server การใช้ P-Server 2 อุปกรณ์ทำงานคู่ขนานกันไปโดยออกแบบให้มีข้อความรับ-ส่งระหว่าง P-Server เพื่อป้องกันการจ่ายเลขหมายซ้ำซ้อน และถ้าเครือข่ายใดมีการทำงานของลูกข่าย MN ไม่มากนักเพื่อลดค่าใช้จ่าย P-Server ต่อหนึ่งเครือข่ายลงโดยให้ P-Server ทำงานอยู่ระหว่างเครือข่าย โดย P-Server ดูแลมากกว่าหนึ่งกลุ่ม prefix ก็เป็นปัญหาที่น่าสนใจอีกปัญหาหนึ่ง

เอกสารและสิ่งอ้างอิง

Silvia Hagen. 2002. **IPv6 Essential.** O'Reilly, California.

Wolfgang Fritsche and Florian Heissenhuber. 2000. **Mobile IPv6: Mobility support for the Next Generation Internet.** IABG mbH, Ottobrunn Germany.

T. Narten, E. Nordmark, W. Simpson and H. Soliman. 2007. **RFC 4861 - Neighbor Discovery for IP version 6 (IPv6).** Internet Engineering Task Force. Available Source: http://tools.ietf.org/html/rfc4861, December 23, 2007.

S. Thomson, T. Narten and T. Jinmei. 2007. **RFC 4862 - IPv6 Stateless Address Autoconfiguration.** Internet Engineering Task Force. Available Source: http://tools.ietf.org/html/rfc4862, December 23, 2007.

N. Moore. 2006. **RFC 4429 - Optimistic Duplicate Address Detection for IPv6.** Internet Engineering Task Force. Available Source: http://tools.ietf.org/html/rfc4429, January 10, 2007.

Y. Han, J. Choi and H. Jang. 2003. **Internet Draft - Advance Duplicate Address Detection.** Internet Engineering Task Force. Available Source: http://tools.ietf.org/html/draft-han-mobileip-adad-01, January 20, 2007.

Chien-Chao Tseng, Yung-Chang Wong, Li-Hsing Yen and Kai-Cheng Hsu. 2006. Proactive DAD: A Fast Address-Acquisition Strategy for Mobile IPv6 Networks. **IEEE Internet Computing 2006**: 50-55.

R. Droms, Ed. 2003. **RFC 3315 - Dynamic Host Configuration Protocol for IPv6 (DHCPv6).** Internet Engineering Task Force. Available Source: http://tools.ietf.org/html/rfc3315, January 20, 2007.

R. Koodli. 2005. **RFC 4429 - Fast Handovers for Mobile IPv6.** Internet Engineering Task Force. Available Source: http://tools.ietf.org/html/rfc4429, January 20, 2007.

C.C. Tseng. 2005. Location-Based Fast Handoff for 802.11 Networks. **IEEE Comm. Letters 2005** (vol.9, no.4): 304–306.

D. Johnson, C. Perkins and J. Arkko. 2004. **RFC 3775 - Mobility Support in IPv6.** Internet Engineering Task Force. Available Source: http://tools.ietf.org/html/rfc3775, January 30, 2007.

Y.-H. Han and S.-H. Hwang. 2006. Care-of address provisioning for efficient IPv6 mobility support. **Computer Communications 29**: 1422-1432.

Anonymous. 2006. OMNeT++ Community Site. Available Source: http://www.omnetpp.org, Febuary 17, 2007.

Adisak Busaranun, Panita Pongpaibool and Pichaya Supanakoon. 2006. Handover Performance of Mobile IPv6 on Linux Testbed, pp. 131-134. **ECTI-CON 2006.** NECTEC, Pathumthani Thailand.

ภาคผนวก

ภาคผนวก ก

ผลงานตีพิมพ์

1. Fast and Robust Duplicate Address Detection for Mobile IPv6

(ITST2008), ตุลาคม 22-24, 2551

2. Investigation of Duplicate Address Detection in Mobile IPv6

(NAC2007), มีนาคม 29-31, 2550

3. Fast Duplicate Address Detection for Mobile IPv6

(ICON2007), 2550

Fast and Robust Duplicate Address Detection for Mobile IPv6

Pahol Sotthivirat[1], Panita Pongpaibool[2], Sukumal Kitisin[1], Chavalit Srisathapornphat[1]

[1] Kasetsart University, 50 Pahol Yothin Rd., Chatuchak, Bangkok 10900 THAILAND

mr.pahol@gmail.com, sukumal.i@ku.ac.th and chavalit.s@ku.ac.th

[2] NECTEC, 112 Pahol Yothin Rd., Klong Luang, Pathumthani 12120 THAILAND panita@nectec.or.th

***Abstract*— In Mobile IPv6 networks, duplicate address detection (DAD) process is necessary for confirming uniqueness of IPv6 address when Mobile Node moves to a new network. In standard IPv6 protocol, DAD delay takes at least 1000ms. It is desirable to reduce DAD delay especially if nodes wish to run real-time applications on the Mobile IPv6 networks. This paper proposes Fast and Robust DAD (FR-DAD) to improve handover delay and provide reliability and backward compatibility with standard IPv6 networks. FR-DAD is a semi-stateful address assignment system. It utilizes a server that manages a list of unique IPv6 addresses. Simulations show FR-DAD successfully reduces delay by as much as 99.74% of standard DAD delay, in best case. In the worst case, FR-DAD still outperforms a similar stateful DAD technique by as much as 23%. FR-DAD can provide unique IPv6 addresses for mobile nodes quickly and reliably without adding any more overhead to the network than previous techniques.**

Keywords: Mobile IPv6, Duplicate Address Detection

I. INTRODUCTION

In preparation for the next generation network on mobile devices, Mobile IPv6 [1] has been developed to manage movement of Mobile Nodes (MN) in wireless IPv6 networks. Mobile IPv6 provides seamless handover when MN moves to different networks. MN can continue to use its home address. When MN moves away from home, it must register its new IP address with a Home Agent (HA) in the home network.

Any packets sent to MN are sent to home network first. Then the HA forwards the packets to MN at the foreign network.

Layer-3 handover process in the Mobile IPv6 network is as follows. After MN moves to a foreign network and establishes connection with an access point (i.e., finishes layer-2 handover process) it creates a new address, called *care-of address* (CoA) from prefix information in the Router Advertisement (RA) message sent by the access router. After that, MN performs *duplicate address detection* (DAD) to confirm uniqueness of this CoA. When the CoA passes the test, MN registers its CoA with HA via a *binding update* (BU) message. HA confirms the registration with a *binding acknowledgment* (BA). When MN finishes registering its new address, it can receive any packets from existing connections forwarded from HA.

Handover latency in Mobile IPv6 directly affects performance of time critical applications such as voice-over-IP. Reference [2] investigates four components of delay time during mobile IPv6 handover: (1) movement detection time, (2) address configuration time, (3) binding registration time, and (4) route optimization time. Testbed experiments show that address configuration time is the largest delay time due to DAD process. Although MN can turn off the DAD process, it is not recommended. This is because IP address is at risk for duplicating, especially in a large IPv6 networks.

According to RFC 4862 IPv6 Stateless Address Autoconfiguration, after MN receives prefix information via RA message, MN randomly generates an interface ID or use a hardware address as an interface ID. A combination of subnet prefix and interface ID is referred to as Tentative Address. Before the Tentative Address can be used, it must undergo DAD test to check for uniqueness within the network. The DAD test works as follows. First, an IPv6 node sends a *Neighbor Solicitation* (NS) message to a solicited-node multicast group asking if anyone is using this Tentative Address. In this NS-DAD message, source IP address is all zero or unspecified and the target address is the Tentative Address. If the address has already been used by another neighbor node, that node must defend itself by sending a *Neighbor Advertisement* (NA) message to all nodes. The node that initiates the NS-DAD has to set up a new Tentative Address

and repeat the DAD process. If it does not hear any defending NA message within *RetransTimer* milliseconds, it may ask again *DupAddrDetectTransmits* times to make sure there is really no conflict. After that, it can start using the Tentative Address. So the DAD process takes at least *RetransTimer*·*DupAddrDetectTransmits* milliseconds. Under default setting, the delay is at least 1000 ms, where *RetransTimer* = 1000 and *DupAddrDetectTransmits* = 1. This delay time is not acceptable in real-time applications. Although one can decrease RetransTimer below 1000 ms, this increases a risk of not detecting duplicate addresses.

In this paper, we propose a fast and robust DAD procedure (FR-DAD). The proposed DAD improves handover delay, and is compatible with IPv6 standards. The proposed procedure guarantees unique IP address configuration much faster than the standard DAD mechanism. Moreover, the new procedure is robust against packet loss, network error, and is backward compatible with standard IPv6 and Mobile IPv6 networks.

The rest of this paper is organized as follows. Section II discusses the existing techniques for improving DAD. Section III presents our new DAD protocol. Performance analysis of the new DAD is shown in section IV. Finally section V provides discussion and conclusions.

II. Related Work

This section discusses features and characteristics of existing methods that aim to improve DAD delay. For a comprehensive review of DAD techniques, readers are referred to reference [5].

Advanced DAD [7, 8]

The Advanced DAD or A-DAD technique improves DAD delay by storing a pool of unique IPv6 addresses at an access router (AR). Each AR generates random addresses as a background process and performs Standard DAD on them. The duplicate-free addresses are stored at the AR. AR acts as a passive proxy for addresses. It listens to neighbor discovery messages from other nodes in the network. If it hears another node performing DAD on the same address in its pool, AR silently removes that address from its list and tests a new address to keep the list size constant.

A-DAD is designed to work in with

movement prediction, utilizing Fast Handover technique [9], as well as without movement prediction. A-DAD modifies Router Solicitation (RS) and Router Advertisement messages with new options (Fig. 1). MN sends RS with NCoA-Request option to the AR of the new network, and waits for RA with NCoA-Reply option which contains new CoA from the AR. A-DAD puts extra burden on AR. AR has to take care of both routing and managing a list of unique IP addresses. If RA or RS message is lost, or new networks do not support A-DAD, MN falls back to use Standard DAD. Recovery time in this case is even longer than Standard DAD delay.

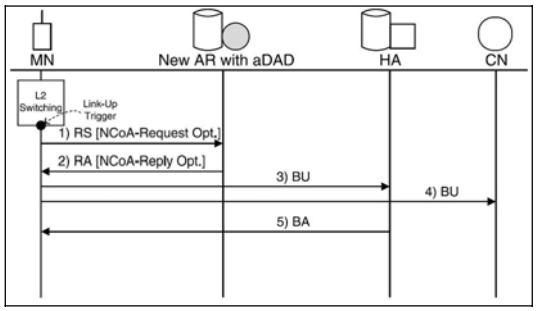


Fig. 1 A-DAD process [8]

A. Other DAD Techniques

There are several other techniques which aim to reduce delay time to confirm unique CoA. Optimistic DAD [6] assumes probability of address collision is very small. So it allows MN to use Tentative Address while the uniqueness test is in process. Proactive DAD [12] checks DAD on new CoA before MN moves to a new network. MLD-DAD [13] and END [14] take advantage of IPv6 multicast listener discovery. Both assume all nodes must join a solicited-node multicast address associated with each unicast address. So to verify address uniqueness, they simply check whether a specific solicited-node multicast group is empty. These methods could end up with false duplicates because multiple unicast addresses could join the same solicited node multicast group.

Another common technique is to terminate DAD early by sending positive acknowledgement. MLD-DAD, END, and FNDD [10] all send a message to inform MN that the Tentative Address is unique. MN can start using the address right away without waiting for the *RetransTimer*. To make a positive confirmation, a central entity that keeps track of addresses in use is necessary. All methods discussed in this section, except Optimistic DAD, require an entity such as router or server to manage an address list.

Inherently IPv6 address configuration is a

stateless process. Therefore, it is necessary to perform DAD. To eliminate DAD delay completely, one could opt for a stateful process at the cost of network scalability. For example, A-DAD, FNDD, and DHCPv6 [11] are stateful address assignment. One drawback of stateful configuration is that MN has no choice to what address is assigned. Consequently, novel addressing schemes cannot be used.

III. New Method

The proposed *Fast and Robust Duplicate Address Detection*, or FR-DAD, method is based on the idea of stateful address configuration. Similar to A-DAD, FR-DAD utilizes an additional server, called *P-Server*, to store unique IPv6 addresses. This server is responsible for assigning an IPv6 address from its pool to a MN that recently joins the network. The unique IPv6 address pool is constructed via the standard DAD process. The P-Server, which is required in every network, acts like an access router in A-DAD. However, by separating the address server from the access router, we reduce access router's workload. Communication between the server and the MN uses standard Neighbor Discovery messages with a new flag and a new option. So the new method can work with standard IPv6 networks. The FR-DAD also handles cases of packet loss and works with standard mobile IPv6 nodes and networks. Moreover, FR-DAD does not depend on movement detection. Thus, it is suitable and effective on both wired and wireless IPv6 network

Fig. 2 shows network architecture and communication steps of FR-DAD. When a MN attempts to configure a new address, it exchanges three ICMP Neighbor Discovery messages with P-Server: Address Request, Address Reply, and Address Acknowledgement. The three-way acknowledgement ensures the P-Server the assigned address really reaches MN. This three-way handshake tackles a flaw in all previous works which rely on just request and reply messages. With the two-way handshake, P-Server would not detect the loss of a reply packet. As a result, it would not know whether the assigned address is taken or not. On the other hand, with the three-way handshake, P-Server can detect the loss of reply message. Thus, it can maintain an accurate view of its address pool.

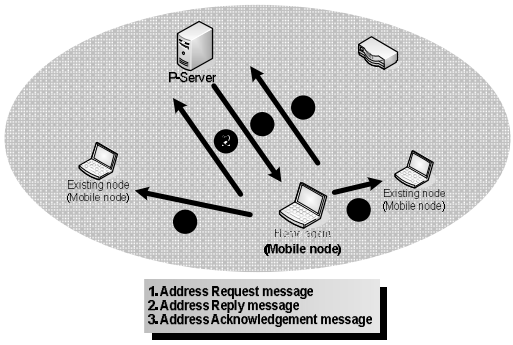


Fig. 2 Network Topology for FR-DAD

The FR-DAD defines a new flag and a new option to use with the three Neighbor Discovery messages exchanged between P-Server and MN. First, the Address Request is an ICMP NS message with a new FR flag in the reserved field, and a new FR option containing Link-Layer address (Fig. 3). MN sends this Address Request after it generates a Tentative Address based on prefix information received from a router. This Address Request message uses the same parameters as those in the standard NS-DAD message. For example, the source IP address is unspecified. The destination IP address is the solicited-node multicast address of the Tentative Address. The Target Address is the Tentative Address. In addition, the FR option contains a Link-Layer Address of the MN in order for the P-Server to know where to return the Address Reply message. In other words, the MN is testing DAD of the Tentative Address at the same time it is requesting a new care-of-address (NCoA) from the P-Server. P-Server is set up so that it listens to all solicited-node multicast addresses with the FR flag.

Secondly, the Address Reply is an ICMP NA message with the FR flag in the reserved field, and the Target Link-Layer Address option (Fig. 4a). When the P-Server receives the Address Request message that is known by FR flag and new option, it chooses a unique IPv6 address (NCoA) from its pool and sends it in the Address Reply message back to an all-node multicast group. The Address Reply message is an unsolicited-NA message. The Target Link-Layer Address (TLLA) option contains the Link-Layer Address of the MN. In addition, the Address Reply message carries NCoA in the Target Address field.

Finally, when MN receives the Address Reply message that is identified specifically by the 'FR' flag and the Target Link-Layer Address equals its own Link-Layer Address, MN uses the Target Address as the new Care-of Address. At this point, the NCoA is ready. MN can send Binding Update to register the NCoA with the HA, as well as sending Address Acknowledgement

message to confirm the P-Server that it will use this NCoA. The Address Acknowledgement message is an unsolicited-NA with the 'FR' flag, the override flag, and the TLLA option (Fig. 4b). The TLLA option and override flag are necessary because they force MN's neighbors to map the Target Address with MN's Link-Layer Address. When the P-Server hears this Address Acknowledgement, it can remove this NCoA completely from its pool.

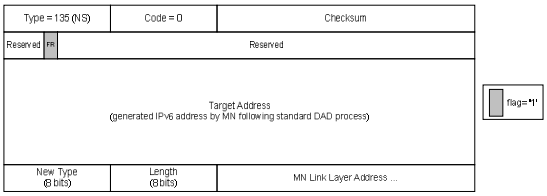
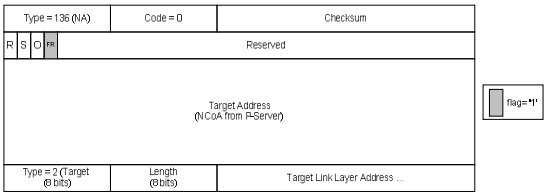
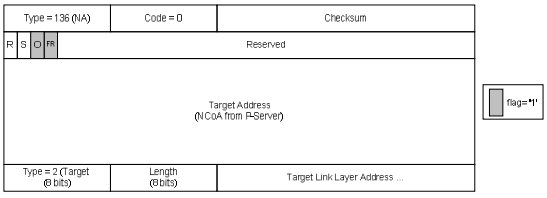


Fig. 3 NS with New Option for FR-DAD (Address Request)



(a) Address Reply



(b) Address Acknowledgement

Fig. 4 NA messages for FR-DAD

Note that while the three-way address negotiation is going, the standard DAD procedure also occurs in the background. The MN's Address Request message initiates this standard DAD procedure. However, the P-Server is the one that listens for a duplicate-defending message. If the *RetransTimers* expires, then the P-Server can suppose the tested address is unique and store it in its address pool (Fig. 5). Otherwise, the P-Server simply ignores this address. The P-Server may generate another random address and test for duplication to maintain the address pool supply level.

To demonstrate robustness of FR-DAD, let us show what happens when MN does not receive NCoA from P-Server. This could occur due to loss of Address Request, loss of Address Reply, or because P-Server does not exist in the foreign network. This means the 1000-ms *RetransTimers* will expire before MN receives the Address Reply message. In such case, as shown in Fig. 6, MN uses the Tentative Address that passes the DAD test as the NCoA. MN then sends the Address Acknowledgement message (unsolicited-NA), with the Target Address being the IPv6 address that passes standard

DAD process, to confirm that it will use this IPv6 address. If P-Server exists in the network, it will know that MN does not receive its Address Reply. Therefore, the NCoA it gives out is still available in the pool.

Another possibility is the loss of Address Acknowledgement message. This event does not affect MN since it already obtains a unique address. It can continue with handover process normally. However, P-Server must keep an acknowledgement timer. If this timer expires before it receives Address Acknowledgement, P-Server must remove both the selected NCoA and the Tentative Address of this MN (if passing the DAD test) from the address pool. No acknowledgement could mean MN did not receive NCoA (MN would use Tentative Address), or MN has received and used NCoA. Since P-Server does not know the cause, it assume both addresses are no longer available.

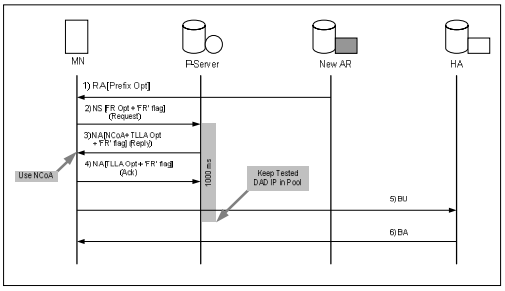


Fig. 5 FR-DAD Process in ideal case

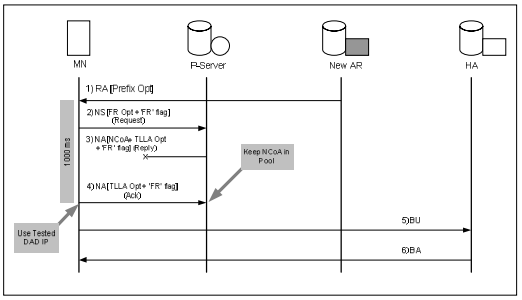


Fig. 6 FR-DAD Process in missing NCoA case

IV. Performance Analysis

A. Simulation Environment

Performance of the FR-DAD is simulated under *Omnet++ simulator* with IPv6 Suite for INET framework [15] on Kubuntu Linux OS. The proposed mechanism is implemented as additional methods in Omnet++ modules. Simulated network consists of three Mobile IPv6 networks: Home Network (HN), Foreign network 1 (FN1), and Foreign network 2 (FN2) according to Fig. 7 MN movement begins from HN to FN1, then to FN2, and finally back to HN. From this movement, MN must

change IPv6 address three times along each network. We study FR-DAD performance in terms of DAD delay, overall layer-3 handover delay, and message overhead. These performance metrics are compared among FR-DAD, Standard DAD and A-DAD (non predictive case).

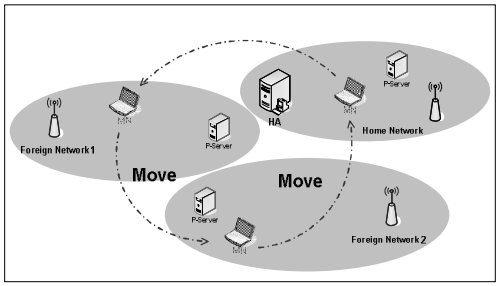


Fig. 7 Network for Simulation

B. Delay Performance

To benchmark delay performance of FR-DAD with Standard DAD and A-DAD, we set up five simulation scenarios: FR-DAD ideal case, A-DAD ideal case, Standard DAD, FR-DAD with NCoA missing, and A-DAD with NCoA missing. For each scenario, we measure DAD delay and overall handover delay during movement of MN from HN to FN1 to FN2 and to HN. We repeat the simulation ten times for each scenario. We average measurements over ten experiments. Simulation parameters are given in Table 1.

TABLE 1
SIMULATION PARAMETERS

RetransTimer	1000 ms
DupAddrDetectTransmits	1
Router Advertisement interval	(1 s, 1.5 s)
P-Server's *Acknowledgement Timer*	1500 ms
A-DAD's *NCoA Timeout*	300 ms

Fig. 8 shows average DAD delay of the five scenarios, under each movement direction. DAD delay measures the time from MN sending the Address Request (NS-DAD) until MN obtains a unique IPv6 address. In the ideal cases (e.g., no packet loss), FR-DAD and A-DAD take only around 3 ms for MN to complete address configuration. This delay is much faster than the DAD delay of Standard DAD which takes 1000 ms. In the case that MN does not receive NCoA (due to packet loss or no P-Server in network), FR-DAD delay equals to 1000 ms of Standard DAD process. This is because MN finishes address configuration when it receives Address Reply or when the Standard DAD's *RetransTimer* expires, whichever event comes first. As for A-DAD with missing NCoA, its DAD delay is longer than 1000 ms. This is because it must

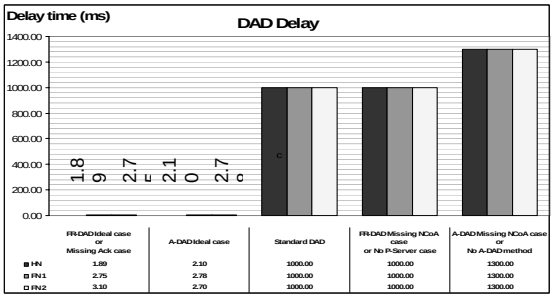


Fig. 8 DAD Delay time

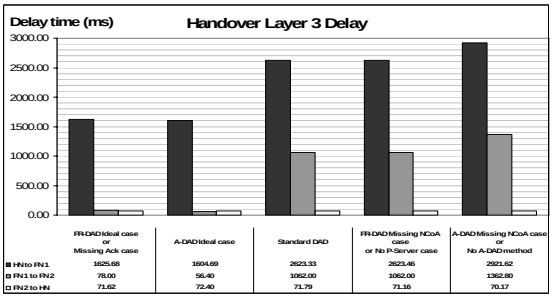


Fig. 9 Handover layer-3 Delay time

wait for a timeout to detect missing NCoA Reply before it can start the Standard DAD process. If we assume the *NCoA Timeout* is 300 ms, the DAD delay in this case is as long as 1300 ms. Moreover, variation of DAD delay in each network is very small.

Fig. 9 shows layer-3 handover delay of the five scenarios, under each movement direction. The overall handover delay includes DAD delay plus movement detection and binding registration delay. We measure the time from when an access point detects a new node (Layer-2 handover completes) until MN receives Binding Acknowledgement (binding registration completes). In the ideal case, handover delay of FR-DAD and A-DAD is very much the same. A-DAD handover is slightly faster than FR-DAD because A-DAD does not have to wait for prefix information from the Router Advertisement. In the case of missing NCoA, FR-DAD's handover delay approaches that of Standard DAD, but A-DAD's handover delay is much slower than that of both FR-DAD and Standard DAD. This is because A-DAD has additional timeout waiting for the NCoA Reply message.

The difference between Fig. 8 and Fig. 9 is that we can see large variation of handover delay among different movement directions. Moving out of Home Network the first time takes the longest delay, especially in binding registration. Moving between the two foreign networks is faster, but returning to Home Network is the fastest. This is because MN can skip both binding registration and DAD.

C. Message Overhead

Our proposed FR-DAD technique achieves quickness and robustness by sending additional control messages. In this section, we compare number of control messages required under the proposed FR-DAD technique to that of standard DAD and A-DAD techniques. The comparison counts messages generated by MN, P-Server, and

routers.

Let M_x represent a message of type x. Let d_1 be the number of IP duplications generated by MN and d_2 be the number of IP duplications generated by FR-DAD's P-Server or A-DAD's access router. The number of control messages for protocol y is represented by N_y.

For Standard DAD process with no duplication, it needs *DupAddrDetectTransmits* NS-DAD messages to test a Tentative Address. With duplication, a neighbor sends NA to protect its address, and MN resends another NS-DAD. This repeats until the MN gets a unique IP address.

$$N_{StdDAD} = M_{NS\text{-}DAD} \cdot DupAddrDetectTransmits + (M_{NA} + M_{NS\text{-}DAD})d_1$$

$$N_{StdDAD} = 2d_1 + DupAddrDetectTransmits \quad (1)$$

For A-DAD ideal case, MN sends one RS (NCoA-Request) message to request IP address. Then an access router replies with one RA (NCoA-Reply) message. Duplication does not happen. However, when one address is taken from the pool, the router must compute one unique address via Standard DAD to replace it.

$$N_{A\text{-}DAD\text{-}1} = M_{RS(req)} + M_{RA(reply)} + N_{StdDAD\text{-}AR}$$

$$= 1 + 1 + 2d_2 + DupAddrDetectTransmits$$

$$N_{A\text{-}DAD\text{-}1} = 2d_2 + DupAddrDetectTransmits + 2 \quad (2)$$

For A-DAD with a lost NCoA-Reply, MN must try again with Standard DAD. Access router also has to replenish the address pool with one address. So the overhead includes the lost message as well as messages required for one test of Standard DAD.

$$N_{A\text{-}DAD\text{-}2} = M_{RS(req)} + M_{RA(reply\text{-}lost)} + N_{StdDAD\text{-}MN} + N_{StdDAD\text{-}AR}$$

$$= 1 + 1 + (2d_1 + DupAddrDetectTransmits) + (2d_2 + DupAddrDetectTransmits)$$

$$N_{A\text{-}DAD\text{-}2} = 2d_1 + 2d_2 + 2DupAddrDetectTransmits + 2 \quad (3)$$

Next we consider the case where the access router does not support A-DAD. MN sends one RS message but the method is not established in a network. When the *NCoA Timeout* expires, MN retries with Standard DAD process.

$$N_{A\text{-}DAD\text{-}3} = M_{RS(req)} + N_{StdDAD\text{-}MN}$$

$$N_{A\text{-}DAD\text{-}3} = 2d_1 + DupAddrDetectTransmits + 1 \quad (4)$$

In the ideal case of FR-DAD, MN exchanges three messages with P-Server. If the IP address generated by MN is a duplicate, P-Server must generate a new address and test Standard DAD until the address is unique. Number of messages in this case is equal to eq. (2).

$$N_{FR\text{-}DAD\text{-}1} = M_{NS(req)} + M_{NA(reply)} + M_{NA(ack)} + (N_{StdDAD\text{-}Pserver} - M_{NS(req)})$$

$$N_{FR\text{-}DAD\text{-}1} = 2d_2 + DupAddrDetectTransmits + 2 \quad (5)$$

In the case of FR-DAD where Address Acknowledgement is lost, the only difference is that P-Server cannot keep neither the IP address generated by MN nor NCoA. As a result, P-Server must generate new address and test Standard DAD itself to replenish its address pool.

$$N_{FR\text{-}DAD\text{-}2} = M_{NS(req)} + M_{NA(reply)} + M_{NA(ack\text{-}lost)} + N_{StdDAD\text{-}Pserver}$$

$$N_{FR\text{-}DAD\text{-}2} = 2d_2 + DupAddrDetectTransmits + 3 \quad (6)$$

For FR-DAD with a loss of Address Reply, MN relies on Standard DAD. If the tested address is unique, an Address Acknowledgement completes FR-DAD. Otherwise, MN must resend NS-DAD in Address Request format. There are two outcome possibilities. The second Address Reply may arrive, and MN ends up using NCoA assigned by P-Server. In this case P-Server must compute one unique address to replace the assigned NCoA, as in eq. (7). Or the Address Reply may be lost again causing MN to use Tentative Address. The overhead is shown in eq. (8).

$$N_{FR\text{-}DAD\text{-}3} = (M_{NS(req)} + M_{NA(reply\text{-}lost)} + M_{NA})d_1 + M_{NS(req)} + M_{NA(reply)} + M_{NA(ack)} + (N_{StdDAD\text{-}Pserver} - M_{NS(req)})$$

$$N_{FR\text{-}DAD\text{-}3} = 3d_1 + 2d_2 + DupAddrDetectTransmits + 2 \quad (7)$$

$$N_{FR\text{-}DAD\text{-}4} = (M_{NS(req)} + M_{NA(reply\text{-}lost)} + M_{NA})d_1 + (M_{NS(req)} + M_{NA(reply\text{-}lost)}) DupAddrDetectTransmits + M_{NA(ack)}$$

$$N_{FR\text{-}DAD\text{-}4} = 3d_1 + 2DupAddrDetectTransmits + 1 \quad (8)$$

If MN performs FR-DAD where P-Server does not exist, MN receives no Address Reply. MN relies on Standard DAD test until finding a unique address. Then MN

sends Address Acknowledgement to complete FR-DAD. Number of messages in this case is similar to (4).

$$N_{FR\text{-}DAD\text{-}5} = N_{StdDAD\text{-}MN} + M_{NA(ack\text{-}lost)}$$

$$N_{FR\text{-}DAD\text{-}5} = 2d_1 + DupAddrDetectTransmits + 1 \quad (9)$$

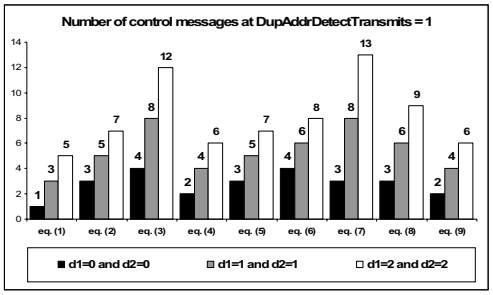


Fig. 10 Comparison of number of control messages

In all scenarios, the number of control messages depends on number of IP duplications. Fig. 10 shows message overhead with different combinations of d_1 and d_2. Standard DAD (eq. 1) requires the lowest overhead. In the ideal case, FR-DAD (eq. 5) requires the same amount of messages and delay time as A-DAD (eq. 2). In the error cases, FR-DAD needs slightly more control messages than A-DAD. More messages help FR-DAD complete DAD faster than A-DAD.

V. Conclusion

In this paper, we propose the Fast and Robust Duplicate Address Detection (FR-DAD) technique. We demonstrate that FR-DAD can help reducing DAD delay time in Mobile IPv6 networks. In the ideal case or best case, FR-DAD reduces the DAD delay by as much as 99.74% of Standard DAD delay (1.89 ms vs. 1000 ms). Although the best-case performance of FR-DAD is similar to a previous technique called Advanced DAD (A-DAD), FR-DAD outperforms A-DAD in error cases (worst case) by 23% (1000 ms vs. 1300 ms). Moreover, FR-DAD delay in the worst case only takes as long as the Standard DAD delay. The benefit of FR-DAD comes at the cost of more signaling message overhead than the Standard DAD. However, the number of FR-DAD messages is about the same as that of A-DAD. In conclusion, the FR-DAD can provide unique IPv6 address quickly and reliably without adding any more overhead to the network than previous improvement techniques.

FR-DAD is designed to be backward-compatible and robust against message loss and error. FR-DAD can work in standard IPv6/Mobile IPv6 network. Modification is required only in mobile nodes and in the P-

Server. FR-DAD offers an alternative for mobile nodes that wish to have faster address autoconfiguration. Future work includes performance study in real mobile IPv6 networks.

REFERENCES

[1] D. Johnson, C. Perkins and J. Arkko, "Mobility Support in IPv6," RFC 3775, June 2004.

[2] A. Busaranun, P. Pongpaibool and P. Supanakoon, "Handover Performance of Mobile IPv6 on Linux Testbed," ECTI-CON 2006.

[3] S. Thomson, T. Narten and T. Jinmei, "IPv6 Stateless Address Autoconfiguration," IETF RFC 4862, Sep 2007.

[4] T. Narten, E. Nordmark, W. Simpson and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)," IETF RFC 4861, Sep 2007.

[5] P. Pongpaibool, P. Sotthivirat, S. I. Kitisin and C. Srisathapornphat, "Fast Duplicate Address Detection for Mobile IPv6," ICON 2007.

[6] N. Moore, "Optimistic Duplicate Address Detection for IPv6," IETF RFC 4429, April 2006.

[7] Y. Han, J. Choi and H. Jang, "Advance Duplicate Address Detection," Internet Draft, Dec 2003, expired.

[8] Y.-H. Han, S.-H. Hwang, "Care-of address provisioning for efficient IPv6 mobility support," Computer Communications 29(2006), pp.1422-1432.

[9] E. Rajeev Koodli, "Fast Handovers for Mobile IPv6," IETF RFC 4068, July 2005.

[10] B. Park, S. Lee, and H. Latchman, "A Fast Neighbor Discovery and DAD Scheme for Fast Handover in Mobile IPv6 Networks," Proc. Of ICNICONSMCL'06.

[11] R. Droms, Ed., "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)," IETF RFC 3315, July 2003.

[12] C.-C. Tseng, Y.-C. Wong, L.-H. Yen and Kai-Cheng Hsu, "Proactive DAD: A Fast Address-Acquisition Strategy for Mobile IPv6 Networks," Dec 2006.

[13] G. Daley and R. Nelson, "Duplicate Address Detection Optimization using IPv6 Multicast Listener Discovery," Internet Draft, Feb 2003, expired.

[14] F. Xia and B. Sarikaya, "Duplicate Address Detection Optimization Using

Enhanced Neighbor Discovery,"
Internet Draft, Dec 2006.

[15]OMNeT++ Discrete Event Simulation
System, http://www.omnetpp.org.

Investigation of Duplicate Address Detection in Mobile IPv6

พหล โสตถิวิรัช ดร.พนิตา พงษ์ไพบูลย์ ผศ.ดร.สุขุมาล กิติสิน ดร.ชวลิต ศรีสถาพรพัฒน์
มหาวิทยาลัย ศูนย์เทคโนโลยีอิเล็กทรอนิกส์ มหาวิทยาลัย มหาวิทยาลัย
เกษตรศาสตร์ และคอมพิวเตอร์แห่งชาติ เกษตรศาสตร์ เกษตรศาสตร์

บทคัดย่อ -- **ในระบบเครือข่ายแบบเคลื่อนที่ที่ใช้ IPv6 (Mobile IPv6 Network) ที่มีการเคลื่อนที่ของอุปกรณ์ภายในเครือข่ายและการเคลื่อนที่ระหว่างเครือข่ายอยู่ตลอดเวลา เพื่อการติดต่อสื่อสารที่มีการเคลื่อนที่อยู่ตลอดเวลาให้สามารถทำงานได้อย่างราบรื่น ทำให้การติดตั้ง IPv6 address แบบอัตโนมัติเข้ามามีบทบาท เพราะช่วยให้ผู้ใช้งานไม่ต้องสนใจการติดตั้ง IPv6 address เมื่อมีการเคลื่อนย้ายอุปกรณ์ระหว่างเครือข่ายที่มีกลุ่มของ IPv6 address คนละชุดที่เปลี่ยนแปลงอยู่ตลอด โดยปกติแล้วขั้นตอนการตั้งค่า IPv6 address สามารถทำได้อัตโนมัติ โดย Router จะมีหน้าที่แจกจ่าย Network prefix ให้กับเครื่องลูกข่าย และเครื่องลูกข่ายแต่ละเครื่องจะเลือก host address ของตนเองนำมารวมเข้ากับ Network prefix ที่ได้รับกลายเป็น IPv6 address ใหม่ เนื่องจากมีความเป็นไปได้ที่เครื่องลูกข่ายสองเครื่องอาจเลือกใช้ host address ชุดเดียวกัน มีผลให้ทั้งสองเครื่องมี IPv6 address ซ้ำกัน ดังนั้น IPv6 จึงกำหนดให้มีการตรวจสอบว่า IPv6 address ที่แต่ละเครื่องเลือกขึ้นมานั้นยังไม่มีใครใช้อยู่ ขั้นตอนนี้เรียกว่า Duplicate Address Detection (DAD) โดยการถามทุก node ว่ามีการใช้แล้วหรือไม่ และรอข้อความตอบกลับเป็นเวลา 1000 ms หากไม่มีใครตอบกลับมาแสดงว่ายังไม่มีใครใช้สามารถใช้ได้เลย บทความนี้จะเน้นการศึกษาขั้นตอนการทำงานที่ส่งผลต่อความล่าช้าในการตรวจสอบ IPv6 address ใหม่และเทคนิคที่สามารถนำมาใช้เพื่อลดความล่าช้าในส่วนนี้ลง**

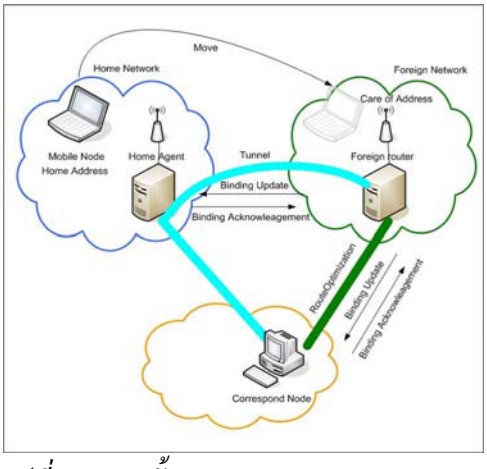
คำสำคัญ -- **IPv6, mobile IPv6, DAD, Duplicate Address Detection**

---

1. บทนำ

ในระบบเครือข่ายแบบเคลื่อนที่ที่ใช้ IPv6 (Mobile IPv6 Network) นี้ สามารถให้อุปกรณ์ลูกข่าย (Mobile Node: MN) สามารถเคลื่อนย้ายอุปกรณ์ได้ตามต้องการ โดยอุปกรณ์ในเครือข่ายจะมี IPv6 address ค่อยระบุตัวตนของอุปกรณ์ ซึ่งแต่ละเครือข่ายก็จะมีกลุ่มของ IPv6 address แตกต่างกันออกไป การจัดการในการติดต่อสื่อสารกันเมื่อ MN เคลื่อนย้ายข้ามเครือข่ายที่มีการใช้ IPv6 address คนละกลุ่มกันจึงเริ่มมีความจำเป็นโดยจัดให้ MN ใดๆ มี IPv6 address 2 ชนิด คือ Home Address เป็น IPv6 address ของ Home network ที่ MN อยู่เครือข่ายแรก และ Care-Of Address (CoA) เป็น IPv6 address ที่มีการเปลี่ยนไปตามกลุ่มของเครือข่ายที่ MN ย้ายไปอยู่ใช้ในการติดต่อสื่อสารภายในและภายนอกเครื่อข่าย ซึ่งแต่ละเครือข่ายจะมีอุปกรณ์เสริมในการจัดการการติดต่อสื่อสารกับอุปกรณ์ภายนอกเครือข่าย เรียกว่า Home Agent ในกรณีที่ MN ย้ายเข้ามาในเครือข่ายใหม่จะมีขั้นตอนหลักๆ คือ การได้รับจัดสรร Care-of Address ซึ่ง Care-of Address ที่ได้มาจะตั้งไม่มีการใช้งานอยู่ก่อน ทำให้มีขั้นตอนการตรวจสอบ Care-of Address ขึ้นมาด้วย และขั้นตอนการลงทะเบียนบอกกับ Home Agent ที่ Home network ว่า MN ได้ย้ายมาที่เครือข่ายใหม่ และบอก Care-of Address ของเครือข่ายนี้โดยผ่านทาง Home Agent ของเครือข่ายใหม่ เรียกขั้นตอนนี้ว่า Binding Update (BU)

กรณีที่มีอุปกรณ์ที่ต้องการติดต่อกับ MN (Corresponding node: CN) สามารถทำได้โดย CN ส่งข้อความที่ต้องการติดต่อไปยัง IPv6 address เดิม Home Address ของ MN ซึ่งจะถูก Home Agent ดักจับข้อความดังกล่าวไว้ แล้วส่งต่อไปให้กับ MN ที่ Care-of Address เมื่อ MN ต้องการตอบกลับสามารถส่งไปที่ CN โดยตรง ไม่ต้องผ่าน Home Agent ได้เลย เรียกการติดต่อแบบนี้ว่า Triangle Routing จะเห็นได้ว่าวิธีการติดต่อดังกล่าวทำให้สูญเสียเวลาในการส่งกลับไปให้ Home Agent สามารถแก้ปัญหาทำให้ MN และ CN ส่งถึงกันได้ตรงๆ โดยการทำขั้นตอนที่เรียกว่า Route optimization ดังรูปที่ 1



รูปที่ 1 แสดงขั้นตอนการทำงานของ Mobile IPv6

ในบทความนี้เน้นการศึกษาปัจจัยที่ส่งผลต่อความล่าช้าในการตรวจสอบ Care-of Address ซึ่งเป็นขั้นตอนหนึ่งในการได้รับจัดสรร Care-of Address ในระบบ IPv6 ทั่วไปนั้น IPv6 address ในแต่ละอุปกรณ์จะต้องไม่ซ้ำกันทำให้การตรวจสอบ IPv6 address ว่ามีการใช้อยู่หรือไม่นั้นเป็นสิ่งจำเป็นในเครือข่ายที่ต้องการความเชื่อถือ ยกตัวอย่างเช่น ถ้ามีอุปกรณ์สองอุปกรณ์ใดๆ ได้รับ IPv6 address เหมือนกัน ตัวที่ได้รับการจัดสรรทีหลัง จะไม่ได้ได้รับสิทธิ์ในการติดต่อสื่อสาร ทำให้ขั้นตอนของการลงทะเบียน Home Agent ไม่สามารถทำได้ และถ้าเป็นระบบเครือข่ายแบบเคลื่อนที่ที่ใช้ IPv6 แล้วมีความเป็นไปได้สูงที่การติดต่อสื่อสารขณะกำลังเคลื่อนที่เกิดขึ้น ซึ่งถ้าลงทะเบียนกับ Home Agent ไม่ได้ ทำให้การสื่อสารหยุดชะงักไป การตรวจสอบว่าไม่มีอุปกรณ์ใดใช้ IPv6 address ซ้ำ นี้เรียกว่า Duplicate Address Detection (DAD)

จากการศึกษาขั้นตอนการเตรียมการหลังจาก MN มีการย้ายเครือข่ายนั้น จะเกิด Delay time จาก 4 ส่วนด้วยกันคือ (1) เวลาในการ Hand-off ที่ Data Link Layer (2) เวลาในการรอรับ Network prefix ของเครือข่ายใหม่ (3) เวลาในการตรวจสอบการซ้ำกันของ IPv6 address (เวลาในการทำ DAD) (4) เวลาในการลงทะเบียน Care-of Address ใหม่กับ Home Agent ในเครือข่ายเดิมหรือการทำ Binding Update

จากบทความ [7] พบว่าระยะเวลาในการตรวจสอบการซ้ำกันของ IPv6 address นั้นต้องใช้เวลาอย่างน้อย 1000 ms ซึ่งคิดเป็นกว่า 10% ของเวลาในการย้ายข้ามเครือข่ายทั้งหมดของ Mobile IPv6 ดังนั้นการลด Delay time ของการทำ DAD จะส่งผลโดยตรงต่อเวลาโดยรวมที่ MN ใช้ในการย้ายข้ามเครือข่าย ในปัจจุบันมีนักวิจัยได้ออกแบบเทคนิคใหม่ๆ เพื่อลด Delay time ของการทำ DADหลายวิธี ในบทความนี้จะเริ่มจากการอธิบายขั้นตอนการทำ DAD แบบมาตรฐานและกล่าวถึงกระบวนการตรวจสอบ DAD แบบใหม่ 4 วิธี ได้แก่ RFC4429 Optimistic DAD [1], Advanced DAD [2], Proactive DAD[3] และ RFC3315 DHCPv6 [4]

2. การทำงาน DAD แบบมาตรฐาน

การตรวจสอบการซ้ำกันของIPv6 address หรือ DAD แบบมาตรฐานอยู่ในขั้นตอนหนึ่งของการทำ Stateless Address Autoconfiguration (RFC2462) [5] โดยการใช้ข้อความ ICMPv6 ที่อยู่ในกลุ่มของ Neighbor Discovery for IPv6 (RFC2461) [6] ในการติดต่อสื่อสาร มีขั้นตอนการทำงานดังนี้

2.1 เมื่อ MN เข้าไปยังเครือข่ายใหม่จำเป็นต้องกำหนด Care-of Address สำหรับเครือข่ายใหม่ ในการกำหนด IPv6 address ใหม่นั้น MNต้องการ 2 ส่วนมาประกอบกัน

ส่วนที่หนึ่งคือ Prefix Information ความยาว 64 บิต ซึ่งได้มาจากการรอข้อความ Router Advertisement ที่มี Prefix Information Option ที่ส่งมาเป็นทุกช่วงเวลา หรือส่งมาเมื่อ MN ส่งข้อความ Router Solicitation ที่ Source address = ไม่ระบุ, Destination address = FF02::2 (All Router multicast group) ไปถาม Prefix Information กับ Default Router แล้ว Default Router จะส่งข้อความ Router Advertisement ที่มี Prefix Information Option กลับมา และส่วนที่สอง Interface ID ความยาว 64 บิต เป็นส่วนที่ MN คิดคำนวณขึ้นมาเองจากการสุ่มหรือใช้ MAC address (EUI64) หรือวิธีอื่นๆ เมื่อนำทั้งสองส่วนมาประกอบกันจะได้ IPv6 address ที่มีความยาว 128 บิต อย่างไรก็ตาม IPv6 address ที่ได้นี้ยังไม่สามารถใช้ส่งหรือรับข้อมูลได้ (เรียกว่า Tentative address)จนกว่าจะได้ทำการตรวจสอบว่าIPv6 address นี้ไม่ซ้ำกับอุปกรณ์อื่นในเครือข่าย

2.2 หลังจากที่ MN ได้ Tentative address จะเริ่มขั้นตอนการตรวจสอบ DAD MN จะส่งข้อความ ICMPv6 ชนิด Neighbor Solicitation ทีมีค่าพารามิเตอร์เหล่านี้ Source address = ไม่ระบุ, Destination address = FF02::1 (All Node muticast group) Target Address = Tentative address ไปยังทุกอุปกรณ์ในเครือข่าย เป็นการถามอุปกรณ์ต่างๆ ว่ามีใครใช้ address เดียวกันนี้หรือไม่

2.3 MN รอการตอบกลับของข้อความ Neighbor Advertisement โดยถ้ามีการตอบกลับจะแสดงว่ามีอุปกรณ์ที่ใช้ IPv6 address ที่ตรงกับ Tentative address อยู่แล้ว ระยะเวลาในการรอแบบมาตรฐานคือ 1000 ms หากพ้นระยะเวลานี้แล้ว MN ไม่ได้รับข้อความ Neighbor Advertisement มันจะตีความว่า Tentative address นี้ยังไม่มีใครใช้ (Unique) และสามารถใช้เป็น Care-of Address ในเครือข่ายนี้ได้เลย เป็นการจบขั้นตอนการตรวจสอบ DAD อย่างไรก็ตาม หาก MN ได้รับข้อความ Neighbor Advertisement มันจะต้องกำหนด Tentative address ขึ้นมาใหม่และย้อนกลับไปทำขั้นตอนที่ 2.2 ใหม่

จากขั้นตอนทั้งหมดจะเห็นได้ว่าการตรวจสอบ DADนั้นจะต้องรอเวลาอย่างน้อยที่สุดคือ 1000 ms และในเวลา 1000 ms เป็นช่วงเวลาที่ไม่สามารถติดต่อสื่อสารได้ เนื่องจากเป็น Tentative address ทำให้ถ้า MN มีการส่งข้อมูลอยู่ แล้วเคลื่อนที่มาในเครือข่ายใหม่จะต้องหยุดการติดต่ออย่างน้อยที่สุดคือ 1000 ms ซึ่งถ้าเป็น Real-time application จะมีผลกระทบที่สามารถสังเกตได้ทันที แสดงให้เห็นว่าการรอการตรวจสอบ DAD นั้นนานเกินไปสำหรับ application บางประเภท การแก้ปัญหาในเรื่องนี้มีแนวคิดอยู่ด้วยกันหลายวิธีไม่ว่าจะเป็นการเพิ่มตัวกลางในการจัดการ IPv6 address แบบต่างๆ การพยายามปรับให้ช่วงที่เป็น Tentative address

สามารถส่งหรือรับข้อความที่ใช้สำหรับติดต่อสื่อสารข้อมูลได้ หรือการปรับโครงสร้างเครือข่ายและเพิ่มโพรโตคอลใหม่ให้เอื้ออำนวยกับการเตรียม Care-of Address มีรายละเอียดการทำงานดังต่อไปนี้

3. วิธีการพัฒนากระบวนการ DAD ที่ผ่านมา

เนื่องจากขั้นตอนการตรวจสอบ IPv6 address ใหม่ นั้นต้องใช้เวลาอย่างน้อย1000 ms ซึ่งคิดเป็นกว่า 10% ของเวลาในการย้ายข้ามเครือข่ายทั้งหมดของ Mobile IPv6 [7] การปรับปรุงกระบวนการ DAD ให้มีการทำงานที่เร็วขึ้นนั้นได้มีผลงานวิจัยออกมาอยู่หลายวิธีด้วยกัน ได้แก่

3.1 Optimistic DAD (RFC4429)

วิธีนี้ใช้หลักการที่คาดว่าในระบบเครือข่ายที่ใช้ IPv6 address จะมีความน่าจะเป็นที่การซ้ำกันของ IPv6 address มีค่าน้อยมากตามหลักความน่าจะเป็นของ Birthday Paradox [1] จากแนวคิดนี้ทำให้สรุปได้ว่ามีความเป็นไปได้ที่ Tentative address จะยังไม่มีใครนำไปใช้ จึงกำหนดให้ Tentative address หรือในวิธีนี้จะเรียกว่า Optimistic address สามารถที่จะส่งหรือรับข้อความที่ใช้สำหรับติดต่อสื่อสารข้อมูลได้บางชนิด โดยใช้แนวคิดของการปรับมาตรฐาน RFC2461 และ RFC2462 บางข้อกำหนด และเพิ่ม Optimistic flag เข้าไปเพื่อใช้สำหรับ Optimistic address แนวคิดในการปรับมาตรฐานคือจะไม่ต้องการให้ Optimistic address ส่ง Link Layer address (MAC address) ของตนไปกับข้อความทุกข้อความที่ส่งออกเพื่อป้องกันการจับคู่ IPv6 address กับ Link Layer address ที่ไม่ถูกต้องใน Neighbor cache entries ของอุปกรณ์อื่นๆ ในกรณีที่ Optimistic address นี้มีอุปกรณ์ใช้ไปแล้ว ดังนี้

3.1.1 ขณะที่ใช้ Optimistic address จะหลีกเลี่ยงไม่ให้ Neighbor cache entries ของ neighbor ต่างๆ เก็บ Optimistic address เพื่อหลีกเลี่ยงการชนกัน ถ้าเป็น address ที่ซ้ำ ซึ่งอาจจะไปทับใน Neighbor cache entries ใดๆ โดย

4) ใช้ Override flag = 0 ในข้อความ Neighbor Advertisement เป็นการบอกให้เขียน Optimistic address ทับใน Neighbor cache entries ถ้ามีการจับคู่ที่ไม่เหมือนกัน
5) จะไม่ส่ง ข้อความ Neighbor Solicitation จาก Optimistic address เพราะปกติจะมีการใส่ Source Link Layer Address Option อาจมีการใส่ใน Neighbor cache entries แต่ถ้าเป็น Neighbor Solicitation ในการถาม DAD ต้องไม่ระบุ IPv6 address ผู้ส่ง
6) ไม่ส่งข้อความ Router Solicitation ที่มี Source Link Layer Address Option ถ้าต้องการส่งต้องไม่มี Source Link Layer Address Option

3.1.2 การปรับในข้อกำหนด RFC2461 Neighbor Discovery for IPv6

7) ในการใช้ข้อความ Router Solicitation ถาม Prefix Information ควรส่งจาก MN ที่ไม่ได้ใช้ Optimistic address หรือ จาก Optimistic address ซึ่งต้องไม่ใส่ Source Link Layer Address Option
8) ใน Address Resolution (AR) ต้องไม่ใช้ Optimistic address เป็น Source address ของข้อความ Neighbor Solicitation และถ้าการส่งข้อความ Router Solicitation ที่ไม่มี Source Link Layer Address Option ทำให้การตอบข้อความ Router Advertisement ไม่สามารถทำได้ให้รอจนการตรวจสอบ DAD เสร็จก่อนจึงสามารถทำการติดต่อ router ได้
9) กรณีที่ต้องการส่ง packet ไปให้ neighbor ขณะที่ใช้ Optimistic address ห้ามถาม Access Router ให้ใช้ข้อความ Redirect ที่มี Target Link Layer Address Option ไปที่ default router

3.1.3 การปรับในข้อกำหนด RFC2462 IPv6 Stateless Address Autoconfiguration

10) ในการสร้าง Global และ Site-Local address ไม่ควรนำ Optimistic address มาเป็น Address ใหม่ถ้ายังไม่รู้ Link Layer Address ของ router
11) ขณะที่เป็น Optimistic address ต้องใช้ Optimistic flag ในการใช้งานก่อน และถ้า DAD Process เสร็จแล้วให้เปลี่ยนสถานะเป็น Preferred หรือ Deprecated address ตามแบบมาตรฐานปกติ
12) Router ไม่ตอบข้อความ Router Solicitation จากข้อความที่ไม่ระบุ Source address แต่จะตอบกับข้อความที่มากจาก Optimistic address ที่ใช้ Unicast address โดยการใช้ Override flag ร่วมด้วย (O=0)

จากการปรับเปลี่ยนข้อกำหนดจะทำให้ได้ขั้นตอนการตรวจสอบที่เปลี่ยนไปดังนี้

(1) เมื่อ MN เข้ามาในเครือข่ายใหม่ให้รอรอบของข้อความ Router Advertisement ที่มีการบอก Prefix Information แล้วทำการกำหนด Optimistic address ที่มีการรวม Prefix Information แล้ว
(2) MN ส่งข้อความ Neighbor Solicitation ที่ไม่มี Source Link Layer Address Option และไม่ระบุปลายทางทันที เป็นการตรวจสอบ DAD พร้อมทั้งใช้งานอื่นๆได้เลย โดยมีข้อกำหนดการส่งข้อความ ตามข้างต้น
(3) เมื่อถึงเวลา 1000 ms แล้วยังไม่มีใครตอบกลับ Optimistic address จะ

เปลี่ยนสถานะไปเป็น Preferred address ตาม RFC 2462

จากแนวคิดนี้ ข้อดีที่เห็นได้คือการปรับเปลี่ยนมาตรฐาน ซึ่งสามารถทำงานกับ DAD แบบมาตรฐานได้ ผลที่ได้ดีขึ้นโดยไม่มีการเพิ่มอุปกรณ์ ยังคงเป็นการทำงานที่ไม่ต้องมีศูนย์กลาง อย่างไรก็ตาม การส่งและรับข้อความยังไม่สามารถทำงานได้อย่างเต็มที่ขณะที่เป็น Optimistic address

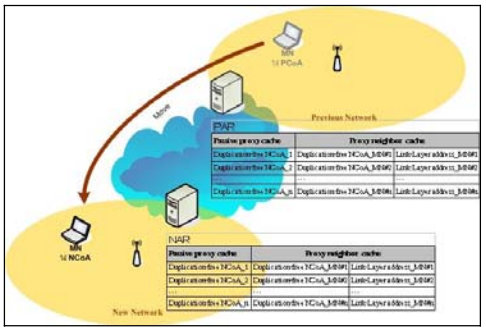
3.2 Advanced DAD (A-DAD)

ใช้แนวคิดของการเก็บ IPv6 address ที่ได้รับการทดสอบ DAD ไว้แล้วจำนวนหนึ่ง (กำหนดไว้ที่ 100 addresses) ไว้ที่ Access router ของแต่ละเครือข่าย และใช้การทำนายการเคลื่อนที่ของ MN ด้วยวิธี Fast Handover [8] เพื่อเตรียม IPv6 address ไว้ล่วงหน้า

ในด้านของการออกแบบนั้น ให้ Care-of Address ของ MN ในเครือข่ายใหม่ เรียกว่า NCoA (New Care-of Address) และในเครือข่ายก่อนหน้านี้เรียก PCoA (Previous Care-of Address) โดย NCoA ที่ผ่านการตรวจสอบ DAD แล้วจะเรียกว่า Duplication-free NCoA ซึ่งจะอยู่ใน Passive proxy cache และ Access router ในเครือข่ายใหม่เรียกว่า NAR (New Access Router) และที่เครือข่ายก่อนหน้าเรียกว่า PAR (Previous Access Router) มีการใช้ส่วนเพิ่มเติมและโพรโตคอลใหม่ ดังนี้

3.2.1 มีการเพิ่ม Passive proxy cache เพื่อเก็บ Duplication-free NCoA ที่อยู่ใน Access router มีข้อกำหนดดังนี้

13) Passive proxy cache ต้องไม่ตอบข้อความ Neighbor Solicitation ที่มี Target address เป็น IPv6 address ที่อยู่ใน Passive proxy cache และลบ IPv6 address นั้นออกจาก Passive proxy cache เลยเพราะแสดงว่า IPv6 address นั้นมีการใช้งานอยู่หรือกำลังถูกตรวจสอบ
14) Passive proxy cache ต้องไม่ใช้ Duplication-free NCoA ใน Passive proxy cache ส่ง multicast ในข้อความ unsolicited Neighbor Advertisement
15) การหา IPv6 address ของ Passive proxy cache จะใช้วิธีสุ่มเอาแล้วใช้การตรวจสอบ DAD แบบมาตรฐาน
16) Passive proxy cache ต้องไม่มีผลกระทบกับ Neighbor cache ของ Neighbor node เพราะ Duplication-free NCoA ที่ได้ยังไม่มีการใช้งาน IPv6 Address



รูปที่ 2 แสดงโครงสร้างเครือข่ายของวิธี A-DAD

3.2.2 มีการเพิ่ม Proxy neighbor cache เป็น cache ที่เก็บ Duplication-free NCoA กับ link layer address ของ MN ที่จับคู่กับหลังจากมีการขอ IPv6 address จาก Passive proxy cache แล้ว

3.2.3 มีการใช้โพรโตคอลใหม่ 2 ตัวในการขอ IPv6 Address จาก Passive proxy cache คือ

17) Duplication-free NCoA Request Option เป็นข้อความ ICMPv6 ที่ใช้ร้องขอ NCoA ใหม่ที่ได้ทดสอบว่าไม่มีการซ้ำกันของ Address แล้วมีโครงสร้างดังรูปที่ 2 ซึ่ง ‘B’ flag คือ Buffer packets flag จะใช้เมื่อ MN ไม่ได้อยู่ในเครือข่ายของ NAR ซึ่งหมายถึงว่าสามารถทำนายได้ว่า MN จะย้ายมา Network ที่ NAR

18) Duplication-free NCoA Reply Option ข้อความ ICMPv6 ที่ใช้ตอบการขอ NCoA ใหม่ที่ได้ทดสอบว่าไม่มีการซ้ำกันของ Address แล้วมีโครงสร้างดังรูปที่ x ซึ่ง ‘B’ flag จะใช้คู่กับ request message ที่มี ‘B’ flag คู่กันและ Identifer จะเป็นรหัสที่ใช้ตรงกันของคู่ข้อความที่ร้องขอเพื่อระบุว่าเป็นข้อความถาม-ตอบที่เป็นคู่กันที่ถูกต้อง

ขั้นตอนการทำงานของ A-DAD จะแบ่งได้เป็น 2 กรณีคือ

กรณีที่ทำนายการเคลื่อนที่ของ MN ได้ (Predictive case) มีขั้นตอนดังนี้

(4) เมื่อทราบว่า MN จะเคลื่อนที่เครือข่ายไหนแล้วก็ทำการส่ง Duplication-free NCoA Request Option message ที่ใช้ ‘B’ flag ผ่านทาง PAR ไปยัง NAR

(5) เมื่อ NAR ได้รับข้อความจะเลือก Duplication-free NCoA จาก Passive proxy cache 1 ตัวไปใส่กับ Duplication-free NCoA Reply Option message ที่ใช้ ‘B’ flag และลบ address นั้นออกจาก Passive proxy cache แล้วไปใส่ใน Proxy neighbor cache ที่ใช้ IP address นั้นกับ Link layer address ของ MN แล้วทำการส่ง Duplication-free NCoA Reply Option message ที่ใช้ ‘B’ flag กลับไปให้ MN ผ่านทาง PAR

(6) เมื่อ MN ได้รับ จะเริ่มเคลื่อนย้ายไปที่เครือข่ายใหม่จะติดต่อกับ NAR แล้ว NAR จะย้ายข้อมูลของ Proxy neighbor cache ไปยัง Neighbor cache พร้อมกำหนดสถานะเป็น REACHABLE

(7) เริ่มทำ Binding update ได้

กรณีที่ไม่สามารถทำนายการเคลื่อนที่ของ MN ได้ (Non-predictive case) มีขั้นตอนดังนี้

(1) เมื่อ MN มาถึงเครือข่ายใหม่จะทำการติดต่อกับ NAR เพื่อขอ Duplication-free NCoA ทันทีโดยการส่งข้อความ Duplication-free NCoA Request Option ไปที่ NAR

(2) เมื่อ NAR ได้รับข้อความจะเลือก Duplication-free NCoA จาก Passive proxy cache 1 ตัวไปใส่กับข้อความ Duplication-free NCoA Reply Option และลบ Duplication-free NCoA นั้นออกจาก Passive proxy cache แล้วไปใส่ใน Neighbor cache ที่ใช้ Duplication-free NCoA นั้นกับ Link layer address ของ MN พร้อมกำหนดสถานะเป็น REACHABLE แล้วทำการส่งข้อความ Duplication-free NCoA Reply Option

(3) เมื่อ MN ได้รับ NCoA เริ่มทำ Binding Update ได้

จากกระบวนการทำงานของ A-DAD จะเห็นได้ว่าไม่ต้องรอการตรวจสอบ DAD 1000 ms เพราะมีการตรวจสอบ DAD และเก็บ IPv6 address ล่วงหน้าจำนวนหนึ่งแล้ว ทำให้สามารถทำขั้นตอน Binding update ได้เร็วขึ้น โอกาสข้อมูลสูญหายน้อยลง อย่างไรก็ตามอุปกรณ์ MN ต้องรู้จักโพรโตคอลใหม่ถึงจะทำงานได้ และเมื่อสังเกตการทำงานของ Access router นอกจากจะทำหน้าที่สร้างทางเชื่อมต่อแล้ว ยังต้องทำการตรวจสอบ DAD และรับ-ส่งข้อความของ A-DAD ซึ่งเป็นการเพิ่มภาระหน้าที่ของ Access router ถ้า Access router ไม่สามารถทำงานได้ A-DAD ก็ไม่สามารถใช้ได้เช่นกัน

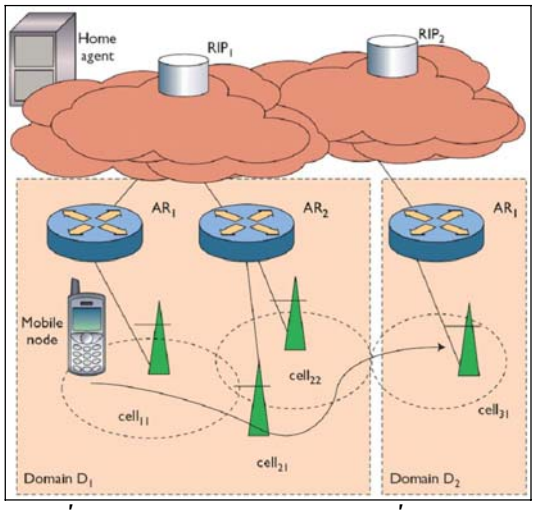
3.3 Proactive DAD (P-DAD)

วิธีนี้มีแนวคิดที่จะเตรียมทำการตรวจสอบก่อนถึงเครือข่ายใหม่โดยไม่มีการเก็บ IPv6 address จำนวนหนึ่งเหมือนกับ A-DAD ซึ่งต้องใช้การทำนายการเคลื่อนที่ของ MN ด้วยวิธีตาม [9] เพื่อเป็นข้อมูลในการส่งข้อความไปตรวจสอบ DAD ล่วงหน้าที่เครือข่ายที่จะเคลื่อนย้ายไป มีการเพิ่มอุปกรณ์ที่เรียกว่า Regional Information Point (RIP) server อยู่เหนือ Access router และ Access point ใน domain เดียวกันดังรูปที่ 3 RIP server นี้จะเก็บ Mobile-node Attachment point (MAP) table เป็นตารางที่มีการเก็บค่า 3 ค่าใน 1 entry ตามตารางที่ 1 คือ

(1) access point’s basic service set ID (BSSID)

(2) access router IP address ที่ใช้ BSSID นั้น

(3) Prefix information ของ access router นั้น



รูปที่ 3 แสดงโครงสร้างเครือข่ายที่ใช้ใน P-DAD

AP's BSSID	AR's IP address	Advertised prefix
(AP_{11}) 00-01-4A-C3-07-01	(AR_1) 2001:0E10:6440:0001::1	$Prefix_1$
(AP_{21}) 00-01-4A-C3-07-02	(AR_2) 2001:0E10:6440:0002::1	$Prefix_2$
(AP_{22}) 00-01-4A-C3-07-03	(AR_2) 2001:0E10:6440:0002::1	$Prefix_2$
(AP_{31}) 00-01-4A-C3-07-04	(AR_3) 2001:0E10:6440:0003::1	$Prefix_3$

ตารางที่ 1 แสดง Mobile-node Attachment point (MAP) table

ได้มีการใช้ข้อความใหม่สำหรับติดต่อกัน เพื่อการตรวจสอบล่วงหน้านี้มี 4 ข้อความคือ

(1) ข้อความ CoA-preAllocate Request ใช้สำหรับตรวจสอบ Care-of Address ของเครือข่ายใหม่ขณะที่ยังไม่ย้ายมาโดยมีพารามิเตอร์ addr เป็น Care-of Address ของเครือข่ายใหม่ที่ต้องการตรวจสอบ

(2) ข้อความ CoA-preAllocate Reply ใช้ตอบข้อความ CoA-preAllocate Request โดยมี U bit บอกว่า Care-of Address ที่ตรวจสอบยังไม่มีการใช้งาน

(3) ข้อความ CoA_activation Request ใช้เมื่อย้ายไปเครือข่ายใหม่และต้องการใช้ Care-of Address ที่ได้ตรวจสอบแล้ว ต้องบอกกับ Access router ของเครือข่ายใหม่

(4) ข้อความ CoA_activation Replyใช้ตอบข้อความ CoA_activation Request message

จากโครงสร้างทั้งหมดนำมาเป็นขั้นตอนของ P-DAD ได้ดังนี้

3.3.1 ก่อนที่จะทำการย้ายไปยัง Access point ใหม่ MN เอา prefix ของ Access router ใหม่จาก MAP table ของ RIP Server และคิดคำนวณ tentative Care-of Address แล้วจึงทำการส่งข้อความ CoA-preAllocate Request กับพารามิเตอร์ addr สำหรับการตรวจสอบ DAD

3.3.2 เมื่อ Access router ใหม่ได้รับข้อความ CoA-preAllocate Request จะตรวจสอบข้อมูลใน Registration cache ซึ่งเป็น cache ที่เก็บข้อมูล 3 อย่างคือ Home address, Pre-allocated new Care-of Address และ CoA's lifetime value และถ้าจำเป็นอาจจะใช้การตรวจสอบ DAD แบบมาตรฐานตรวจสอบเพิ่มเติม

หลังจากที่ตรวจสอบแล้วว่า Care-of Address ที่มาจากพารามิเตอร์ addr นี้

ไม่มีใครใช้มาก่อนจะส่งข้อความ CoA_preAllocate Reply ที่มี U bit

3.3.3 เมื่อ MN ได้รับข้อความ CoA_preAllocate Reply จะเก็บ Care-of Address นั้นไว้ที่ Registration cache ของตนและส่งข้อความ CoA_activation Request ถึง Access router ใหม่และ Access router ใหม่จะเก็บ Care-of Address นี้ลงไปใน Registration cache แต่ถ้าไม่ได้รับ U bit จะต้องทำการตรวจสอบ DAD แบบมาตรฐานและข้ามไปข้อ 5

3.3.4 ในกรณีที่ได้รับ U bit เมื่อไปถึง Access router ใหม่จะรู้การมาถึงของ MN และส่ง CoA_activation Reply message และลบ Care-of Address นี้ออกจาก Registration cache แต่ถ้า MN ไม่ได้รับ CoA_activation Reply message ให้ทำการตรวจสอบ DAD แบบมาตรฐาน

3.3.5 MN เริ่มทำ Binding Update กับ Home Agent

เห็นได้ว่าการทำงานของ P-DAD สามารถแบ่งได้เป็นสองกรณีคือ กรณีที่หนึ่งคือกรณีที่สามารถทำนายการเคลื่อนย้ายของ MN ได้จะสามารถตรวจสอบ DAD ได้ล่วงหน้า เมื่อ MN มาถึงที่เครือข่ายใหม่สามารถใช้ Care-of Address ทำ Binding Update ได้เลย ซึ่งการที่สามารถทำให้การทำนายได้ผลจะต้องมี RIP Server เป็น Hardware ที่เพิ่มเติมขึ้นมาในกลุ่มเครือข่ายใดๆ และข้อความการติดต่อใหม่เพื่อการตรวจสอบ DAD ล่วงหน้า รวมถึงวิธีการทำนายการเคลื่อนที่ของ [9] สังเกตได้ว่าการใช้วิธีการทำนายใช้ได้เฉพาะส่วนของเครือข่ายไร้สายตามมาตรฐาน IEEE 802.11 เท่านั้น กรณีที่สองคือกรณีที่ไม่สามารถทำนายได้หรือทำนายผิดพลาดจะทำงานการตรวจสอบ DAD แบบมาตรฐาน ซึ่งเสียเวลาการตรวจสอบประมาณ 1000 ms

3.4 DHCPv6 (RFC3315)

Dynamic Host Configuration Protocol (DHCP) สำหรับ IPv6 นั้นมีลักษณะ ขั้นตอนการทำงาน และแนวคิดเหมือนกับ DHCP ที่ใช้กับ IPv4 นอกจากการพัฒนาและปรับเปลี่ยนให้เข้ากับการใช้งานระบบ IPv6 แล้วการเรียกชื่ออุปกรณ์ในระบบยังเหมือนเดิมคือมี DHCP Server คอยจัดสรร IPv6 address ให้ DHCP Client โดย DHCP Server จะต้องจำสถานะการจัดสรร Address เพื่อป้องกันการจ่าย IPv6 address ซ้ำ ในส่วนของโพรโตคอลที่มีการปรับให้เหมาะสมกับระบบ IPv6 ได้แก่ ใช้ Multicast address All_DHCP_Relay_Agents_and_Servers (FF02::1:2) ในข้อความ SOLICIT ที่ DHCP Client ถาม DHCP Server แทนที่ใช้ Boardcast address (255.255.255.255) กำหนด UDP Port ใหม่ได้แก่ UDP port 546 เป็นช่องทาง DHCP Client รับข้อความและ

UDP port 547 เป็นช่องทาง DHCP Server รับข้อความ และจากที่มีการใช้ MAC address ติดต่อสื่อสารกันก็เปลี่ยนมาเป็น DHCP Unique Identifier (DUID) ภายในจะมีชนิดของโครงสร้างข้อความ 3 ชนิดคือ DUID Based on Link-layer Plus Time (DUID-LIT), DUID Assigned by Vendor Based on Enterprise Number (DUID-EN) และ DUID Based on Link-layer (DUID-LL) มี Identifier Association ID (IAID) ใช้สำหรับระบุตัวตนของ DHCP Client ที่จะมาติดต่อ DHCP Server นั้นๆ ซึ่งจะไม่ใช้ซ้ำกันโดย DHCP Client เป็นคนสร้างขึ้นมาเอง

ข้อความหลักๆในการรับ-ส่งขอ IPv6 address มีการตอบรับทั้งหมด 4 ข้อความ ซึ่งจะอธิบายไปพร้อมกับขั้นตอนการทำงาน ดังนี้

3.4.1เมื่อ MN เข้ามาที่เครือข่ายใหม่ จะได้รับข้อความให้รู้ว่ามีการใช้ Stateful Address Autoconfiguration โดย Neighbor Discovery ทำให้ทราบได้ว่ามีการใช้ DHCPv6 อยู่ จากนั้น MN จะส่งข้อความถามหา DHCP Server ด้วยข้อความ SOLICIT ที่มี การสุ่มเลข transaction-id ที่ใช้สำหรับการระบุคู่ข้อความใดๆ และใส่ option เป็น Client Identifier option เป็นการบอก DUID ของ DHCP Client ส่งไปที่ All_DHCP_Relay_Agents_and_ Servers (FF02::1:2)

3.4.2เมื่อ DHCP Server ได้รับข้อความ SOLICIT จะตอบกลับด้วยข้อความ ADVERTISE ที่มี transaction-id เดียวกันกับข้อความ SOLICIT และใส่ option เป็น Server Identifier option กับ Client Identifier option เป็นการบอก DUID ของ DHCP Client และใช้ Source address เป็น DHCP Server address

3.4.3เมื่อ MN หรือ DHCP Client ได้รับข้อความ ADVERTISE จะส่งข้อความ REQUEST ถึง DHCP Server โดยสุ่มเลข transaction-id ใหม่ และมีการใส่ option คือ Server Identifier option กับ Client Identifier option และ Option request option ที่ภายในจะระบุ option-code ที่ต้องการให้ DHCP Server ส่งมาซึ่งเป็น configuration information ต่างๆ มีการใช้ Identifier Association for Non-temporary Address Option (IA_NA) หรือ Identifier Association for Temporary Address Option (IA_TA) แล้วแต่การจัดการของระบบ DHCP ซึ่งภายในจะมี IAID และใน IA_NA หรือ IA_TA ส่งไปที่ All_DHCP

_Relay_Agents_and_Servers (FF02::1:2)

3.4.4เมื่อ DHCP Server ได้รับข้อความ REQUEST จะตรวจสอบ IAID และเก็บไว้เป็นข้อมูลว่า MN นี้ขึ้นกับ DHCP Server ตัวที่รับข้อความ และจะส่งข้อความ REPLY ตอบไปยัง DHCP Client ที่ถามมาโดยมี IA_NA หรือ IA_TA ที่มี option เป็น IA Address option เป็น option ใน IA_NA หรือ IA_TA ที่มี IPv6 Address อยู่ภายใน และใส่ option อื่นตามที่ Option request option ขอมา รวมทั้ง Server Identifier option กับ Client Identifier option ใช้ Source address เป็น DHCP Server address

สังเกตได้ว่า การใช้งาน DHCPv6 นั้น อุปกรณ์ทุกตัวที่ใช้ระบบนี้ต้องรู้จักโพรโตคอลของ DHCPv6 และการทำงานแบบ Server-Client เป็นการทำงานเป็นมีศูนย์กลางโดยมี DHCP Server เป็นศูนย์กลาง DHCP และทำให้ DHCPv6 เป็น Stateful Address Autoconfiguration คือมีการใช้สถานะในการจดจำขั้นตอนต่างๆ ปัญหาที่พบใน DHCPv6 คือ ขั้นตอนการเรียกใช้ IPv6 address ซ้ำอีกครั้ง (Renumbering) ทำได้ยาก และถ้ามี IPv6 address จำนวนมาก การจำ IPv6 address ที่มีการใช้งานอยู่หรือไม่ได้ใช้งานเป็นไปได้ยาก ในเรื่องของการกำหนดขอบเขตของ IPv6 address (Address range) ที่ต้องการจ่ายให้พอดีกับความต้องการของผู้ใช้ซึ่งถ้ามากเกินไปก็จะจัดการได้ยาก ถ้าน้อยเกินไปก็จะใช้ขั้นตอนใช้ IPv6 address ซ้ำอีกครั้ง (Renumbering) บ่อยขึ้น และถ้า DHCP Server ไม่สามารถทำงานได้ DHCP Client จะไม่สามารถแลกเปลี่ยนข้อมูลที่จำเป็นสำหรับการสื่อสารได้เลย แต่ข้อดีของ DHCPv6 คือสามารถแจกจ่าย parameters ต่างๆ ได้ง่าย เช่น DNS parameters, Routing parameter ซึ่งวิธี Stateless Address Autoconfiguration ไม่สามารถส่ง parameters ดังกล่าวได้

4. เปรียบเทียบวิธีการพัฒนา DAD ต่างๆ

จากหลายวิธีของการตรวจสอบ DAD ที่ได้กล่าวมานั้นเราสามารถวิเคราะห์เปรียบเทียบคุณลักษณะที่แตกต่างกันดังตารางที่ 2

ในส่วนของ Hardware มีเพียง P-DAD และ DHCPv6 ที่ต้องการ Hardware เพิ่มเติม โดย P-DAD เพิ่ม RIP Server ไว้เหนือ Access router และ DHCPv6 เพิ่ม DHCP Server ทำให้โครงสร้างเครือข่าย (Network Topology) ของทั้งสองเปลี่ยนแปลง และเมื่อมาดูในส่วนของ Software และโพรโตคอล Optimistic DAD ใช้วิธีปรับเปลี่ยนข้อกำหนดจากการตรวจสอบ DAD แบบมาตรฐานทำให้มีการปรับ software โดยใช้ Optimistic flag เข้ามาระบุขณะใช้ Optimistic address

แต่ใช้โพรโตคอลเดิม ส่วน A-DAD เพิ่ม Passive Proxy Cache กับ Proxy Neighbor Cache เข้าไปใน Access router และใช้โพรโตคอลใหม่โดยมี 2 ข้อความใหม่คือ Duplication-free NCoA Request Option กับ Duplication-free NCoA Reply Option และใช้ B Flag (Buffer packet flag) P-DAD มีการเพิ่ม software เข้าไป 2 ส่วนคือ Mobile-node Attachment point (MAP) table ที่ RIP Server กับ Registration cache ที่ Access router และใช้โพรโตคอลใหม่ได้แก่ ข้อความ CoA_preAllocate Request, CoA_preAllocate Reply, CoA_activation Request, CoA_activation Reply และ addr parameter, U Flag (Unique flag) DHCPv6 มีการเพิ่ม software เพื่อการใช้โพรโตคอล DHCPv6

ในแง่ของการจำสถานะในการจัดสรร IPv6 address A-DAD กับ DHCPv6 เป็น Stateful Address Autoconfiguration เพราะขั้นตอนการจ่าย Care-of Address เป็นขั้นตอนที่มีการใช้สถานะ ทำให้เกิดระบบที่มีศูนย์กลาง (Centralized) ซึ่งเป็นระบบที่มีอุปกรณ์กลางค่อยจัดการระบบ มี A-DAD กับ P-DAD ที่ใช้การทำนายจาก Layer-2 Handover ซึ่งใช้ความแรงของสัญญาณคลื่นความถี่เป็นตัววิเคราะห์ มีผลทำให้เครือข่ายที่ใช้สายสัญญาณเป็นตัวนำ (Wired network) ไม่สามารถใช้งานได้เต็มประสิทธิภาพตามแนวคิดโดย P-DAD กำหนดให้การทำนาย Layer-2 Handover ใช้ได้ในเครือข่าย IEEE 802.11 เท่านั้นตามวิธีการทำนายของ [9] และ A-DAD, P-DAD และ DHCPv6 มีอุปกรณ์ที่ทำงานมากกว่าอุปกรณ์อื่น (Workload) คือ Access Router, Access Router และ DHCP Server ตามลำดับซึ่งไม่มีอุปกรณ์ใดในวิธี Optimistic DAD ที่ทำงานมากกว่าอุปกรณ์อื่น

เมื่อดูปีที่ตีพิมพ์หลังจากที่ DAD แบบมาตรฐานออกมา ได้มี DHCPv6 ออกตามมาในปี 2003 และ A-DAD ได้ตีพิมพ์ออกมาในเวลาใกล้ๆ กัน ในปี 2006 Optimistic DAD ได้ตีพิมพ์ออกมาตามมาด้วย P-DAD ได้ออกมาในช่วงปลายปีเดียวกัน

5. สรุปและแนวทางการวิจัยในอนาคต

บทความนี้นำเสนองานวิจัยที่พยายามปรับปรุงขั้นตอนการตรวจสอบการซ้ำกันของ IPv6 Address ในเครือข่าย 5 วิธี คือ DAD แบบมาตรฐาน Optimistic DAD A-DAD P-DAD และ DHCPv6 ผลการวิเคราะห์เปรียบเทียบแสดงให้เห็นความแตกต่างหลายๆ ด้าน เช่น โครงสร้างเครือข่าย แนวคิดการทำงาน การรับ-ส่งข้อความ อย่างไรก็ตามทุกวิธีล้วนต้องการลดเวลาการตรวจสอบ DAD ทั้งสิ้น การนำมาซึ่งผลลัพธ์ดังกล่าวจำเป็นต้องมีการเพิ่มขั้นตอน วิธีการหรืออุปกรณ์ ทั้งนี้ต้องขึ้นอยู่กับโครงสร้างเครือข่ายและลักษณะการนำไปใช้งานของแต่ละระบบ

การทดสอบประสิทธิภาพความสามารถความเร็วในการได้รับ Care-of Address ที่มีสภาพแวดล้อมเดียวกัน และข้อจำกัดของแต่ละวิธีในเรื่องของขนาดของจำนวน IPv6 address ที่รับได้หรือผลของความคับคั่งที่จะมีผลต่อการตรวจสอบ Care-of Address ยังเป็นคำถามที่น่าสนใจในงานวิจัยด้านนี้ทั้งสิ้น รวมทั้งการคิดค้นวิธีใหม่ที่มีประสิทธิภาพดีให้เป็นมาตรฐานที่สามารถรองรับ Application ที่ต้องการความต่อเนื่องของการติดต่อสื่อสารต่อไป ซึ่งผู้เขียนหวังว่าบทความฉบับนี้จะมีประโยชน์ เป็นแนวทางความรู้ สำหรับนักวิชาการและผู้สนใจในด้านนี้ต่อไป

เอกสารอ้างอิง

[1] N. Moore, “Optimistic Duplicate Address Detection for IPv6,” IETF RFC 4429, April 2006.

[2] Y. Han, J. Choi and H. Jang, "Advance Duplicate Address Detection," Dec 2003.

[3] Chien-Chao Tseng, Yung-Chang Wong, Li-Hsing Yen and Kai-Cheng Hsu, “Proactive DAD: A Fast Address-Acquisition Strategy for Mobile IPv6 Networks,” Dec 2006.

[4] R. Droms, ed., “Dynamic Host Configuration Protocol for IPv6 (DHCPv6),” IETF RFC 3315, July 2003.

[5] S. Thomson and T. Narten, “IPv6 Stateless Address Autoconfiguration,” IETF RFC 2462, Dec 1998.

[6] T. Narten, E. Nordmark, and W. Simpson, “Neighbor Discovery for IP version 6 (IPv6),” IETF RFC 2461, Dec 1998.

[7] Adisak Busaranun, Panita Pongpaibool and Pichaya Supanakoon, “Handover Performance of Mobile IPv6 on Linux Testbed,” ECTI-CON 2006.

[8] R. Koodli, “Fast Handovers for Mobile IPv6,” IETF RFC 4429, July 2005.

[9] C.C. Tseng et al., “Location-Based Fast Handoff for 802.11 Networks,” IEEE Comm. Letters, vol. 9, no. 4, 2005, pp. 304–306.

Fast Duplicate Address Detection for Mobile IPv6

Panita Pongpaibool[1], Pahol Sotthivirat[2], Sukumal I. Kitisin[2], Chavalit Srisathapornphat[2]

[1] NECTEC, 112 Pahol Yothin Rd., Klong Luang, Pathumthani 12120 THAILAND,
panita@nectec.or.th
[2] Kasetsart University, 50 Pahol Yothin Rd., Chatuchak, Bangkok 10900 THAILAND

***Abstract*-Several components contribute to handover delay of Mobile IPv6, namely, movement detection time, address configuration time, binding registration time, and route optimization time. Through testbed experiments, we found that the dominating delay component is the address configuration time, which is the time required to configure a new globally routable IPv6 address on the mobile node after it moves to a foreign network. A typical process of IPv6 address configuration requires a duplicate address detection (DAD) procedure. During DAD, a mobile node sends a neighbor solicitation message to ask whether its new address is being used. If no node replies within a set timer, a mobile node can assume the new address is unique on that network. On the Linux testbed, DAD takes as much as 1 second, which causes intolerable disruption in case of time-critical applications. This work studies techniques for reducing delay during the DAD procedure. Main contributions of this work are the insights into benefits and shortcomings of different DAD techniques, as well as benchmarks for design of future DAD protocols.**

I. INTRODUCTION

The next generation network is expected to be dominated by mobile devices. Mobile IPv6 [1] has been proposed to manage movement of mobile nodes (MN) in wireless IPv6 networks. Mobile IPv6 provides seamless handover when MN moves to a new network. It allows MN to be identified by its static home address, regardless of its location in the Internet. When the MN moves away from home, it must send information about its new location and new IP address to a *home agent* (HA) in the home network. The HA can then intercept and forward packets to the MN at the new location. The new address of MN at a foreign network is called *care-of-address* (CoA).

The signaling process during the mobile IPv6 handover is illustrated in Fig. 1. After the MN moves to the foreign network, it will create a CoA using a prefix received in the router advertisement message. The MN then performs *duplicate address detection* (DAD) to check the validity of this new address. Once the CoA is configured, the MN registers its CoA with HA via the *binding update* message. HA confirms the registration with *binding acknowledgment*. At this point, MN can continue to receive packets from existing connections via HA. If desired, MN can also register its new address with CN to create a direct optimal route, as shown in Figure 1.

Handover latency in Mobile IPv6 is a noteworthy issue especially since it could greatly affect performance of time-critical applications such as voice-over-IP. Reference [2] identifies four delay components during mobile IPv6 handover, namely movement detection time, address configuration time, binding registration time, and route optimization time. Testbed

experiments show that address configuration time is the largest delay component with duplicate address detection taking up as much as 1000ms [2]. While we can reduce delay by turning off DAD, it is not recommended since IP address uniqueness cannot be guaranteed especially in a large IPv6 networks.

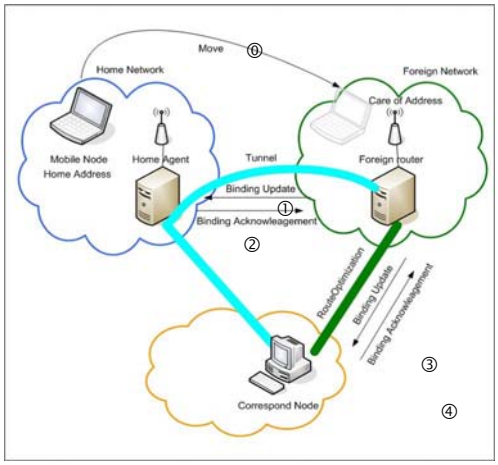


Figure 1. Mobile IPv6 signaling steps

This paper surveys techniques for reducing delay during the DAD procedure. The standard scheme of duplicate address detection is explained in Section II. Section III discusses seven improving techniques, namely Optimistic DAD [3], Advance DAD [4][5], Proactive DAD [6], MLD-DAD [7], END [8], FNDD [9], and DHCPv6 [10]. Their feature comparison and delay analysis are presented in Section IV. Finally, Section V concludes the paper.

II. Duplicate Address Detection

In IPv6 networks, address configuration can be done statelessly [11]. IPv6 nodes configure their own IP addresses based on prefix information sent by IPv6 routers. IPv6 nodes generate a 64-bit suffix and combine with the prefix obtained from the router to get the 128-bit IPv6 address. Note that the 64-bit suffix can be generated randomly or based on the interface hardware address. This new address is referred to as a *Tentative Address*. Before the Tentative Address can be used, it must be tested for uniqueness within the network. This test process is referred to as the *duplicate address detection* or *DAD*.

To perform DAD, first, an IPv6 node sends a *Neighbor Solicitation* (NS) message to all devices in the network asking if anyone is using its Tentative Address. In this NS-DAD message, source IP address is left unspecified, and the target address field is set to the Tentative Address. If the address is already used by another node, that node must send a *Neighbor Advertisement* (NA) message to defend its address. The asking node then has to configure a new Tentative Address and repeat the DAD process.

On the other hand, if there is no conflict, the asking node waits for *RetransTimer* milliseconds. If there is no NA reply, the node may ask again *DupAddrDetectTransmits* times to make sure there is really no conflict. After that, the asking node can start using the Tentative Address in all communication.

The problem of DAD is that during the waiting period, the node cannot use the Tentative Address to correspond with any device other than to send NSs and listen for NAs. In addition, the default value of RetransTimer is 1000ms and DupAddrDetectTransmits is 1. Thus, the DAD delay becomes RetransTimer·DupAddrDetectTransmits. Such delay is not acceptable in real-time applications such as voice-over-IP. Although the value of RetransTimer can be reduced, the risk of a defending NA arriving too late will also increase.

III. Existing Strategies

This section discusses existing techniques for improving DAD. These techniques all have the same goal—to shorten the time that the Tentative Address is not usable.

A. Optimistic DAD (RFC4429)

The Optimistic DAD [3] technique is based on the assumption that the chance of 64-bit address collision is very small, especially if a well-distributed random number generator is used. Optimistic DAD allows the use of Tentative Address in some communication messages while waiting for completion of the normal DAD process. As a result, communication disruption is minimized.

To avoid the effect of address collision, Optimistic DAD modifies the RFC2461 (Neighbor Discovery for IPv6) and RFC2462 (IPv6 Stateless Address Autoconfiguration) such that nodes do not advertise its Link Layer address (MAC address) together with the untested Tentative Address, referred to here as *Optimistic Address*. This is to prevent other devices from saving the Optimistic Address in their neighbor cache entries, in case of a collision. Specifically, this is achieved by:

1) Setting the 'Override' flag to 0 in NA so that neighbors will not override existing cache entries with the Optimistic Address.
2) Never using the Optimistic Address as a source of NS because NS contains a Source Link-Layer Address Option (SLLAO), i.e., Link Layer address of node. This may create conflicts in neighbor cache entries.
3) Never using the Optimistic Address as a source of Router Solicitation (RS) message if the RS contains a SLLAO. Instead, using another address or unspecified address, or sending RS without a SLLAO.

The effects of not being able to send NS and RS from the Optimistic Address are twofold. First, the node cannot contact a router because it cannot send RS from the Optimistic address. As a result, it has to wait until the address is no longer optimistic (DAD process completes) to do so. Second, the node cannot contact a neighbor that is not already in its neighbor cache. To get around this, the node must forward packets via its default router. The default router will then provide an ICMP Redirect message which gives a Target Link-Layer Address Option (TLLAO) containing Link Layer address of the destined neighbor. This allows the node to update its neighbor cache and begin communication with this neighbor directly.

The advantage of Optimistic DAD is that it can interoperate with standard DAD without adding new devices. In addition, the new protocol maintains the stateless characteristic of IPv6. However, its delay performance depends highly on the size of the network. The larger the network is, the higher the probability of address collision becomes. In reality, it would be hard to configure well-distributed and truly random addresses in a large network. So collision is possible. If collision happens, the recover time of Optimistic DAD could be even longer than that of standard DAD.

B. Advance DAD (A-DAD)

The Advanced DAD [4][5] strategy tries to improve DAD delay by storing a pool of unique IPv6 addresses at an access router (AR). Each AR generates random addresses as a background process and performs DAD on them. The duplicate-free addresses are then stored in *Passive Proxy Cache.* AR acts as a passive proxy for addresses. It listens to neighbor discovery messages from other nodes in the network. If it hears another node performing DAD on the same address in its pool, AR must silently remove that address from its Passive Proxy Cache and generate and test a new address to keep the list size constant. Moreover, AR must not send NA messages using the duplicate-free address, nor must it reply to NS messages targeting that address.

A-DAD requires exchange of modified RS and RA messages to help MN obtain the duplicate-free address from AR. The protocol starts when MN receives an L2 switching trigger. MN then sends RS to all-router multicast address (FF02::2). The modified RS includes *NCoA-Request option*, indicating a request of a unique CoA. This option field contains MN's previous CoA and MN's Link-Layer address. Upon receiving RS with such option field, AR sends a modified RA message to MN's previous CoA, containing *NCoA-Reply option*. This option includes the unique CoA selected from AR's Passive Proxy Cache. Upon receiving the address from AR, MN can proceed with Mobile IPv6 binding registration without delay. As for AR, after assigning the address to MN, it must remove that address from the pool, generate and test a new address, and store the assigned address together with MN's Link-Layer address in its neighbor cache.

The steps above explain how A-DAD works with standard Mobile IPv6 handover. A-DAD is also designed to work with Fast Mobile IPv6 protocol [13] that utilizes movement prediction to perform fast handover and early binding registration.

Details of A-DAD with Fast Mobile IPv6 can be found in [5].

Like standard DAD, A-DAD approach is pessimistic. It assumes address collision is inevitable. While standard DAD lets it to happen and corrects it later, A-DAD prevents it from happening. In comparison to Optimistic DAD, A-DAD's pessimistic approach can also eliminate DAD delay completely when used in conjunction with Fast Mobile IPv6. Without movement prediction, the delay is only a roundtrip time of RS and RA between MN and new AR.

A-DAD also has a few drawbacks. First of all, it puts additional burden on AR to manage a pool of candidate addresses. In addition, it breaks the stateless auto-configuration principle of IPv6. Since nodes can no longer provision their own addresses, it is impossible to run protocol like Secure Neighbor Discovery (SEND), which requires addresses generated from node's private key.

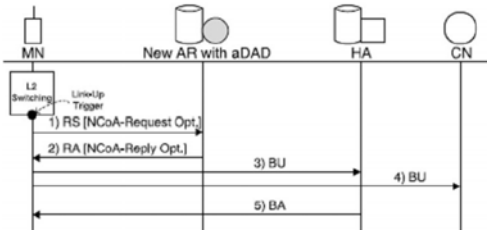


Figure 2. Advanced DAD signaling in standard MIPv6 [5]

C. *Proactive DAD (P-DAD)*

The concept of Proactive DAD [6] is to perform DAD before MN moves to a new network. Movement prediction is an essential part in this protocol. When MN suspects it will join a new AR, it obtains a prefix of the new AR from a central storage called *Mobile-node Attachment Point (MAP) table*. The MAP table stores a list of access point's Basic Service Set ID (BSSID), IP address of AR that manages that BSSID, and prefix information of that AR. The MAP table is located in *Regional Information Point (RIP) servers*, whose IP address is known by all ARs. After MN obtains a new prefix, it can configure a new CoA while still at the home network. The new CoA must be tested for duplication. This is done by communicating a series of new messages with the new AR.

MN performs DAD on new CoA by sending CoA-preAllocate Request message to new AR. This message contains the new CoA MN wants to check. AR will check the address against its *Registration cache* and may perform standard DAD on that address. If the address is unique, AR replies with CoA-preAllocate Reply with a U-bit set to indicate no duplication. AR also stores this address into its Registration cache. When MN actually moves to the new network, it activates this CoA with AR by sending CoA_activation Request and waiting for CoA_activation Reply. Once the address is activated, AR removes it from the Registration cache. If MN does not receive CoA_activation Reply, or cannot predict its movement, it must perform standard DAD in the new network.

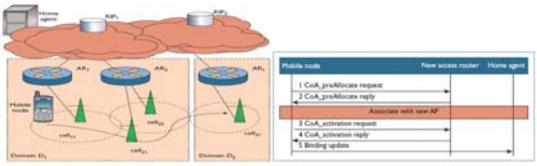


Figure 3. Proactive DAD infrastructure and signaling steps [6]

Compared to A-DAD, P-DAD maintains IPv6 stateless address autoconfiguration. By letting AR perform DAD instead of MN, duplicate checking can happen before MN joins a new network. However, such proactive method requires setting up additional storages and servers, such as MAP table, RIP servers, and Registration Cache.

In terms of delay performance, P-DAD requires a round-trip delay between MN and new AR for exchange of CoA_activation request and reply. This delay is comparable to that of A-DAD without movement prediction.

D. *Multicast Listener Discovery DAD (MLD-DAD)*

This technique attempts to end DAD early by taking advantage of IPv6 multicast

listener discovery. According to [11], before sending NS-DAD, an interface must join the solicited-node multicast address associated with the Tentative Address. Therefore, MLD-DAD checks for address uniqueness simply by asking a router whether a node is the first to enter the solicited-node multicast group based on the Tentative Address [7]. If the node is the first to enter the group, then it implies that no other node is currently using that unicast IPv6 address.

MLD-DAD signaling starts with a node that wants to configure a new IPv6 address sending a MLD Report message to tell its presence and to ask whether the multicast group is empty. If it is, a MLD Querier router will respond with a MLD Response message. Upon receiving the MLD Response, the node can start using the Tentative Address in all communication.

The MLD Report and Response do not replace the NS and NA of standard DAD process. The node must send a NS-DAD message after it sends MLD Report. So the standard DAD process occurs in parallel with the MLD-DAD. DAD will terminate once a node receives MLD Response (i.e., the address is unique) or receives NA (i.e., the address is in use), or the RetransTimer expires (i.e., the address is unique).

In case the solicited-node multicast group already exists with at least one member, MLD-DAD interprets this information as duplicate address and falls back to standard DAD. Note, however, that existence of the solicited-node multicast group does not always guarantee that the Tentative Address is a duplicate. According to [13], the solicited-node multicast address is computed based on the lower 24 bits of the unicast address. Therefore, there is a chance that two interfaces having their 64-bit interface addresses differed in bits 24-63 would still join the same multicast group. Such false-negative interpretation would result in unnecessarily extra DAD delay for MLD-DAD.

In the case of no duplicate address, MLD-DAD shortens DAD delay to one roundtrip time of MLD Report and MLD Response. In the case of address conflicts (real and false conflicts), MLD-DAD takes as long as standard DAD to complete. The improvement of MLD-DAD comes at a burden of a router to monitor all multicast groups and respond to all MLD-DAD Reports. To use MLD-DAD, all devices on the network perform MLD properly. Moreover, they must understand the modified version of MLD Report and the new format of MLD Response.

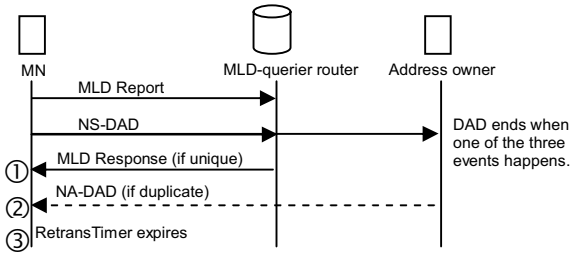


Figure 4. MLD-DAD signaling

E. Enhanced Neighbor Discovery (END)

Similar to MLD-DAD, Enhanced Neighbor Discovery uses a positive acknowledgment message to terminate DAD early, and takes advantage of the solicited-node multicast address group of the Tentative Address [8]. END uses a bridge-like entity called *Neighbor Discovery Relay Agent (NDRA)* to maintain a cache of IPv6 addresses of all nodes. The ND Relay Agent learns about all IPv6 in use from MLD Join messages sent by nodes that have just configured new IPv6 addresses. END protocol uses modified RA as a positive acknowledgment message. The modified RA includes an Available Address Option which indicates the uniqueness of IPv6 address carried in the option. In case of address collision, the NDRA forwards a NS-DAD message to the address owner who, in turn, will reply with a defending NA message.

In this strategy, DAD delay takes NS and RA roundtrip time between MN and NDRA. The delay is comparable to that of MLD-DAD in case of no conflict. In case of conflict, however, END nodes could take longer than MLD-DAD nodes to detect conflict since standard DAD proceeds after, not parallel to, the address lookup. Moreover, like MLD-DAD, END lets an agent keep track of available addresses by listening to solicited-node multicast messages. Therefore, END will also suffer the same false-negative pitfall (i.e., thinking that a unique address is a duplicate),

resulting in longer additional DAD delay than that of MLD-DAD.

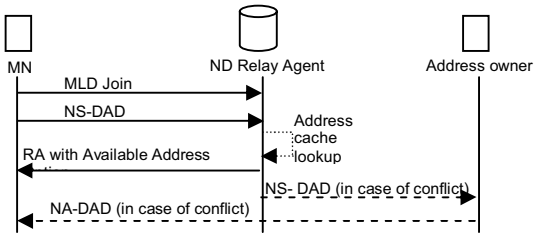


Figure 5. Enhanced Neighbor Discovery for DAD optimization.

F. Fast Neighbor Discovery and DAD (FNDD)

FNDD attempts to reduce DAD latency by using neighbor cache of an access router (AR) at a foreign network [9]. The protocol proposes new RS and RA message formats for communication between MN and the new AR. MN sends new RS with its untested CoA attached in an option field to the new AR. The AR then checks uniqueness of CoA by looking up its neighbor cache. The assumption here is that the AR knows addresses of all nodes attached to it in its neighbor cache. AR sends back new RA with an option field indicating whether CoA can be used or not. If CoA is a duplicate address, new RA will contain a pre-configured, unique address that MN should use.

FNDD protocol is similar in concept to MLD-DAD and END. Like MLD-DAD, FNDD lets AR maintain a list of addresses in use and check address uniqueness. However, instead of listening to multicast messages as the MLD-DAD and END, FNDD makes AR listen to NS, NA, and RS messages to compile a list of existing addresses.

The effect of checking DAD against neighbor cache entries is that AR can only check for duplicate address on one link only (i.e., only on nodes attached to this link). This mechanism will not work if AR has multiple links sharing the same prefix.

DAD latency resulting from FNDD is the roundtrip time of new RS and new RA plus the neighbor cache look up time. Reference [9] claims that the look up time is 5.28 μsec in worst case. Note that the DAD delay is the same in both cases of unique CoA and duplicate CoA since the AR will either approve the CoA or assign a new CoA. Note that to be able to assign a new CoA, AR must maintain a pre-configured table of available addresses (i.e., perform standard DAD on them ahead of time). Thus FNDD address assignment becomes stateful configuration just like A-DAD.

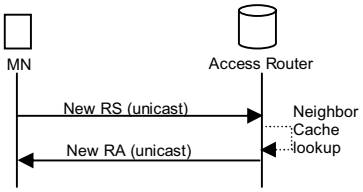


Figure 6. FNDD signaling during DAD

G. Dynamic Host Configuration Protocol for IPv6 (DHCPv6)

In IPv6, network administrators have a choice between using IPv6 stateless address autoconfiguration [11] and stateful address configuration via DHCPv6 [10]. DHCP6 works the same way as DHCP for IPv4. A DHCP server is responsible for address assignment, and must keep states of which addresses are in use and which are free. When MN enters a network that deploys DHCPv6, MN will hear a RA message with 'M' bit set. MN will know to contact DHCPv6 server instead of configuring the address by itself. To contact a DHCPv6 server, MN sends a *Solicit* message to FF02::1:2, *All_DHCP_Relay_Agents_and_Servers* multicast address. Any DHCP servers can respond with an *Advertise* message. The MN then chooses one of the servers and sends a *Request* message to the server asking for IP addresses and other configuration information (e.g., DNS, NTP addresses). The server responds with a *Reply* message that contains the confirmed addresses and configuration parameters.

In the four-way communication explained above, globally unique identifiers are required to identify the client and server. DHCPv6 defines DHCP Unique Identifier (DUID) based on Link-Local address. So standard DAD is still required to ensure Link-Local address uniqueness in DHCPv6 networks.

Therefore, DHCPv6 is not recommended as a replacement for DAD. For one reason, it cannot eliminate DAD completely. Secondly, stateful DHCPv6 is not suitable

for networks with a large number of nodes. Address renumbering under DHCPv6 would be much more complicated than under stateless autoconfiguration. However, DHCPv6 is not without merits. DHCPv6 can provide appropriate configuration information, such as DNS and NTP servers, which is not possible to obtain from the stateless autoconfiguration.

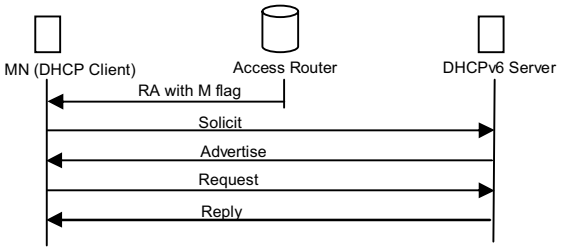


Figure 7. Typical DHCPv6 signaling steps

IV. Comparative Analysis

A. Feature Comparison

This section compares features and characteristics of the seven DAD strategies discussed previously. To facilitate comparison, we group together DAD methods that have similar features or use similar techniques. For example, considering stateful against stateless address configuration, three schemes opt for stateful assignment—A-DAD, FNDD, and DHCPv6. A drawback of stateful configuration is that MN has no choice as to what address is assigned. As a result, some novel addressing schemes cannot be used, such as one that is based on private key cryptography.

In terms of utilizing movement information, only A-DAD and P-DAD stand out. They are the only two schemes that really incorporate movement prediction and Fast Mobile IPv6. Consequently, they are suitable for wireless network environment. Optimistic DAD is also suitable for wireless network because the possibility of address collision causing packet retransmission will not affect performance of wireless networks where packet loss is common [3].

One common concept is to terminate DAD early by transmitting positive acknowledgment. MLD-DAD, END, and FNDD all send various kinds of messages to inform MN that the Tentative Address is good to use. In order to give such positive acknowledgment, these schemes require a central agent that keeps track of addresses in use. It is interesting to note that all schemes presented here, except Optimistic DAD, all require some kinds of address list management by a central agent such as access router or server. As a result, reliability of networks deploying these schemes will depend highly on reliability of such agent.

Another common technique is to take advantage of IPv6 multicast listener discovery. Both MLD-DAD and END assume all nodes must join a solicited-node multicast address associated with each unicast address. So to check for duplicate address, they simply check whether a specific solicited-node multicast group is empty. However, these schemes leave out the possibility that multiple unicast addresses could join the same solicited-node multicast group. Such shortcoming results in determining false duplicates, which unnecessarily lengthens the DAD process.

In terms of backward compatibility with standard DAD, all of the protocols either define new message types or modify options in existing messages. Consequently, compatibility with standard IPv6 protocol becomes an issue. If some networks traversed by MN do not understand the new protocols, MN will not be able to configure its CoA. Out of the seven protocols, only Optimistic DAD, P-DAD, MLD-DAD, and END provides a fall back to standard DAD.

Another critical issue that plagues many of the proposed protocols is the possibility of address collision due to racing conditions. Because many protocols rely on an intermediary device (e.g., router or server) for address monitoring and assignment, there is a chance that a NS-DAD message from node A for address X arrives right after a router has confirmed to MN that address X is unique. If the router lacks a mechanism to defend for address X after its approval, and if the approval message from router to MN is still in transit, node A would think that address X is still available. As a result, both node A and MN will unknowingly configure the same address. Consider A-DAD for example, once a router assigns an address from cache to MN, that address is removed from cache. So the router will not be aware

of any NS-DAD concerning that address. Similar signaling mishap can also happen to P-DAD, END, and FNDD because they all perform positive confirmation on unique addresses. Such condition can be prevented by designing a mechanism to defend for acknowledged addresses until MN is ready to defend for itself.

To sum up, the comparison of protocol characteristics provide insights into benefits and shortcomings of different DAD techniques. For example, advantageous features include movement prediction, positive acknowledgment, and backward compatibility. On the other hand, message modification, additional workload on routers, and racing conditions are undesirable and must be properly handled. As to which protocol is most recommended, it may depend on factors, such as network topology, network load, number of nodes, movement characteristics, and types of applications.

TABLE I

CHARACTERISTICS OF IMPROVED DAD STRATEGIES

Feature Characteristics	**DAD Strategies**
Stateful address assignment	A-DAD, FNDD, DHCPv6
Benefit from movement prediction	A-DAD, P-DAD
Suitable for wireless environment	A-DAD, P-DAD, O-DAD
Use positive acknowledgment	MLD-DAD, END, FNDD
Require central storage of address list	A-DAD, P-DAD MLD-DAD, END, FNDD, DHCPv6
Require workload on devices other than MN, CN, and HA.	A-DAD, P-DAD, MLD-DAD, END, FNDD, DHCPv6
Rely on Multicast Listener Discovery	MLD-DAD, END
Compatible with standard DAD	O-DAD, P-DAD, MLD-DAD, END
Create new message types	P-DAD, MLD-DAD
Modify options in existing messages	O-DAD, A-DAD, MLD-DAD, END, FNDD
Suffer from NS-DAD racing condition	A-DAD, P-DAD, END, FNDD

B. Delay Comparison

This section analyzes delay performance of the DAD protocols described previously. The goal is to compare DAD delay, i.e., time during which the Tentative Address is not usable. Since these protocols require different network setups, it is difficult to measure and benchmark delay performance through testbed experiments. Instead, we provide a quick, back-of-envelope analysis of a few scenarios—when the tested address is unique, and when it is a duplicate.

Following notations are used in the analysis. D_{unq} *and* D_{dup} refer to the DAD delay in case the Tentative Address is unique and duplicate, respectively. T_m donates time to process and transmit a message m. $P_{src\text{-}dst}$ is propagation delay between *src* and *dst*, and M the memory lookup delay.

Standard DAD: D_{unq} of standard DAD is a function of RetransTimer and DupAddrDetectTransmits while D_{dup} is longer than D_{unq} by N-roundtrip time between MN and the address owner, where N is a number of successive DAD trials before a unique address is found.

$$D_{unq} = RetransTimer \cdot DupAddrDetectTransmits \quad (1)$$

$$D_{dup} = (T_{NS} + T_{NA} + 2P_{MN\text{-}AddrOwner})N + D_{unq(StdDAD)} \quad (2)$$

Optimistic DAD: Optimistic DAD can eliminate delay completely in case of unique address assignment. However, D_{dup} could be much longer than D_{dup} of standard DAD due to time to recover broken conversation, $T_{recovery}$.

$$D_{unq} = 0 \quad (3)$$

$$D_{dup} = D_{dup(StdDAD)} + T_{recovery} \quad (4)$$

A-DAD: A-DAD also eliminates DAD delay in case of Fast Mobile IPv6. For normal Mobile IPv6, A-DAD takes a roundtrip time between MN and AR. D_{dup} is not applicable since A-DAD prevents duplication by stateful assignment.

$$D_{unq(FMIPv6)} = 0 \quad (5)$$

$$D_{unq(MIPv6)} = T_{RS} + T_{RA} + 2P_{MN\text{-}AR} + M \quad (6)$$

$$D_{dup} = \text{N/A} \quad (7)$$

P-DAD: We consider two cases, with and without movement prediction. Without movement prediction, delay is the same as standard DAD. With prediction, D_{unq} is

shortened to a roundtrip time between MN and AR, but D_{dup} is lengthened by a timer for waiting to receive the CoA_activation reply message.

$$D_{unq(prediction)} = T_{CoA\text{-}req} + T_{CoA\text{-}rep} + 2P_{MN\text{-}AR} + M \quad (8)$$

$$D_{unq(no\ prediction)} = D_{unq(StdDAD)} \quad (9)$$

$$D_{dup(prediction)} = WaitForCoAactReplyTimer + D_{dup(StdDAD)} \quad (10)$$

$$D_{dup(no\ prediction)} = D_{dup(StdDAD)} \quad (11)$$

MLD-DAD: MLD-DAD performs standard DAD in parallel. So D_{dup} is as long as standard DAD while D_{unq} is the minimum delay of the two events—receiving MLD-Response, or standard DAD ending.

$$D_{unq} = \min(T_{MLD\text{-}Report} + T_{MLD\text{-}Response} + 2P_{MN\text{-}AR} + M, D_{unq(StdDAD)}) \quad (12)$$

$$D_{dup} = D_{dup(StdDAD)} \quad (13)$$

END: D_{unq} takes a roundtrip time between MN and ND Relay Agent while D_{dup} is longer than D_{unq} by N-roundtrips of successive DAD trials via ND Relay Agent.

$$D_{unq} = T_{NS} + T_{RA} + 2P_{MN\text{-}NDRA} + M \quad (14)$$

$$D_{dup} = (2T_{NS} + T_{NA} + P_{MN\text{-}NDRA} + P_{NDRA\text{-}AddrOwner} + M)N + D_{unq(END)} \quad (15)$$

FNDD: D_{unq} takes a roundtrip time between MN and AR. D_{dup} is the same as D_{unq} because it uses stateful assignment.

$$D_{unq} = T_{RS} + T_{RA} + 2P_{MN\text{-}AR} + M \quad (16)$$

$$D_{dup} = D_{unq(FNDD)} \quad (17)$$

DHCPv6: DHCPv6 takes two-roundtrip time for four-message exchange before assigning a unique address.

$$D_{unq} = T_{Solicit} + T_{Advertise} + T_{Request} + T_{Reply} + 4P_{MN\text{-}dhcp} \quad (18)$$

$$D_{dup} = \text{N/A} \quad (19)$$

From this delay analysis, protocols that stand out are Optimistic DAD and A-DAD (FMIPv6) which can eliminate D_{unq} completely, i.e., $D_{unq} = 0$. The other DAD schemes reduce D_{unq} to similar level—approximately one roundtrip time between MN and AR. As for D_{dup}, stateful protocols like A-DAD and DHCPv6 prevent address duplication altogether. Otherwise, protocols like END and FNDD, which do not rely on standard DAD in the case of conflict, offer minimal D_{dup}. Figure 8 shows the comparison.

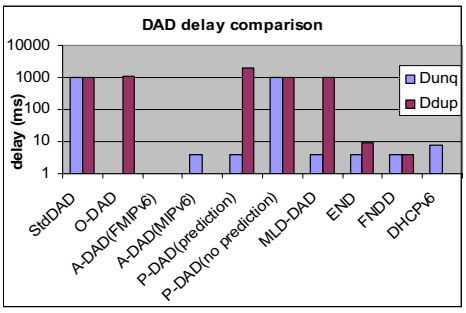


Figure 8. DAD delay comparison

V. CONCLUSIONS

This paper surveys existing techniques for reducing delay during the address configuration in mobile IPv6. We compare and contrast seven new DAD strategies against the standard DAD and against one another. In addition, we examine distinctive features and concepts behind each protocol. Advantages and shortcomings of each protocol are thoroughly discussed. Moreover, we systematically analyze delay performance under the best case and the worst case of each protocol. Through this study, we have obtained insights into strengths and weaknesses of different configuration techniques, as well as a delay benchmark, which will be valuable for designing future DAD protocols.

REFERENCES

[1] D. Johnson, C. Perkins and J. Arkko, “Mobility Support in IPv6”, RFC 3775, June 2004.

[2] A. Busaranun, P. Pongpaibool and P. Supanakoon, “Handover Performance of Mobile IPv6 on Linux Testbed,” ECTI-CON 2006.

[3] N. Moore, “Optimistic Duplicate Address Detection for IPv6,” IETF RFC 4429, April 2006.

[4] Y. Han, J. Choi and H. Jang, "Advance Duplicate Address Detection," Internet Draft, Dec 2003, expired.

[5] Y.-H. Han, S.-H. Hwang, “Care-of address provisioning for efficient IPv6 mobility support,” Computer Communications 29(2006), pp.1422-1432.

[6] C.-C. Tseng, Y.-C. Wong, L.-H. Yen and Kai-Cheng Hsu, “Proactive DAD: A Fast Address-Acquisition Strategy for Mobile IPv6 Networks,” Dec 2006.

[7] G. Daley and R. Nelson, “Duplicate Address Detection Optimization using IPv6 Multicast Listener Discovery,” Internet Draft, Feb 2003, expired.

[8] F. Xia and B. Sarikaya, “Duplicate Address Detection Optimization Using Enhanced Neighbor Discovery,” Internet Draft, Dec 2006.

[9] B. Park, S. Lee, and H. Latchman, “A Fast Neighbor Discovery and DAD Scheme for Fast Handover in Mobile IPv6 Networks,” Proc. Of ICNICONSMCL’06.

[10] R. Droms, Ed., “Dynamic Host Configuration Protocol for IPv6 (DHCPv6),” IETF RFC 3315, July 2003.

[11] S. Thomson and T. Narten, “IPv6 Stateless Address Autoconfiguration,” IETF RFC 2462, Dec 1998.

[12] T. Narten, E. Nordmark, and W. Simpson, “Neighbor Discovery for IP version 6 (IPv6),” IETF RFC 2461, Dec 1998.

[13] R. Koodli Ed., “Fast Handovers for Mobile IPv6,” IETF RFC 4068, July 2005.

[14] R. Hinden and S. Deering, “IP Version 6 Addressing Architecture,” IETF RFC 2373, July 1998.

ภาคผนวก ข

โคดโปรแกรม

ในภาคผนวก ข นี้ประกอบด้วยส่วนของตัวโคดที่มีการแก้ไข้ดึงมาเฉพาะส่วนที่มีการเพิ่มเต็มสำหรับการเพิ่มวิธี FR-DAD และ A-DAD ซึ่งประกอบไปด้วยหลายไฟล์ ดังนี้

1. NDStatehost.cc
2. NDStatehost.h
3. MIPv6CDSMobileNode.cc
4. MIPv6NDStateHost.cc
5. MIPv6NDStateHost.h
6. Netconf2.dtd
7. XMLOmnetParser.cc
8. RoutingTable6.h
9. ICMPv6NDMessage.cc
10. ICMPv6NDMessage.h
11. IPv6Forword.cc
12. IPv6Interfacedata.cc
13. IPv6Interfacedata.h
14. MIPv6test4.xml
15. MIPv6test4.ned
16. Omnet.ini
17. Default2.ini

1. NDStatehost.cc

```
#include <iomanip> //setprecision
#include <climits> //UINT_MAX
#include <memory>  //auto_ptr
#include <cassert>
#include <iostream>
#include <boost/cast.hpp>
//----------------add code-------------------
//#include <boost/bind.hpp>
//#include "cSignalMessage.h"
//----------------add code-------------------

#include "NDStateHost.h"
#include "NDTimers.h"
#include "NeighbourDiscovery.h"
#include "ICMPv6Message_m.h"
…
…
…
NDStateHost::NDStateHost(NeighbourDiscovery* mod)
  :NDState(mod), rt(mod->rt), stateEntered(false), addrResln(0),
   rtrSolicited(0),      managedFlag(false),      otherFlag(false),      cbRetrySol(makeCallback(this,
&NDStateHost::sendRtrSol))
{
  //----------------add code-------------------
            cout<<rt->nodeName()<<" -:Class NDStateHost"<<" at "<<nd->simTime()<<endl;
  //----------------add code-------------------
  //Node is already initialised
  if (nextState != 0)
```

```
  return;

 ift = InterfaceTableAccess().get();

 addrResln = OPP_Global::findModuleByName(nd, "addrResln"); // XXX try to get rid of
pointers to other modules --AV
 assert(addrResln != 0);
 if (!addrResln)
  DoutFatal(dc::core|error_cf, "Cannot find Address Resolution module");

 // Initialise this magic number once only as its constant during runtime
 if (addrResln != 0 && addrResGate == UINT_MAX)
 {
  addrResGate = addrResln->findGate("ICMPIn");
 }

 // join all-node multicast group
 IPv6Address node_multicast(ALL_NODES_NODE_ADDRESS);
 IPv6Address link_multicast(ALL_NODES_LINK_ADDRESS);
 IPv6Address site_multicast(ALL_NODES_SITE_ADDRESS);
 rt->joinMulticastGroup(node_multicast);
 rt->joinMulticastGroup(link_multicast);
 rt->joinMulticastGroup(site_multicast);

 ifStats.resize(ift->numInterfaceGates());

 rtrSolicited = new bool[ift->numInterfaceGates()];
 std::fill(rtrSolicited, rtrSolicited + ift->numInterfaceGates(), false);

 nd->scheduleAt(nd->simTime(),
```

```
            new cTTimerMessage<void, NDStateHost>
            (Ctrl_NodeInitialise, static_cast<NDStateHost*>(this),
             &NDStateHost::nodeInitialise, "InitialiseNode"));
}
…
…
…
void NDStateHost::initialiseInterface(size_t ifIndex)
{
  //----------------add code------------------
        cout<<rt->nodeName()<<" -:Class initialiseInterface"<<" at "<<nd-
>simTime()<<endl;
  //----------------add code------------------
  //For other tentative addr for interface that has assigned
  //link local addr.
  dupAddrDetOtherAddr(ifIndex);

  //Routers already have prefixes set by manual conf.
  if (!rt->isRouter())
  {
    //According to Sec. 6.3.7 host should still send Rtr Sol after receive Unsol
    //Rtr Ad as those may be incomplete. But in this sim it is always
    //complete. Change if sim becomes more realistic.
    if (!rtrSolicited[ifIndex]) //rt->getPrefixesByIndex(ifIndex).empty())
    {
      NDTimer* tmr = new NDTimer;
      tmr->max_sends = MAX_RTR_SOLICITATIONS;
      tmr->counter = 0;
      tmr->ifIndex = ifIndex;

      sendRtrSol(tmr);
    }

  }
  //Unnecessary to do prefix assignments at this time because processing of
  //router adv. will do that anyway
}
…
…
…
IPv6Address NDStateHost::linkLocalAddr(size_t ifIndex)
{
```

```
  //----------------add code------------------
        cout<<rt->nodeName()<<" -:Class linkLocalAddr"<<" at "<<nd-
>simTime()<<endl;
  //----------------add code------------------
  IPv6Address linkLocal;

  InterfaceEntry *ie = ift->interfaceByPortNo(ifIndex);

  // Search for LinkLocal address first as without a unique one can forget about
  // assigning other interfaces
  for (size_t k = 0; k < ie->ipv6()->tentativeAddrs.size(); k++)
  {
   ///Search for presence of manually assigned link local prefix
   if (IPv6Address(LINK_LOCAL_PREFIX).isNetwork(ie->ipv6()-
>tentativeAddrs[k]))
    {
     linkLocal = ie->ipv6()->tentativeAddrs[k];
     ifIndex = ie->outputPort();
     assert(ifIndex != (unsigned int) -1);
     ifStats[ifIndex].manualLinkLocal = true;
  //----------------add code------------------
     linkLocal.scope();
  //----------------add code------------------
     break;
    }
  }

  //No link local addr manually assigned then auto conf one
  if (linkLocal == IPv6_ADDR_UNSPECIFIED)
  {
   IPv6Address lhs;
   lhs.setAddress(LINK_LOCAL_PREFIX);

   //----------------add code------------------
   if (rt->ardad())
        linkLocal = generateID(lhs);
   else
   //----------------add code------------------

   linkLocal = ipv6_addr_fromInterfaceToken(lhs, ie->interfaceToken());
   //----------------add code------------------
   linkLocal.setPrefixLength(EUI64_LENGTH);
   //----------------add code------------------
   //Determine scope
   linkLocal.scope();

   ie->ipv6()->tentativeAddrs.push_back(linkLocal);
```

```
    Dout(dc::xml_addresses, "LLAddr "<<rt->nodeName()<<":"<<ifIndex<<" "
        <<linkLocal);
  }
  return linkLocal;
}
…

…

…
void NDStateHost::nodeInitialise()
{
  //----------------add code-------------------
          cout<<rt->nodeName()<<" -:Class nodeInitialise"<<" at "<<nd-
>simTime()<<endl;
  //----------------add code-------------------
  for(size_t ifIndex = 0; ifIndex < ift->numInterfaceGates(); ifIndex++)
  {

  //----------------add code-------------------
    InterfaceEntry *ie = ift->interfaceByPortNo(ifIndex);
    if(rt->ardad() && rt->isXServer() &&  ie->ipv6()->tentativeAddrs.size()>0)
    {
          int k=0;
          if(ie->ipv6()-
>matchPrefix(c_ipv6_addr(LINK_LOCAL_PREFIX),EUI64_LENGTH,true)!=IPv6_
ADDR_UNSPECIFIED)
            k=1;
          for(int i = 0; i<100; i++)
          {
            IPv6Address pool(ie->ipv6()->tentativeAddrs[k]);
            pool.setAddress(generatePool(ie->ipv6()->tentativeAddrs[k],i+1));
            ie->ipv6()->NCoAPool.push_back(pool);
          cout<<rt->nodeName()<<" pool address"<<" = "<<pool.address()<<endl;
          }
          ie->ipv6()->prefixIP.push_back(ie->ipv6()->tentativeAddrs[k]);
    }

  //----------------add code-------------------

    NDTimer* tmr = new NDTimer;
    tmr->ifIndex = ifIndex;
    tmr->tentativeAddr = linkLocalAddr(ifIndex);
    dupAddrDetection(tmr);
  }
}
```

```
/**
  Prepare for DAD of tentative addr

  MIPv6: tentativeAddr will be added to list of addresses accepted by
  RoutingTable6::localDeliver but will not be used as source address yet.
*/
void NDStateHost::detectDupAddress(size_t ifIndex, const IPv6Address&
tentativeAddr)
{
  //----------------add code-------------------
        cout<<rt->nodeName()<<" -:Class detectDupAddress"<<" at "<<nd-
>simTime()<<endl;
  //----------------add code-------------------
  ift->interfaceByPortNo(ifIndex)->ipv6()->tentativeAddrs.push_back(tentativeAddr);
  //Call func to send ns
  NDTimer* tmr = new NDTimer;
  tmr->ifIndex = ifIndex;
  tmr->tentativeAddr = tentativeAddr;
  dupAddrDetection(tmr);
}

///Do dupAddrDet on manually assigned addresses of interface except the link
///local addr which must have been assigned to reach this point
void NDStateHost::dupAddrDetOtherAddr(size_t ifIndex)
{
  //----------------add code-------------------
        cout<<rt->nodeName()<<" -:Class dupAddrDetOtherAddr"<<" at "<<nd-
>simTime()<<endl;
  //----------------add code-------------------
  //When this is done then do for every other addr when there's outstanding
  //tentativeAddr
  InterfaceEntry *ie = ift->interfaceByPortNo(ifIndex);

  IPv6Address tentativeAddr;

  //Dup Addr Det for each addr
  for (size_t j = 0; j < ie->ipv6()->tentativeAddrs.size(); j++)
  {
    tentativeAddr = ie->ipv6()->tentativeAddrs[j];

    //Call func to send ns
    NDTimer* tmr = new NDTimer;
    tmr->ifIndex = ifIndex;
    tmr->tentativeAddr = tentativeAddr;
 //----------------add code-------------------
        cout<<rt->nodeName()<<" -:Class dupAddrDetOtherAddr"<<" tentative=
"<<tentativeAddr.address()<<endl;
```

```
  //----------------add code------------------
    dupAddrDetection(tmr);
  }
}

/**
  Initiate DupAddrDet
  Resends the NgbrSol during DupAddrDet.
  When timeout expires will assign address.
 */
void NDStateHost::dupAddrDetection(NDTimer* tmr)
{
  //----------------add code------------------
        cout<<rt->nodeName()<<" -:Class dupAddrDetection IPv6 = "<<tmr-
>tentativeAddr.address()<<" at "<<nd->simTime()<<endl;
  //----------------add code------------------
  InterfaceEntry *ie = ift->interfaceByPortNo(tmr->ifIndex);

  double delay = 0;

  if (tmr->msg == 0)
  {

    //Don't forget to do this otherwise ND will not work on tentative/assigned
    //address.  Can't delete solimcast_addr as joinMulticastGroup will take
    //ownership of pointer
    IPv6Address solimcast_addr(tmr->tentativeAddr.solNodeAddr());
    rt->joinMulticastGroup(solimcast_addr);

    if (ie->ipv6()->dupAddrDetectTrans < 1)
    {
  //----------------add code------------------
        cout<<rt->nodeName()<<" -:Class dupAddrDetection ---- skipped
DupAddrDet"<<" at "<<nd->simTime()<<endl;
  //----------------add code------------------
      Dout(dc::ipv6|dc::neighbour_disc|dc::notice|flush_cf, rt-
>nodeName()<<":"<<tmr->ifIndex
          <<" skipped DupAddrDet "<< tmr->tentativeAddr << " assigned on "<<nd-
>simTime());

      //Don't do dup addr detect so just assign addr
      dupAddrDetSuccess(tmr);
      delete tmr;
      return;
    }

    NS* ns = new NS;
```

```
   tmr->dgram = new IPv6Datagram(IPv6Address(IPv6_ADDR_UNSPECIFIED),
solimcast_addr);
   tmr->dgram->setHopLimit(NDHOPLIMIT);

   tmr->dgram->encapsulate(ns);
   tmr->dgram->setName(ns->name());
   tmr->dgram->setTransportProtocol(IP_PROT_IPv6_ICMP);
   ns->setTargetAddr(static_cast<ipv6_addr>(tmr->tentativeAddr));
```

```
  //----------------add code-------------------
   if(rt->ardad() && rt->isNode())
   {
         ns->setarDADFlag(true);
         ns->setSrcLLAddr(ie->llAddrStr());
         rt->setrcvNCOA(false);
   }
  //----------------add code-------------------
```

```
   if (rt->odad())
   {
    assert(!ie->ipv6()->tentativeAddrAssigned(tmr->dgram->srcAddress()));
    if (ie->ipv6()->tentativeAddrAssigned(tmr->dgram->srcAddress()))
    {
     Dout(dc::warning, rt->nodeName()<<":"<<tmr->ifIndex<<" cannot use a
tentative address as source of NS (ODAD)");
      return;
    }
   }

/*#ifdef USE_MOBILITY
   //last paragraph of 11.5.2 of mipv6 revision 24  says should not delay
   if (rt->mobilitySupport() && rt->isMobileNode() && rt->awayFromHome())

    delay = 0;

  else
  {
#endif
    delay = uniform(0, MAX_RTR_SOLICITATION_DELAY);
#ifdef USE_MOBILITY
  }
#endif*/
```

```
  //----------------add code-------------------
         delay = 0;

   if (rt->adad() && rt->ardad() && rt->isNode())
```

```
        tmr->max_sends = 2;
    else
//----------------add code------------------
    tmr->max_sends = ie->ipv6()->dupAddrDetectTrans; //Max Dup addr retries

    tmr->dgram->setOutputPort(tmr->ifIndex);
    tmr->timeout = 1;//ie->ipv6()->retransTimer/1000.0; //Dup addr det timeout

    tmr->msg = new HostTmrMsg(8001,//Tmr_DupAddrSolTimeout,
                        static_cast<NDStateHost*>(this),
                        &NDStateHost::dupAddrDetection, tmr, false /*FIXME causes
memory leak!*/,
                        "DupAddrDet");

    timerMsgs.push_back(tmr->msg);
  }


  //Duplicate packet assuming dest will delete packet
  if (tmr->counter < tmr->max_sends)
  {



    Dout(dc::neighbour_disc|continued_cf, rt->nodeName()<<":"<<tmr->ifIndex
        <<" Tentative addr:"<<tmr->tentativeAddr);
    Dout(dc::finish, nd->simTime() <<" DupAddrDet sends: "<<  tmr->counter
        <<" max:"<<tmr->max_sends<<" timeout:"<< setprecision(4) << tmr->timeout
        <<" initial delay:"<<delay);

    //Send directly to IPv6Output as we need to send to a particular interface
    //which the routingCore can't determine from dest addr.
//----------------add code------------------
    if (rt->adad()&& rt->ardad() && rt->isNode())
    {
          if (tmr->counter == 0)
        {
            nd->sendDelayed(tmr->dgram->dup(), delay, "outputOut", tmr->ifIndex);
            tmr->counter++;
            nd->scheduleAt(nd->simTime() + 0.3, tmr->msg);
          }
          else if (tmr->counter == 1 && rt->rcvNCOA())
          {
            tmr->counter++;
            nd->scheduleAt(nd->simTime(), tmr->msg);
          } else {
          tmr->counter++;
```

```
            nd->scheduleAt(nd->simTime() + delay + tmr->timeout, tmr->msg);
        }
    }
    else
 //----------------add code------------------
    {
    nd->sendDelayed(tmr->dgram->dup(), delay, "outputOut", tmr->ifIndex);

    tmr->counter++;

    nd->scheduleAt(nd->simTime() + delay + tmr->timeout, tmr->msg);
    }
  }
  else
  {

    Dout(dc::ipv6|dc::neighbour_disc|dc::notice|flush_cf, rt->nodeName()<<":"<<tmr-
>ifIndex
         <<" tentative addr: "<<tmr->tentativeAddr<<" assigned on "<<nd->simTime());

  //----------------add code------------------
    if ( rt->ardad() && rt->isNode() && rt->rcvNCOA() )
          rt->setrcvNCOA(false);
    else
  //----------------add code------------------

    dupAddrDetSuccess(tmr);

    timerMsgs.remove(tmr->msg);
    //msg deletes tmr (_arg) internally
    delete tmr->msg;
  }
}


//private funcs

/**
   Assigns the address and initialise rest of interface once only

   @todo store the amount of time used for doing DAD and subtract from
   storedLifetime of assigned address for exact lifetimes
*/
void NDStateHost::dupAddrDetSuccess(NDTimer* tmr)
{
  //----------------add code------------------
```

```
          cout<<rt->nodeName()<<" -:Class dupAddrDetSuccess IPv6 Success =
"<<tmr->tentativeAddr.address()<<" at "<<nd->simTime()<<endl;
  //----------------add code-------------------
  //tmr->tentativeAddr->setStoredLifetime(tmr->tentativeAddr->storedLifetime() -
'DAD time');
  rt->assignAddress(tmr->tentativeAddr, tmr->ifIndex);

  //Need to do this only once for each interface after link local addr is
  //assigned (that is the first address to be assigned)
  if (!ifStats[tmr->ifIndex].initStarted)
  {
    //Do what this node should do after common initialisation is
    //accomplished. (once per node)
    if (!stateEntered)
      enterState();
    //May need to add a delay and call initialiseInterface otherwise
    //the node appears to do processing very quickly
    initialiseInterface(tmr->ifIndex);
    ifStats[tmr->ifIndex].initStarted = true;
  }

  InterfaceEntry *ie = ift->interfaceByPortNo(tmr->ifIndex);

  if (rt->odad())
  {
    sendUnsolNgbrAd(tmr->ifIndex, ie->ipv6()->inetAddrs[ie->ipv6()-
>inetAddrs.size()-1]);
    Dout(dc::addr_resln|dc::neighbour_disc|dc::custom|flush_cf, rt-
>nodeName()<<":"<<tmr->ifIndex
         <<" ODAD sent unsolicated NA with override bit set to all nodes");
    return;
  }

  //----------------add code-------------------
  if (rt->ardad() && rt->isNode())
  {
    cout<<rt->nodeName()<<" -:send Un NA"<<" IPv6 Success = "<<tmr-
>tentativeAddr.address()<<" at "<<nd->simTime()<<endl;
    sendUnsolNgbrAd(tmr->ifIndex, tmr->tentativeAddr);
  }
  //----------------add code-------------------

#if USE_MOBILITY

  if (rt->mobilitySupport() && rt->isMobileNode() && rt->awayFromHome())
  {
    ipv6_addr potentialCoa = ie->ipv6()->inetAddrs[ie->ipv6()->inetAddrs.size()-1];
```

```
 //----------------add code------------------
cout<<rt->nodeName()<<" -:Class dupAddrDetSuccess"<<" potentialCoa = "<<ie-
>ipv6()->inetAddrs[ie->ipv6()->inetAddrs.size()-1]<<" at "<<nd->simTime()<<endl;
 //----------------add code------------------
    MobileIPv6::MIPv6NDStateHost* mipv6StateMN =
      boost::polymorphic_downcast<MobileIPv6::MIPv6NDStateHost*> (this);
    assert(mipv6StateMN);
    Dout(dc::debug, rt->nodeName()<<" potential coa in dupAddrDetSuc
"<<potentialCoa);
 //----------------add code------------------
    mipv6StateMN->FutureCoa(potentialCoa);
 //----------------add code------------------
    mipv6StateMN->sendBU(potentialCoa);

    invokeCallback(potentialCoa);
  }
#endif //USE_MOBILITY

}
…

…

…
void NDStateHost::sendUnsolNgbrAd(size_t ifIndex, const ipv6_addr& target,bool
overrideFlag) //----------------Modifiled function------------------
{
  //----------------add code------------------
          cout<<rt->nodeName()<<" -:Class sendUnsolNgbrAd"<<" at "<<nd-
>simTime()<<endl;
  //----------------add code------------------
//Not called from another module yet
//  OPP_Global::ContextSwitcher switcher(nd);

  InterfaceEntry *ie = ift->interfaceByPortNo(ifIndex);

  assert(ie->ipv6()->addrAssigned(target));
  //Send NA that is
  ipv6_addr src = target;
  IPv6Datagram* UNAdgram = new IPv6Datagram(src,
                                c_ipv6_addr(ALL_NODES_LINK_ADDRESS));
  UNAdgram->setHopLimit(NDHOPLIMIT);
  bool solicited = false;

  //----------------add code------------------
  bool override = overrideFlag; //true;
  //----------------add code------------------
```

```
  NA* una = new NA(src, ie->llAddrStr(), rt->isRouter(), solicited, override);

  //----------------add code------------------
  if (rt->ardad())
          una->setarDADFlag(true);
  //----------------add code------------------

  UNAdgram->encapsulate(una);
  UNAdgram->setName(una->name());
  UNAdgram->setTransportProtocol(IP_PROT_IPv6_ICMP);
  nd->send(UNAdgram, "outputOut", ifIndex);
}

///disc 6.3.4 and autoconf 5.5.3
std::auto_ptr<RA> NDStateHost::processRtrAd(std::auto_ptr<RA> rtrAdv)
{
  //----------------add code------------------
          cout<<rt->nodeName()<<" -:Class processRtrAd of NDState"<<" at "<<nd-
>simTime()<<endl;
  //----------------add code------------------
  IPv6Datagram* dgram = check_and_cast<IPv6Datagram*>(rtrAdv-
>encapsulatedMsg());
  ipv6_addr srcAddr = dgram->srcAddress();
  size_t ifIndex = dgram->inputPort();

  RouterEntry* re = 0;

  assert(ifIndex < ift->numInterfaceGates());
  Dout(dc::router_disc|flush_cf, rt->nodeName()<<":"<<ifIndex<<" "<<nd-
>simTime()
      <<" received rtr adv "<<*rtrAdv);
…
…
…
bool  NDStateHost::prefixAddrConf(size_t ifIndex, const ipv6_addr& prefix,
                          size_t prefix_len, unsigned int preferredLifetime,
                          unsigned int storedLifetime)
{
  //----------------add code------------------
          cout<<rt->nodeName()<<" -:Class prefixAddrConf"<<" at "<<nd-
>simTime()<<endl;
  //----------------add code------------------
  InterfaceEntry *ie = ift->interfaceByPortNo(ifIndex);
  assert(storedLifetime != 0);
  //Create a new address and add it to tentative if link local address was
  //an addr conf addr from unique Interface ID
```

```
  if (prefix_len + ie->interfaceToken().length() == IPv6_ADDR_BITLENGTH )
  {
    //Can't autoconf other addresses till link local done (implementation
    //requires that link local be first address assigned not part of spec).
    //Just wait till link local addr assigned as host will solicit router
    if (!linkLocalAddrAssigned(ifIndex))
      //received an unsolicated rtr adv or other nodes solicited this adv
      return false;
```

```
    //----------------add code-------------------
    if(ie->ipv6()->matchPrefix(prefix,prefix_len,false)!=IPv6_ADDR_UNSPECIFIED)
            return true;

    ipv6_addr newAddr;
    if (rt->ardad())
            newAddr = generateID(prefix);
    else
            newAddr = ipv6_addr_fromInterfaceToken(prefix, ie->interfaceToken());
    //----------------add code-------------------
```

```
    //ipv6_addr newAddr = ipv6_addr_fromInterfaceToken(prefix, ie->interfaceToken());

    IPv6Address addrObj(newAddr);
    addrObj.setPrefixLength(prefix_len);
    addrObj.setStoredLifetimeAndUpdate(storedLifetime);
    addrObj.setPreferredLifetime(preferredLifetime);
```

```
    //----------------add code-------------------
    cout<<rt->nodeName()<<" -:Class prefixAddrConf"<<" IPv6="<<addrObj.address()<<endl;
    for (size_t k = 0; k < ie->ipv6()->tentativeAddrs.size(); k++)
            cout<<rt->nodeName()<<" Ten "<<k<<" = "<<ie->ipv6()->tentativeAddrs[k].address()<<endl;
    for (size_t k = 0; k < ie->ipv6()->inetAddrs.size(); k++)
            cout<<rt->nodeName()<<" INet "<<k<<" = "<<ie->ipv6()->inetAddrs[k].address()<<endl;

    //----------------add code-------------------
```

```
#ifdef USE_MOBILITY
    //Guess anyone that's away from home is applicable not just mobile node role
    //(e.g. mobile router/map).

    //Doing DAD always not just when away from home as trend is to do it always
    //even in home subnet I guess
    if (rt->mobilitySupport() && rt->isMobileNode())
      Dout(dc::debug, rt->nodeName()<<" awayfromHom="<<(rt->awayFromHome()?"away":"home"));
```

```
    if (rt->mobilitySupport() && rt->isMobileNode())// && rt->awayFromHome())
    {
     //Do not do DAD on outstanding DAD which is indicated by been in
     //tentativeAddrs. Only when interface initialises are there already
     //stuff in tentativeAddrs
     if (ie->ipv6()->tentativeAddrAssigned(addrObj))
     {
      return true;
     }

     Dout(dc::ipv6|dc::address_timer|flush_cf, rt->nodeName()<<":"<<ifIndex<<" "
          <<addrObj<<" undergoing DAD prefix="
          <<prefix<<"/"<<dec<<(int)prefix_len<<" at "
          <<nd->simTime());
     detectDupAddress(ifIndex, addrObj);
    }
    else
    {
#endif //USE_MOBILITY
     if (!ie->ipv6()->tentativeAddrAssigned(addrObj))
     {
     if (!ifStats[ifIndex].manualLinkLocal)
     {
      //Skip DAD if auto conf link local interfaceID passed
      Dout(dc::ipv6|dc::address_timer|flush_cf, rt->nodeName()<<":"<<ifIndex<<" "
          <<addrObj
          //<<ie->ipv6()->inetAddrs[ie->ipv6()->inetAddrs.size()-1]
          <<" assigned (autoconf) from prefixOpt="
          <<prefix<<"/"<<dec<<(int)prefix_len<<" at "
          <<nd->simTime());
      rt->assignAddress(addrObj, ifIndex);
     }
     else
      detectDupAddress(ifIndex, addrObj);
     }
#ifdef USE_MOBILITY
    }
#endif //USE_MOBILITY

    return true;
  }
  return false;
}

/**
 * @brief test received Neighbour solicitations' and advertisements'
```

```
 * target address is a tentative address of this node and process them
 * accordingly Sec. 5.4.3-5 inclusive of RFC 2462.
 *
 *  @param targetAddr is the target address from the NS or NA.
 *  @param recDgram is the datagram that contained the NS or NA
 */
bool NDStateHost::checkDupAddrDetected(const ipv6_addr& targetAddr,
IPv6Datagram* recDgram)
{
  //----------------add code-------------------
          cout<<rt->nodeName()<<" -:Class checkDupAddrDetected"<<" at "<<nd-
>simTime()<<endl;
  //----------------add code-------------------
  InterfaceEntry *ie = ift->interfaceByPortNo(recDgram->inputPort());
…
…
…
void NDStateHost::processNgbrSol(std::auto_ptr<NS> msg)
{

  if (!valNgbrSol(msg.get()))
  {

    Dout(dc::neighbour_disc|dc::warning|flush_cf, rt->nodeName()
        <<" Discarded invalid NS");
    //Silently Discard sol Sec. 7.1.1
    return;
  }
  //----------------add code-------------------
          cout<<rt->nodeName()<<" -:Class process NS"<<" at "<<nd-
>simTime()<<endl;
  //----------------add code-------------------
  IPv6Datagram* recDgram = check_and_cast<IPv6Datagram*>(msg-
>encapsulatedMsg());

  if (!rt->odad())
  {
    InterfaceEntry *ie = ift->interfaceByPortNo(recDgram->inputPort());

    //check if dupAddrDetection failed (another node is doing dupAddrDetection on
    //a tentative addr)
    if(recDgram->srcAddress() == IPv6_ADDR_UNSPECIFIED)
    {
      if (checkDupAddrDetected(msg->targetAddr(), recDgram))
        return;
    }
    else  //unicast src address
```

```
    {
     //Another node performs addr resln on a tentative addr so we ignore
     for(size_t j = 0; j < ie->ipv6()->tentativeAddrs.size(); j++)
     {
      if((ipv6_addr)(ie->ipv6()->tentativeAddrs[j]) == msg->targetAddr())
      {
       Dout(dc::neighbour_disc|dc::notice, rt->nodeName()
           <<"Ignoring Ngbr Sol with target of "<<msg->targetAddr()
           <<" as target is a tentative address.");
       return;
      }
     }
    }
  }
  else
  {
   Dout(dc::debug, rt->nodeName()<<":"<<recDgram->inputPort()<<" "<<nd-
>simTime()
       <<" ODAD processNgbrSol responding to NS always");
  }
```

```
  //----------------add code-------------------
  InterfaceEntry *ie = ift->interfaceByPortNo(recDgram->inputPort());
  if (rt->ardad() && msg->arDADFlag())
    {
         if (rt->isXServer() &&  msg->hasSrcLLAddr() && ie->ipv6()-
>prefixIP[0].isNetwork(msg->targetAddr()))
            {

             // Send NA + NCoA + LLA(MN) + override = false

              //ipv6_addr NCOAtarget = generateID(msg->targetAddr()); //for Phase I no
pooling

              ipv6_addr NCOAtarget = ie->ipv6()->NCoAPool[0];

              if(!(rt->adad()))
              {
              //write map (msg->target, target)
              IPv6Address wait_poolncoa(ie->ipv6()->NCoAPool[0]);
              IPv6Address wait_pooldad(ie->ipv6()->NCoAPool[0]);
              wait_pooldad.setAddress(msg->targetAddr());

              ie->ipv6()->WaitPoolncoa.push_back(wait_poolncoa);
              ie->ipv6()->WaitPooldad.push_back(wait_pooldad);
              }
for(size_t i = 0; i<ie->ipv6()->WaitPooldad.size(); i++)
```

```
{
        cout<<rt->nodeName()<<" "<<i<<" pool ncoa address"<<" = "<<ie->ipv6()-
>WaitPoolncoa[i].address()<<endl;
        cout<<rt->nodeName()<<" "<<i<<" pool dad address"<<" = "<<ie->ipv6()-
>WaitPooldad[i].address()<<endl;
}
            ie->ipv6()->removeFromPool(ie->ipv6()->NCoAPool[0],true);

            ipv6_addr NCOAsrc = NCOAtarget;
            IPv6Datagram* NCOAdgram = new IPv6Datagram(NCOAsrc,
                                    c_ipv6_addr(ALL_NODES_LINK_ADDRESS));
            NCOAdgram->setHopLimit(NDHOPLIMIT);
            bool solicited = false;
            bool override = false;
            NA* NCOAna = new NA(NCOAsrc,msg->srcLLAddr(), rt->isRouter(),
solicited, override);
            NCOAna->setarDADFlag(true);
            NCOAdgram->encapsulate(NCOAna);
            NCOAdgram->setName(NCOAna->name());
            NCOAdgram->setTransportProtocol(IP_PROT_IPv6_ICMP);
            nd->send(NCOAdgram, "outputOut", recDgram->inputPort());


            if(!(rt->adad()))
            {
           // prepare stamp ACK timeout
            IPv6Address wait_pooldad(ie->ipv6()->NCoAPool[0]);
            wait_pooldad.setAddress(msg->targetAddr());
            NDTimer* wACK = new NDTimer;
            wACK->ifIndex = recDgram->inputPort();
            wACK->tentativeAddr = wait_pooldad;
            waitACK(wACK);
            }
           }
    }
  //----------------add code-------------------
```

```
  // TODO Check for NUD solicitation

  //Otherwise respond with NA should be done by forwarding orig message to Addr
  //Resln Module
  assert(addrResln != 0);
```

```
  //----------------add code-------------------
if ( !(msg->arDADFlag() && rt->isXServer()))
```

```
  //----------------add code-------------------
  nd->sendDirect(msg.release(), 0, addrResln, addrResGate);

}
```

```
  //----------------add code-------------------
void NDStateHost::waitACK(NDTimer* wACK)
{
  InterfaceEntry *ie = ift->interfaceByPortNo(wACK->ifIndex);
  if(wACK->msg == 0)
  {
    wACK->max_sends = ie->ipv6()->dupAddrDetectTrans;
    wACK->timeout = 1.5;

    wACK->msg = new HostTmrMsg(8001,
                        static_cast<NDStateHost*>(this),
                        &NDStateHost::waitACK, wACK, false /*FIXME causes memory
leak!*/,
                        "DupAddrDet");

    timerMsgs.push_back(wACK->msg);
    rt->setdadACK(false);
    rt->setncoaACK(false);
  }

  if (wACK->counter < wACK->max_sends)
  {
    wACK->counter++;

    nd->scheduleAt(nd->simTime() + wACK->timeout, wACK->msg);
  }
  else
  {

    size_t i = -1;
    size_t k = ie->ipv6()->findIPFromWaitPool(wACK->tentativeAddr,false);

/*        cout<<rt->nodeName()<<" "<<"k = ....."<<k<<endl;
          cout<<rt->nodeName()<<" tentative address"<<" = "<<wACK-
>tentativeAddr.address()<<endl;
for(size_t i = 0; i<ie->ipv6()->WaitPooldad.size(); i++)
{
          cout<<rt->nodeName()<<" "<<i<<" pool ncoa address"<<" = "<<ie->ipv6()-
>WaitPoolncoa[i].address()<<endl;
          cout<<rt->nodeName()<<" "<<i<<" pool dad address"<<" = "<<ie->ipv6()-
>WaitPooldad[i].address()<<endl;
}*/
```

```
IPv6Address wait_address;
if (i!=k)
   wait_address = ie->ipv6()->WaitPooldad[k];
else
{
      timerMsgs.remove(wACK->msg);
   delete wACK->msg;
      return;
}

if ( rt->rcvncoaACK() )
{
   // tmr->tentativeAddr ---> in IP Pooling
      //--------Keep IP in IP Pooling and delete NCaA from IP Pooling

      ie->ipv6()->NCoAPool.push_back(wait_address);
      ie->ipv6()->removeFromPool(ie->ipv6()->WaitPooldad[k],false);
      ie->ipv6()->removeFromPool(ie->ipv6()->WaitPoolncoa[k],false);
      rt->setncoaACK(false);
}
else if ( rt->rcvdadACK() )
{
   // NCoA ---> in IP Pooling
      // tmr->tentativeAddr Delete
      wait_address.setAddress(ie->ipv6()->WaitPoolncoa[k]);
      ie->ipv6()->NCoAPool.push_back(wait_address);
      ie->ipv6()->removeFromPool(ie->ipv6()->WaitPooldad[k],false);
      ie->ipv6()->removeFromPool(ie->ipv6()->WaitPoolncoa[k],false);
      rt->setdadACK(false);
}
else
{
      //--------delete NCoA from IP Pooling and IP Test
      ie->ipv6()->removeFromPool(ie->ipv6()->WaitPooldad[k],false);
      ie->ipv6()->removeFromPool(ie->ipv6()->WaitPoolncoa[k],false);

      NDTimer* tmr = new NDTimer;
      tmr->ifIndex = wACK->ifIndex;
      tmr->tentativeAddr = generateID(wait_address);

      wait_address.setAddress(tmr->tentativeAddr);

      dupAddrDetection(tmr);

      if (ie->ipv6()->addrAssigned(tmr->tentativeAddr))
      {
        ie->ipv6()->removeAddress(tmr->tentativeAddr);
```

```
            ie->ipv6()->NCoAPool.push_back(wait_address);
        }
    }

    for(size_t i = 0; i<ie->ipv6()->NCoAPool.size(); i++)
        cout<<"      "<<rt->nodeName()<<"  "<<i<<" pool address"<<" = "<<ie-
>ipv6()->NCoAPool[i].address()<<endl;

    timerMsgs.remove(wACK->msg);
    delete wACK->msg;
  }
}
  //----------------add code-------------------
///Rename to processDupAddrResp
void NDStateHost::processNgbrAd(std::auto_ptr<NA> ngbrAdv)
{

  if (!valNgbrAd(ngbrAdv.get()))
    return;

  IPv6Datagram* recDgram = check_and_cast<IPv6Datagram*>(ngbrAdv-
>encapsulatedMsg());
  //----------------add code-------------------
  if (rt->ardad() && ngbrAdv->arDADFlag())
  {
        InterfaceEntry *ie = ift->interfaceByPortNo(recDgram->inputPort());
        if (rt->isNode() &&  ngbrAdv->hasTargetLLAddr())
        {
          if ((ngbrAdv->targetLLAddr().compare(ie->llAddrStr())==0) && (ie-
>ipv6()->tentativeAddrs.size() == 1) )
            {
                IPv6Address target(ie->ipv6()->tentativeAddrs[0]);
                target.setAddress(ngbrAdv->targetAddr());
                ie->ipv6()->removeTentativeAddress(ie->ipv6()->tentativeAddrs[0]);
                //rt->assignAddress(target, recDgram->inputPort());
                rt->setrcvNCOA(true);

                NDTimer* ncoa_tmr = new NDTimer;
                ncoa_tmr->ifIndex = recDgram->inputPort();
                ncoa_tmr->tentativeAddr = target;
                dupAddrDetSuccess(ncoa_tmr);

            }
        }

        if (!(rt->adad()) && rt->isXServer() && !(rt->adad()) && ie->ipv6()-
>prefixIP[0].isNetwork(ngbrAdv->targetAddr()))
```

```
        {
          size_t i=-1;

          size_t k = ie->ipv6()->findIPFromWaitPool(ngbrAdv->targetAddr());
          if (k==i)
          {
                if (i!=(ie->ipv6()->findIPFromWaitPool(ngbrAdv-
>targetAddr(),false)))
                  rt->setdadACK(true);
          }
          else
                rt->setncoaACK(true);
        }
  } else
  //----------------add code-------------------
  //Check if NA comes from a node that has assigned our tentative addr
  //Determine if this was a dupAddrDet adv, or one response to an addr res req.
  if(!ngbrAdv->solicited())
    if (checkDupAddrDetected(ngbrAdv->targetAddr(), recDgram))
      return;


  // TODO Check for NUD solicitation

  //Otherwise proc of addr res responses should be done by forwarding orig
  //message to Addr Resln Module
  //----------------add code-------------------
//if ( !(rt->ardad() && ngbrAdv->arDADFlag() && rt->isNode()))
  //----------------add code-------------------
  nd->sendDirect(ngbrAdv.release(), 0, addrResln, addrResGate);

}
…
…
…
void  NDStateHost::processRedirect(std::auto_ptr<Redirect> msg)
{
  //----------------add code-------------------
          cout<<rt->nodeName()<<" -:Class processRedirect"<<" at "<<nd-
>simTime()<<endl;
  //----------------add code-------------------
  if (!valRedirect(msg.get()))
    return;
…
…
…
ipv6_addr NDStateHost::generateAddress(const ipv6_prefix& prefix) const
```

```
{
  //----------------add code------------------
        cout<<rt->nodeName()<<" -:Class generateAddress"<<" at "<<nd-
>simTime()<<endl;
  //----------------add code------------------
  ipv6_addr ret;
…
…
…
void NDStateHost::TriggerCallback(cTimerMessage *tmr){

    cerr<<rt->nodeName()<<" "<<nd->simTime()
        <<" Received L2 Trigger "<<tmr->kind()<<endl;
    Dout(dc::neighbour_disc|dc::notice|flush_cf, rt->nodeName()<<" "<<nd-
>simTime()
        <<" Received L2 Trigger "<< tmr->kind());
}
  //----------------add code------------------

ipv6_addr NDStateHost::generateID(const ipv6_addr& prefix) const
{
        cout<<rt->nodeName()<<" -:Class generateID"<<" at "<<nd-
>simTime()<<endl;

  ipv6_addr ret;
  ret.extreme=prefix.extreme;
  ret.high=prefix.high;
  ret.normal = OPP_Global::generateInterfaceId();
  ret.low = OPP_Global::generateInterfaceId();

  //set u and g bits to 0
  ret.normal &= 0xFFFcFFFF;
  return ret;
}

ipv6_addr NDStateHost::generatePool(const ipv6_addr& prefix, int setlow) const
{
  ipv6_addr ret;
  ret.extreme=prefix.extreme;
  ret.high=prefix.high;
  ret.normal = 0xFFFFFFFF;
  ret.low = setlow;

  return ret;
}
```

```
void NDStateHost::forReturnHome(size_t ifIndex,const ipv6_addr& haAddr)
{
InterfaceEntry *ie = ift->interfaceByPortNo(ifIndex);

IPv6Datagram* dgramNA = new IPv6Datagram(ie->ipv6()-
>inetAddrs[1],c_ipv6_addr(ALL_NODES_LINK_ADDRESS));
dgramNA->setHopLimit(NDHOPLIMIT);
  bool solicited = true;
NA* na= new NA(ie->ipv6()->inetAddrs[1],ie->llAddrStr(),false, solicited,true);
dgramNA->encapsulate(na);
  dgramNA->setName(na->name());
  dgramNA->setTransportProtocol(IP_PROT_IPv6_ICMP);
  nd->send(dgramNA, "outputOut", ifIndex);

IPv6Datagram* dgram = new IPv6Datagram(ie->ipv6()-
>inetAddrs[1],c_ipv6_addr(ALL_NODES_LINK_ADDRESS));
NS* ns= new NS(haAddr,ie->llAddrStr());

dgram->setHopLimit(NDHOPLIMIT);
dgram->encapsulate(ns);
  dgram->setName(ns->name());
  dgram->setTransportProtocol(IP_PROT_IPv6_ICMP);
  nd->send(dgram, "outputOut", ifIndex);

cout<<rt->nodeName()<<" -:Class forReturnHome MAC="<< ie->llAddrStr() <<" at
"<<nd->simTime()<<endl;

}


  //----------------add code-------------------
} //namespace IPv6NeighbourDiscovery

//  LocalWords:  multicast sim addr
```

2. NDStatehost.h

…
…
…
/// Invoked to test whether received Neighbour solicitations and
/// advertisements' target address is a tentative address of this node and
/// process them accordingly Sec. 5.4.3-5 inclusive of RFC 2462.
bool checkDupAddrDetected(const ipv6_addr& targetAddr, IPv6Datagram*
recDgram);

```
//----------------add code-------------------
  void waitACK(NDTimer* wACK);
//----------------add code-------------------
```

//@}
…
…
…
bool callbackAdded(const ipv6_addr& tentativeAddr, int message_id);
// GD inserted
void TriggerCallback(cTimerMessage* tmr);

```
//----------------add code-------------------
  ipv6_addr generateID(const ipv6_addr& prefix) const;
  ipv6_addr generatePool(const ipv6_addr& prefix, int setlow) const;
  void forReturnHome(size_t ifIndex,const ipv6_addr& haAddr);
//----------------add code-------------------
```

protected:
…
…
…

3. MIPv6CDSMobileNode.cc

…

…

…

```
ipv6_addr MIPv6CDSMobileNode::formCareOfAddress(
   boost::weak_ptr<MIPv6RouterEntry> re, InterfaceEntry *ie) const
  {
   assert(re.lock().get() != 0);
   const ipv6_prefix& pref = re.lock().get()->prefix();
   assert((int) pref.length <= ie->interfaceToken().length());
   IPv6Address coaObj(pref);
   coaObj.truncate();
   ipv6_addr coa;

 //----------------add code-------------------
   if(ie->ipv6()->matchPrefix(coaObj,pref.length,false)!=IPv6_ADDR_UNSPECIFIED)
            coa = ie->ipv6()->matchPrefix(coaObj,pref.length,false);
   else
       coa = ipv6_addr_fromInterfaceToken(coaObj, ie->interfaceToken());

   cout<<" -:Class formCareOfAddress"<<" CoA= "<<coa<<endl;
 //----------------add code-------------------

   return coa;
  }

  ipv6_prefix MIPv6CDSMobileNode::formHomeAddress(
   boost::weak_ptr<MIPv6RouterEntry> re, InterfaceEntry *ie, bool primaryHoa)
  {
 //----------------add code-------------------
```

```
    cout<<"   -:Class formHomeOfAddress set homeAddress"<<endl;
  //----------------add code------------------
    ipv6_prefix pref = re.lock().get()->prefix();
    ipv6_addr hoa = formCareOfAddress(re, ie);
    pref.prefix = hoa;
    if (primaryHoa)
    {
  //----------------add code------------------
    cout<<"   -:Class formHomeOfAddress set homeAddress2"<<endl;
  //----------------add code------------------
      _homeAddr = pref;
    }
    return pref;
  }
…
…
...
std::ostream& MIPv6CDSMobileNode::operator<<(std::ostream& os) const
  {
    for (BULI it = bul.begin(); it != bul.end(); it++)
    {
      os<<*(*it)<<"\n";
    }
    //Not useful as its a shared_ptr to bue
    //copy(bul.begin(), bul.end(), ostream_iterator<BindingUpdateList::value_type >(os, "\n"));
    return os;
  }
//----------------add code------------------
 void MIPv6CDSMobileNode::setFutureCoa(const ipv6_addr& ncoa)
 {
```

```
    cerr<<"futureCoa is " <<futureCoa<<endl;
    assert((ncoa == IPv6_ADDR_UNSPECIFIED && futureCoa !=
IPv6_ADDR_UNSPECIFIED) ||
          (ncoa != IPv6_ADDR_UNSPECIFIED && futureCoa == IPv6_ADDR_UNSPECIFIED));
    futureCoa = ncoa;
  }
 //----------------add code-------------------
} //namespace MobileIPv6
```

4. MIPv6 NDStateHost.cc

```
…
…
…
std::auto_ptr<RA> MIPv6NDStateHost::processRtrAd(std::auto_ptr<RA> rtrAdv)
{
 //----------------add code------------------
        cout<<rt->nodeName()<<"  MIPv6 -:Class processRtrAd of MIPv6"<<" at "<<nd-
>simTime()<<endl;
 //----------------add code------------------
 rtrAdv = IPv6NeighbourDiscovery::NDStateHost::processRtrAd(rtrAdv);

 if (rtrAdv.get() == 0)
  return rtrAdv;

 IPv6Datagram* dgram = check_and_cast<IPv6Datagram*>(rtrAdv->encapsulatedMsg());

 size_t ifIndex = dgram->inputPort();

 const ipv6_addr& ll_addr = dgram->srcAddress();

 boost::shared_ptr<MIPv6RouterEntry> bha = mipv6cdsMN->findRouter(ll_addr);
 MIPv6RouterEntry* ha = bha.get();

 bool isHA = rtrAdv->isHomeAgent();

 //After rereading 11.3.1 of draft 18 it appears that no HA flag is totally
 //ignored. (All subsequent bullet points refer to HA)
 //However for now will leave original assumption about existence of
```

```
  //MIPv6Router role also can't really remove isHA from here and
  //MIPv6RouterEntry and just make it a list of HA because we'd still need to
  //maintain a copy of currentRouter for move detect(missedRtrAdv)/handover purposes
//   if (!isHA)
//   {
//     if (bha && bha->isHomeAgent())
//       mipv6cdsMN->removeHomeAgent(bha);
//     return rtrAdv;
//   }
  if (!isHA && bha && bha->isHomeAgent())
  {
    mipv6cdsMN->removeHomeAgent(bha);
    return rtrAdv;
  }
```

```
  //----------------add code-------------------
    InterfaceEntry *ie = ift->interfaceByPortNo(ifIndex);
    if (!linkLocalAddrAssigned(ifIndex))
      //received an unsolicated rtr adv or other nodes solicited this adv
      return rtrAdv;
    if(IPv6Address(LINK_LOCAL_PREFIX).isNetwork(ie->ipv6()->inetAddrs[ie->ipv6()-
>inetAddrs.size()-1]))
            return rtrAdv;
    if(awayFromHome() && (ie->ipv6()->inetAddrs.size()==2) && (!rt->odad()))
            return rtrAdv;
  //----------------add code-------------------
```

```
  int preference = 0;
  unsigned long lifetime = rtrAdv->routerLifetime();
…
```

```
…
...
void MIPv6NDStateHost::handover(boost::shared_ptr<MIPv6RouterEntry> newRtr)
{
  //----------------add code------------------
cout<<rt->nodeName()<<" -:Class Handover"<<" at "<<nd->simTime()<<endl;
  //----------------add code------------------

  assert(mipv6cdsMN->currentRouter() != newRtr);

  boost::shared_ptr<MIPv6RouterEntry> oldRtr = mipv6cdsMN->currentRouter();

  mipv6cdsMN->setAwayFromHome(true);

  //normal handover requires sending BU to pHA. (if we don't have pHA) just
  //handover to new subnet and remove old currentRouter)

  if (newRtr)
  {
    Dout(dc::debug|flush_cf, rt->nodeName()<<" handover - new router global is "<<newRtr-
>prefix()
        <<" link is "<<newRtr->re.lock()->addr());

    unsigned int ifIndex = newRtr->re.lock()->ifIndex();
    //Check for return home case i.e. if pha is now on link or newRtr == pha or
    //pha reachable direct
    if (mipv6cdsMN->primaryHA() == newRtr)
    {
      if (!mipv6cdsMN->bulEmpty())
      {
```

```
    Dout(dc::notice|dc::mipv6|dc::mobile_move,  rt->nodeName()<<" "<<nd->simTime()
        <<" Returning home case detected");
    returnHome();
  }
  else
    Dout(dc::mipv6, rt->nodeName()<<" "<<nd->simTime()<<" went back home and"
        <<" no BUL entries so must have never moved i.e. false movement"
        <<" detection from missedRtrAdv not sending BU to HA at all");
}
else if (mipv6cdsMN->primaryHA())
{
  InterfaceEntry *ie = ift->interfaceByPortNo(ifIndex);
  ///Form current care of addr regardless of what type of handover
  ipv6_addr coa = mipv6cdsMN->formCareOfAddress(newRtr, ie);
  cout << "At " << nd->simTime() << " sec, "<< rt->nodeName() << endl;
  //mipv6cdsMN->setFutureCoa(coa);

  //Make sure coa is already assigned i.e. we've seen the rtrAdv from newRtr
  //and processed it in processRtrAd.
  bool assigned = false;
  if (rt->odad())
  {
    assert(ie->ipv6()->addrAssigned(coa)||ie->ipv6()->tentativeAddrAssigned(coa));
    if (ie->ipv6()->addrAssigned(coa)||ie->ipv6()->tentativeAddrAssigned(coa))
      assigned = true;
  }
  //else if (ie->ipv6()->addrAssigned(coa))
        //assigned = true;
```

```
//----------------add code-------------------
```

```
    else
    {

        const ipv6_prefix& pref = newRtr.get()->prefix();
        IPv6Address coaObj(pref);

        if(ie->ipv6()->matchPrefix(coaObj,pref.length,false)!=IPv6_ADDR_UNSPECIFIED)
        {
                coa = ie->ipv6()->matchPrefix(coaObj,pref.length,false);
                assigned = true;
        }
    }
//----------------add code------------------
    if (assigned)
    {
//----------------add code------------------
        mipv6cdsMN->setFutureCoa(coa);
//----------------add code------------------
     //newRtr could be passed in as some places may need it but then DAD dup
     //success bit is missing this. Convert this to a callback too?
     sendBU(coa);

     if ( mob->signalingEnhance() == CellResidency )
     {
      if ( mob->linkDownTime )
      {
       mob->handoverDelay = nd->simTime() - mob->linkDownTime;
       mob->linkDownTime = 0;
      }
     }
```

```
      if ( mob->isEwuOutVectorHODelays() )
        mob->recordHODelay( nd->simTime() );
    }
    else
    {
      //Not assigned at all yet unless DAD has finished (it would be in
      //tentativeAddr in this case)
      Dout(dc::mipv6|dc::neighbour_disc|flush_cf, rt->nodeName()<<" "<<nd->simTime()
          <<ifIndex<<" waiting for dad completion before sending BU for coa "
          <<coa);
    }

  } //if primary HA exists
  else
  {
    //we'll remove oldRtr  at end of func if no primaryHA.
  }

  mipv6cdsMN->setMovementDetected(false);
 } //if newRtr exists

 relinquishRouter(oldRtr, newRtr);

 if ( mob->isEwuOutVectorHODelays() )
  mob->setLinkUpTime(0); // clear the link up time
}
…
…
…
```

```
 //----------------add code------------------
void MIPv6NDStateHost::FutureCoa(const ipv6_addr& ncoa)
{
        mipv6cdsMN->setFutureCoa(ncoa);
}

 //----------------add code------------------
// void MIPv6NDStateHost::enterState(void)
// {}

// void MIPv6NDStateHost::leaveState(void)
// {}

// void MIPv6NDStateHost::initialiseInterface(size_t ifIndex)
// {}

// void MIPv6NDStateHost::disableInterface(size_t ifIndex)
// {}

// bool MIPv6NDStateHost::valRtrAd(RA* ad)
// {
//   return false;
// }

} // end namesapce MobileIPv6
```

5. MIPv6NDStateHost.h

…

…

…

```
public:
  ///gcc34 does not allow friendship to be granted to a protected function of base class? Guess
friendship and inheritance are orthogonal
  ///Need access to sendBU
  //friend void IPv6NeighbourDiscovery::NDStateHost::dupAddrDetSuccess(NDTimer* tmr);
  void sendBU(const ipv6_addr& ncoa);
```

```
//----------------add code-------------------
  void FutureCoa(const ipv6_addr& ncoa);
//----------------add code-------------------
```

```
protected:
```

…

…

…

6. Netconf2.dtd

```
…
…
…
<!ELEMENT local (interface*, (route?| tunnel?| sourceRoute?|mobileConfig?)*)>
<!ATTLIST local
        node NMTOKEN #REQUIRED
        routePackets (on|off) "off"
        forwardSitePackets (on|off) "on"
        optimisticDAD (on|off) "off"
        A-DAD (on|off) "off"
        ARDADRole (None|Node|XServer) "None"
        mobileIPv6Support (on|off) "off"
        mobileIPv6Role (None|HomeAgent|MobileNode) "None"
        routeOptimisation (on|off) "on"
        returnRoutability (on|off) "off"
        earlyBU (on|off) "off"
            signalingEnhance (None|Direct|CellResidency) "None"
        respondBindingRequest (on|off) "off"
        eagerHandover (on|off) "off"
        hierarchicalMIPv6Support (on|off) "off"
        map (on|off) "off"
        mapMode (Basic|Extended) "Basic"
        mapMNMAYSetRoCAAsSource (on|off) "on"
        mapMNMUSTSetRoCAAsSource (on|off) "on"
        mapReverseTunnel (on|off) "on"
        edgeHandoverType CDATA ""
        ewuOutVectorHODelays (on|off) "off"
        >
```

7. XMLOmnetParser.cc

…

…

…

```
void XMLOmnetParser::parseNodeAttributes(RoutingTable6* rt, cXMLElement* ne)
{
  rt->IPForward = getNodeProperties(ne, "routePackets") == XML_ON;
  rt->forwardSitePacket = getNodeProperties(ne, "forwardSitePackets") == XML_ON;
  rt->setODAD(getNodeProperties(ne, "optimisticDAD") == XML_ON);
  if (rt->odad())
    Dout(dc::notice|flush_cf, rt->nodeName()<<" ODAD is on");
```

```
  //----------------add code-------------------
  rt->setADAD(getNodeProperties(ne, "A-DAD") == XML_ON);
  if (getNodeProperties(ne,"ARDADRole") == string("XServer"))
     rt->arrole = RoutingTable6::XSERVER;
   else if (getNodeProperties(ne,"ARDADRole") == string("Node"))
     rt->arrole = RoutingTable6::NODE;
   else
     rt->arrole = RoutingTable6::NONE;

  assert((rt->odad() && rt->ardad()) == false);

  if (rt->ardad())
    Dout(dc::notice|flush_cf, rt->nodeName()<<" AR-DAD is on");
  //----------------add code-------------------
```

```
#ifdef USE_MOBILITY
  rt->mipv6Support = getNodeProperties(ne, "mobileIPv6Support") == XML_ON;
  if (rt->mobilitySupport() && version() >= 3)
```

```
  {
    if (getNodeProperties(ne,"mobileIPv6Role") == string("HomeAgent"))
      rt->role = RoutingTable6::HOME_AGENT;
    else if (getNodeProperties(ne,"mobileIPv6Role") == string("MobileNode"))
      rt->role = RoutingTable6::MOBILE_NODE;
    else
      rt->role = RoutingTable6::CORRESPONDENT_NODE;
  }
…
…
…
```

8. RoutingTable6.h

…

…

…

```
bool awayFromHome() const;

#endif //USE_MOBILITY

  bool odad() const { return odadSupport; }
```

```
  //----------------add code------------------
  bool isNode() const
    {
      return arrole == NODE;
    }
  bool isXServer() const
    {
      return arrole == XSERVER;
    }
  bool ardad() const
    {
      return ((arrole == NODE) || (arrole == XSERVER));
    }
  bool adad() const { return adadSupport; }
  void setADAD(bool a) { adadSupport = a; }
  //----------------add code------------------
```

```
  void setODAD(bool o) { odadSupport = o; }

#ifdef USE_HMIP
```

```
  /// Support for HMIPv6 for Routers and Hosts
  bool hmipSupport() const { return hmipv6Support; }

  void setHmipSupport(bool hmip) { hmipv6Support = hmip; }

  void setMapSupport(bool map) { mapSupport = map; }

  bool isMAP()
    {
      return mapSupport;
    }
…
…
…
```

9. ICMPv6NDMessage.cc

…

…

…

const int IPv6NeighbourDiscovery::OVERRIDE_MASK = 0x20000000;

```
 //----------------add code-------------------
const int IPv6NeighbourDiscovery::ARDADFLAG_MASK = 0x10000000;
 //----------------add code-------------------
```

const int IPv6NeighbourDiscovery::MANAGED_MASK = 0x800000;

const int IPv6NeighbourDiscovery::OTHER_MASK = 0x400000;

…

…

…

10. ICMPv6NDMessage.h

…

…

…

```
extern const int OTHER_MASK;
```

```
 //----------------add code-------------------
extern const int ARDADFLAG_MASK;
 //----------------add code-------------------
```

```
#ifdef USE_MOBILITY
extern const int HOMEAGENT_MASK;
#endif
```

…

…

…

```
void setTargetAddr(const ipv6_addr target) { setAddress(target, 0); }
 ipv6_addr targetAddr() { return address(0); }
 //@}
```

```
//----------------add code-------------------

 bool arDADFlag() const { return optInfo() & ARDADFLAG_MASK; }
 void setarDADFlag(bool arDADFlag)
  {
   if (arDADFlag)
     setOptInfo(optInfo() | ARDADFLAG_MASK);
   else
     setOptInfo(optInfo() & ~ARDADFLAG_MASK);
  }
```

```
 //----------------add code------------------
};
…
…
…
void setFlags(bool isRouter, bool solicited, bool override)
    {
      setIsRouter(isRouter);
      setSolicited(solicited);
      setOverride(override);
    }
 //@}
```

```
//----------------add code------------------

 bool arDADFlag() const { return optInfo() & ARDADFLAG_MASK; }
 void setarDADFlag(bool arDADFlag)
   {
     if (arDADFlag)
       setOptInfo(optInfo() | ARDADFLAG_MASK);
     else
       setOptInfo(optInfo() & ~ARDADFLAG_MASK);
   }
 //----------------add code------------------
```

```
…
…
…
```

11. IPv6Forword.cc

…

…

…

```
void IPv6Forward::endService(cMessage* theMsg)
{
  std::auto_ptr<IPv6Datagram> datagram(check_and_cast<IPv6Datagram*> (theMsg));

  assert(datagram->inputPort() < (int)ift->numInterfaceGates());

// {{{ localdeliver

  //Disabled for now. Don't know why some packets when they leave this function
  //still display 0 source address when they shouldn't. Recording at prerouting
  //and output
  //printRoutingInfo(routingInfoDisplay, datagram.get(), rt->nodeName());
```

```
  //----------------add code-------------------
  IPv6Datagram* copy = datagram->dup();
  if((rt->ardad()) && (datagram->transportProtocol()==IP_PROT_IPv6_ICMP))
  {
            send(copy, "localOut");
if (!datagram->destAddress().isMulticast())
     return;
       /*if (copy->decapsulate() != NULL)
       {
            ICMPv6Message* icmpmsg = check_and_cast<ICMPv6NDMNgbrSol*>
(std::auto_ptr<ICMPv6Message> copy->decapsulate());
```

```
        if ((icmpmsg->type()==ICMPv6_NEIGHBOUR_SOL) || (icmpmsg-
>type()==ICMPv6_NEIGHBOUR_AD))
        {
                send(copy, "localOut");
                  return;
        }
        }

        bool ar_dad=false;
        if (icmpmsg->type()==ICMPv6_NEIGHBOUR_SOL)
        {
                        ICMPv6NDMNgbrSol*
ns(check_and_cast<ICMPv6NDMNgbrSol*>(copy->decapsulate()));
                        ar_dad = ns->arDADFlag();

            }
      if (icmpmsg->type()==ICMPv6_NEIGHBOUR_AD)
        {
                        ICMPv6NDMNgbrAd*
na(check_and_cast<ICMPv6NDMNgbrAd*>(copy->decapsulate()));
                        ar_dad = na->arDADFlag();
            }
     if (ar_dad)
            {
                        //IPv6Datagram* copy = datagram->dup();
                        send(copy, "localOut");
                return;
            }*/
  }else
  //----------------add code------------------
```

```
  if (rt->localDeliver(datagram->destAddress()))
  {
    Dout(dc::forwarding|flush_cf, rt->nodeName()<<":"<<hex<<datagram->inputPort()<<dec<<"
"
         <<simTime()<<" received packet from "<<datagram->srcAddress()
         <<" dest="<<datagram->destAddress());

    IPv6Datagram* copy = datagram->dup();

    //This condition can occur when upper layer originates packets without
    //specifying a src address destined for a multicast destination or local
    //destination so the packet is missing the src address. This is a bit
    //dodgy but the only other solution would be to enforce the app layer to
    //choose a src address.
    if (datagram->srcAddress() == IPv6_ADDR_UNSPECIFIED &&
        datagram->inputPort() == -1)
      copy->setSrcAddress(ift->interfaceByPortNo(0)->ipv6()->inetAddrs[0]);

    send(copy, "localOut");
    //Check if it is a multicast packet that we need to forward
    if (!datagram->destAddress().isMulticast())
      return;
  }
…
…
…
```

12. IPv6Interfacedata.cc

…

…

…

```
bool IPv6InterfaceData::tentativeAddrAssigned(const ipv6_addr& addr) const
{
  IPv6Addresses::const_iterator it = std::find(tentativeAddrs.begin(),tentativeAddrs.end(),addr);
  return it!=tentativeAddrs.end();
}
```

```
  //----------------add code------------------
size_t IPv6InterfaceData::findIPFromWaitPool(const ipv6_addr& waitpool,bool NCoA)
{
            size_t k=-1;
            if (NCoA)
             for (size_t i = 0; i < WaitPoolncoa.size(); i++)
               if (WaitPoolncoa[i].higherNbBits(IPv6_ADDR_BITLENGTH)==waitpool)
               {
                        k=i;
                        break;
               }
            else
             for (size_t i = 0; i < WaitPooldad.size(); i++)
               if (WaitPooldad[i].higherNbBits(IPv6_ADDR_BITLENGTH)==waitpool)
               {
                        k=i;
                        break;
               }
            return k;
}
```

```
size_t IPv6InterfaceData::findIPFromWaitPool(IPv6Address waitpool,bool NCoA)
{
        size_t k=-1;
        if (NCoA)
         for (size_t i = 0; i < WaitPoolncoa.size(); i++)
         {
          if (WaitPoolncoa[i].address().compare(waitpool.address()) == 0)
          {
                k=i;
                break;
          }
         }
        else
         for (size_t i = 0; i < WaitPooldad.size(); i++)
         {
          if (WaitPooldad[i].address().compare(waitpool.address()) == 0)
          {
                k=i;
                break;
          }
         }
        return k;
}
bool IPv6InterfaceData::findIPFromNCoA(const ipv6_addr& ncoa) const
{
  IPv6Addresses::const_iterator it = std::find(NCoAPool.begin(),NCoAPool.end(),ncoa);
  return it!=NCoAPool.end();
}
  //----------------add code------------------
```

13. IPv6Interfacedata.h

…

…

…

```
///Return the shortest validLifetime assigned to an address on this iface
unsigned int minValidLifetime();

///Return an assigned or tentative address that matches the specified prefix
IPv6Address matchPrefix(const ipv6_addr& prefix, unsigned int prefixLength,
                        bool tentative = false);
```

```
//----------------add code------------------
void removeFromPool(const IPv6Address& addr,bool NCoA=true)
{
 if (NCoA)
   removeAddrFromArray(NCoAPool,addr);
 else
 {
   removeAddrFromArray(WaitPoolncoa,addr);
   removeAddrFromArray(WaitPooldad,addr);
 }
}

size_t findIPFromWaitPool(const ipv6_addr& waitpool,bool NCoA=true);
size_t findIPFromWaitPool(IPv6Address waitpool,bool NCoA=true);

bool findIPFromNCoA(const ipv6_addr& ncoa) const;
//----------------add code------------------
```

```
public:
 // XXX naked public data members here MUST BE WRAPPED INTO GET/SET
FUNCTIONS!!!!!! --AV

 // Type of network interface (Ethernet, Point to Point)  XXX OUT!!!!! --AV
 const char* encap(void);

 /**
  * @todo Change to use shared_ptrs of ipv6_prefix (once that has an interface
  * resembling IPv6Address) and thus STL list container.  This enables other
  * protocols that need to know when an addr assigned to the node is still
  * valid by maintaining a weak_ptr to it.  For now when an addr lifetime
  * expires we will have to set all other pointers to null manually which is
  * tedioius and error prone.
  *
  */

 IPv6Addresses inetAddrs;

 /// tentative address for duplicate address detection
 /// This data structure serves 4 purposes 1. Store manually configured address
 /// parsed from XML network configuration. 2. ND uses it to determine which
 /// addresses are tentative during Neighbour Discovery and DAD procedure
 /// 3. DAD started on addresses inside here right after link local addr is
 /// assigned ( detectDupOther) only once 4. Subsequent tentative addr derived
 /// from router prefixes are stored here but DAD is initiated for them from
 /// elsewhere.  (This is to prevent DAD from recurring on addresses that are
 /// currently undergoing DAD).
 IPv6Addresses tentativeAddrs;
```

```
//----------------add code------------------
IPv6Addresses NCoAPool;
IPv6Addresses WaitPoolncoa;
IPv6Addresses WaitPooldad;
IPv6Addresses prefixIP;
//----------------add code------------------
```

```
///@name Node Configuration Variables
//@{
short curHopLimit;
unsigned int retransTimer;
unsigned int baseReachableTime;
…
…
…
```

14. MIPv6test4.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE netconf SYSTEM "../../Etc/netconf2.dtd">

<netconf>

<!--
"Mos.lot:MobileMove:notice:custom:Ping6:MIPv6MissedAdv:AddrResln:MIPv6:AddressTimer:
PrefixTimer:RouterTimer:IPv6AddressDeAlloc:Routering:Forwarding:NeighbourDisc:RouterDis
c:debug:IPv6">-->

<local node="client1" mobileIPv6Support="on" ARDADRole="None"
mobileIPv6Role="MobileNode" hierarchicalMIPv6Support="off" optimisticDAD="off" A-
DAD="off"  respondBindingRequest="on" ewuOutVectorHODelays="on">
        <interface name="wlan0" HostDupAddrDetectTransmits="1">
    <!--<inetAddr>3018:ffff:0:0:0:0:4444/64</inetAddr>-->
  </interface>
  <mobileConfig>
        <mobileNode>
          <staticConfig homeAddrIface="wlan0" homeAgentAddr="3018:ffff:0:0:0:0:1/64"/>
        </mobileNode>
  </mobileConfig>
 </local>

 <local node="PCconnect" mobileIPv6Support="off" optimisticDAD="off"
mobileIPv6Role="None">
   <interface name="eth0" HostDupAddrDetectTransmits="1">
     <!--<inetAddr>fe80:0:0:0:0:0:2222:2222/64</inetAddr>-->
     <inetAddr>3021:ffff:0:0:0:0:abc2/64</inetAddr>
```

```
   </interface>
  </local>

  <local node="XServer1" mobileIPv6Support="on" optimisticDAD="off"
mobileIPv6Role="MobileNode" ARDADRole="None" A-DAD="off" >
   <interface name="wlan0" HostDupAddrDetectTransmits="1">
            <inetAddr>3018:ffff:0:0:0:0:1111/64</inetAddr>
   </interface>
  </local>

  <local node="XServer2" mobileIPv6Support="on" optimisticDAD="off"
mobileIPv6Role="MobileNode" ARDADRole="None" A-DAD="off" >
   <interface name="wlan0" HostDupAddrDetectTransmits="1">
            <inetAddr>3019:ffff:0:0:0:0:1111/64</inetAddr>
   </interface>
  </local>

  <local node="XServer3" mobileIPv6Support="on" optimisticDAD="off"
mobileIPv6Role="MobileNode" ARDADRole="None" A-DAD="off" >
   <interface name="wlan0" HostDupAddrDetectTransmits="1">
            <inetAddr>3020:ffff:0:0:0:0:1111/64</inetAddr>
   </interface>
  </local>

<!-- routing table configuration -->

<local node="haa" routePackets="on" mobileIPv6Support="on" mobileIPv6Role="HomeAgent"
optimisticDAD="off">
         <interface name="eth0" AdvSendAdvertisements="on" AdvHomeAgent="on"
HostDupAddrDetectTransmits="1">
```

```
    <inetAddr>3018:ffff:0:0:0:0:0:1/64</inetAddr>
    <AdvPrefixList>
        <AdvPrefix AdvOnLinkFlag="on"
AdvRtrAddrFlag="on">3018:ffff:0:0:0:0:0:1/64</AdvPrefix>
    </AdvPrefixList>
  </interface>

  <interface name="eth1" AdvSendAdvertisements="on" HostDupAddrDetectTransmits="1">
    <inetAddr>4018:ffff:0:0:0:0:0:1/64</inetAddr>
  </interface>

  <interface name="eth2" AdvSendAdvertisements="on" HostDupAddrDetectTransmits="1">
    <inetAddr>4020:ffff:0:0:0:0:0:1/64</inetAddr>
  </interface>

  <interface name="eth3" AdvSendAdvertisements="on" HostDupAddrDetectTransmits="1">
        <inetAddr>3021:ffff:0:0:0:0:0:1/64</inetAddr>
  </interface>

   <route>
    <routeEntry routeIface="eth0" routeDestination="3018:ffff:0:0:0:0:0:0/64"/>
    <routeEntry routeIface="eth1" routeDestination="3019:ffff:0:0:0:0:0:0/64"
      routeNextHop="4018:ffff:0:0:0:0:0:2/64"/>
    <routeEntry routeIface="eth2" routeDestination="3020:ffff:0:0:0:0:0:0/64"
      routeNextHop="4020:ffff:0:0:0:0:0:2/64"/>
    <routeEntry routeIface="eth3" routeDestination="3021:ffff:0:0:0:0:0:0/64"/>
   </route>

</local>
```

```
<local node="router2" routePackets="on" mobileIPv6Support="on"
mobileIPv6Role="HomeAgent" optimisticDAD="off">
   <interface name="eth0" AdvSendAdvertisements="on" AdvHomeAgent="on"
HostDupAddrDetectTransmits="1">
     <inetAddr>3019:ffff:0:0:0:0:0:1/64</inetAddr>
     <AdvPrefixList>
          <AdvPrefix AdvOnLinkFlag="on"
AdvRtrAddrFlag="on">3019:ffff:0:0:0:0:0:1/64</AdvPrefix>
     </AdvPrefixList>
   </interface>

   <interface name="eth1" AdvSendAdvertisements="on" HostDupAddrDetectTransmits="1">
     <inetAddr>4018:ffff:0:0:0:0:0:2/64</inetAddr>
   </interface>

   <interface name="eth2" AdvSendAdvertisements="on" HostDupAddrDetectTransmits="1">
     <inetAddr>4019:ffff:0:0:0:0:0:1/64</inetAddr>
   </interface>

   <route>
     <routeEntry routeIface="eth0" routeDestination="3019:ffff:0:0:0:0:0:0/64"/>
     <routeEntry routeIface="eth1" routeDestination="3018:ffff:0:0:0:0:0:0/64"
          routeNextHop="4018:ffff:0:0:0:0:0:1/64"/>
     <routeEntry routeIface="eth1" routeDestination="3021:ffff:0:0:0:0:0:0/64"
       routeNextHop="4018:ffff:0:0:0:0:0:1/64"/>
     <routeEntry routeIface="eth2" routeDestination="3020:ffff:0:0:0:0:0:0/64"
       routeNextHop="4019:ffff:0:0:0:0:0:2/64"/>
   </route>
</local>
```

```
<local node="router3" routePackets="on" mobileIPv6Support="on"
mobileIPv6Role="HomeAgent" optimisticDAD="off">
   <interface name="eth0" AdvSendAdvertisements="on" AdvHomeAgent="on"
HostDupAddrDetectTransmits="1">
     <inetAddr>3020:ffff:0:0:0:0:0:1/64</inetAddr>
     <AdvPrefixList>
          <AdvPrefix AdvOnLinkFlag="on"
AdvRtrAddrFlag="on">3020:ffff:0:0:0:0:0:1/64</AdvPrefix>
     </AdvPrefixList>
   </interface>

   <interface name="eth1" AdvSendAdvertisements="on" HostDupAddrDetectTransmits="1">
     <inetAddr>4020:ffff:0:0:0:0:0:2/64</inetAddr>
   </interface>

   <interface name="eth2" AdvSendAdvertisements="on" HostDupAddrDetectTransmits="1">
     <inetAddr>4019:ffff:0:0:0:0:0:2/64</inetAddr>
   </interface>

   <route>
     <routeEntry routeIface="eth0" routeDestination="3020:ffff:0:0:0:0:0:0/64"/>
     <routeEntry routeIface="eth1" routeDestination="3018:ffff:0:0:0:0:0:0/64"
          routeNextHop="4020:ffff:0:0:0:0:0:1/64"/>
     <routeEntry routeIface="eth1" routeDestination="3021:ffff:0:0:0:0:0:0/64"
          routeNextHop="4020:ffff:0:0:0:0:0:1/64"/>
     <routeEntry routeIface="eth2" routeDestination="3019:ffff:0:0:0:0:0:0/64"
          routeNextHop="4019:ffff:0:0:0:0:0:1/64"/>
   </route>
</local>
```

```
 <misc>
  <ObjectMovement>
    <MovingNode NodeName="client1" startTime="0">
      <move moveToX="470" moveToY="600" moveSpeed="6" moveXFirst="off"/>
          <move moveToX="120" moveToY="118" moveSpeed="6" moveXFirst="off"/>
    </MovingNode>
  </ObjectMovement>
 </misc>


</netconf>
```

15. MIPv6test4.ned

```
//
// Copyright (C) 2001, 2003 Monash University, Australia
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
//



import
    "EtherHub",
    "Router6",
    "UDPNode",
    "WorldProcessor",
    "WirelessAccessPoint",
    "WirelessMobileNode",
    "Cables";
```

```
//
// A MIPv6 test network containing a server, several routers with
// wireless access points, and a laptop as mobile node.
//
// Can be used to test handovers between access points.
// Mobile node is pinging the server while it's moving.
//
module MIPv6test4
    submodules:
        worldProcessor: WorldProcessor;
            display: "p=468,56;i=misc/sun";
        client1: MobileNode;
            parameters:
                //# routingFile = "client1.irt",
                IPForward = false;
            gatesizes:
                wlin[1],
                wlout[1];
            display: "p=120,118;i=device/laptop";
        PCconnect: UDPNode;
            parameters:
                //# routingFile = "server4.irt",
                IPForward = false;
            gatesizes:
                in[1],
                out[1];
            display: "p=447,153;i=device/pc2";
        XServer1: MobileNode;
            parameters:
```

```
        //# routingFile = "server4.irt",
        IPForward = false;
    gatesizes:
        wlin[1],
        wlout[1];
    display: "p=207,81;i=device/pc2";
XServer2: MobileNode;
    parameters:
        //# routingFile = "server4.irt",
        IPForward = false;
    gatesizes:
        wlin[1],
        wlout[1];
    display: "p=103,529;i=device/pc2";
XServer3: MobileNode;
    parameters:
        //# routingFile = "server4.irt",
        IPForward = false;
    gatesizes:
        wlin[1],
        wlout[1];
    display: "p=487,465;i=device/pc2";
haa: Router6;
    gatesizes:
        in[4],
        out[4];
    display: "p=232,288;i=device/router";
ap1: AccessPoint;
    gatesizes:
        in[1],
```

```
            out[1];
        display: "p=190,142;i=device/accesspoint";
    ap2: AccessPoint;
        gatesizes:
            in[1],
            out[1];
        display: "p=170,546;i=device/accesspoint";
    ap3: AccessPoint;
        gatesizes:
            in[1],
            out[1];
        display: "p=486,418;i=device/accesspoint";
    router2: Router6;
        gatesizes:
            in[3],
            out[3];
        display: "p=232,412;i=device/router";
    router3: Router6;
        gatesizes:
            in[3],
            out[3];
        display: "p=352,348;i=device/router";
connections nocheck:

                // Correct for ping test

    haa.out[1] --> intranetCable --> router2.in[1];
    haa.in[1] <-- intranetCable <-- router2.out[1];

    haa.out[2] --> intranetCable --> router3.in[1];
```

```
        haa.in[2] <-- intranetCable <-- router3.out[1];

        router2.out[2] --> intranetCable --> router3.in[2];
        router2.in[2] <-- intranetCable <-- router3.out[2];

        router3.out[0] --> intranetCable --> ap3.in[0];
        router3.in[0] <-- intranetCable <-- ap3.out[0];

        haa.out[0] --> intranetCable --> ap1.in[0];
        haa.in[0] <-- intranetCable <-- ap1.out[0];

        haa.out[3] --> intranetCable --> PCconnect.in[0];
        haa.in[3] <-- intranetCable <-- PCconnect.out[0];

        router2.out[0] --> intranetCable --> ap2.in[0];
        router2.in[0] <-- intranetCable <-- ap2.out[0];

    display: "b=613,705";
endmodule


network MosNetwork : MIPv6test4
endnetwork
```

16. Omnet.ini

```
#% old-wildcards
[General]
preload-ned-files=*.ned @../../nedfiles2.lst
num-rngs=1
;Was omnetpp.ini but since that's been modified so much now have extracted this out instead.

total-stack-kb=7535
ini-warnings = no
warnings = no
sim-time-limit = 320
debug-on-errors=false
;fname-append-host=yes

[Cmdenv]
default-run=1
module-messages=no
event-banners=no
message-trace=no

[Tkenv]
print-banners=no
use-mainwindow=yes
breakpoints-enabled = no
animation-speed = 1.0

[Run 1]
seed-0-mt=1111100000
;seed-0-mt=1111111111
```

```
;seed-0-mt=1111122222
;seed-0-mt=1111133333
;seed-0-mt=1111144444
;seed-0-mt=1111155555
;seed-0-mt=1111166666
;seed-0-mt=1111177777
;seed-0-mt=1111188888
;seed-0-mt=1111199999
network = MosNetwork
MosNetwork.client1.linkLayers[*].NWIName="WirelessEtherModule"
MosNetwork.XServer1.linkLayers[*].NWIName="WirelessEtherModule"
MosNetwork.XServer2.linkLayers[*].NWIName="WirelessEtherModule"
MosNetwork.XServer3.linkLayers[*].NWIName="WirelessEtherModule"
MosNetwork.ap1.ds[*].NWIName="EtherModuleAP"
MosNetwork.ap2.ds[*].NWIName="EtherModuleAP"
MosNetwork.ap3.ds[*].NWIName="EtherModuleAP"
;MosNetwork.ap4.ds[*].NWIName="EtherModuleAP"

MosNetwork.*.IPv6routingFile = xmldoc("MIPv6test4.xml")

;MosNetwork.client1.pingApp.startTime=5
;MosNetwork.client1.pingApp.stopTime=700
;MosNetwork.client1.pingApp.destAddr = "3088:AABB:1122:3456:0:0:0:1"
;MosNetwork.client1.pingApp.srcAddr = "3018:ffff:0:0:0:0:0:4444"
;MosNetwork.client1.pingApp.destAddr = "3021:ffff:0:0:0:0:0:abc2"
;MosNetwork.client1.pingApp.interval = 0.1

;MosNetwork.PCconnect.pingApp.startTime=5
;MosNetwork.PCconnect.pingApp.stopTime=700
;MosNetwork.PCconnect.pingApp.destAddr = "3011:bbbb:3333:6666:0:0:0:1"
```

```
;MosNetwork.PCconnect.pingApp.destAddr = "3020:ffff:0:0:0:0:0:2"
;MosNetwork.PCconnect.pingApp.srcAddr = "3011:bbbb:3333:6666:abc2:abc2:abc2:abc2"
;MosNetwork.PCconnect.pingApp.destAddr = "client1"
;MosNetwork.PCconnect.pingApp.interval = 0.1

MosNetwork.PCconnect.pingApp.startTime=140
MosNetwork.PCconnect.pingApp.stopTime=150
MosNetwork.PCconnect.pingApp.destAddr = "XServer3"
MosNetwork.PCconnect.pingApp.interval = 0.5



;MosNetwork*.routing6.ra[2].RoutingAlgorithmType="NonExistent"
;MosNetwork*.routing6.**.RoutingAlgorithmType="RoutingAlgorithmStatic"
;MosNetwork*.routing6.ra[1].RoutingAlgorithmType="RoutingProtocolRIP"
;MosNetwork.*.routing6.RACount = 2

*.ap1.chann = 1
*.ap2.chann = 1
*.ap3.chann = 1
;*.ap4.chann = 6
**.networkInterface.txPower = 11

;; Videostream
;MosNetwork.client1.numUdpApps = 1
;MosNetwork.client?.udpAppType = "UDPVideoStreamCli"
;MosNetwork.client1.udpApp[*].serverAddress = "3011:BBBB:3333:6666:ac24:0aff:fe11:bba"
;MosNetwork.client?.udpApp[*].serverPort = 3088
;MosNetwork.client?.udpApp[*].startTime = 8

;;Videostream server
```

```
;MosNetwork.server4.numUdpApps = 1
;MosNetwork.server4.udpAppType = "UDPVideoStreamSvr"
;MosNetwork.server4.udpApp[0].videoSize = 1e8
;MosNetwork.server4.udpApp[0].serverPort = 3088
;output-vector-file = MIPv6Network-r1.vec

[Parameters]
**.networkLayer.proc.forwarding.routingInfoDisplay = false

**.networkLayer.proc.ICMP.icmpv6Core.icmpRecordRequests = false
**.networkLayer.proc.ICMP.icmpv6Core.icmpRecordStart = 0
**.networkLayer.proc.ICMP.icmpv6Core.replyToICMPRequests = true
**.routingTable6.displayIfconfig = true



;;default global movement info
**.mobilityHandler.moveXmlConfig=xmldoc("MIPv6test4.xml", "netconf")

;;default global wireless network info
**.networkInterface.nwiXmlConfig=xmldoc("MIPv6test4.xml", "netconf")
**.networkInterface[*].nwiXmlConfig=xmldoc("MIPv6test4.xml", "netconf")

;;Default empty xml config file
**.IPv6routingFile = xmldoc("MIPv6test4.xml")
include ../../Etc/default2.ini
```

17. Default2.ini

```
#% old-wildcards

[Parameters]
;;Default ping6 settings
**.pingApp.stopTime = 0 ;-> forever
**.pingApp.startTime = 1
**.pingApp.interval = 1
**.pingApp.packetSize = 56
**.pingApp.hopLimit = 64
**.pingApp.count = 0 ;-> forever
**.pingApp.destAddr = ""
**.pingApp.srcAddr = ""
**.pingApp.printPing = 1
;;Default ICMP settings
**.networkLayer.proc.ICMP.icmpv6Core.icmpRecordRequests = false
**.networkLayer.proc.ICMP.icmpv6Core.icmpRecordStart = 0
**.networkLayer.proc.ICMP.icmpv6Core.replyToICMPRequests = true

;;Default MIPv6 settings
;**.networkLayer.proc.mobility.homeAgent = "0:0:0:0:0:0:0:0"
**.networkLayer.proc.mobility.homeAgent = ""

;;Default UDP apps
**.numUdpApps = 0
**.udpAppType = "UDPBasicApp"

;;Default UDP VideoStream client and server setttings
**.udpApp[*].waitInterval = uniform(.005, .01)
```

```
**.udpApp[*].packetLen = uniform(12000, 15000)
**.udpApp[*].videoSize = 650e6
**.udpApp[*].startTime = 0 ; Time to request video stream from server
**.udpApp[*].localPort = 20403 ; some arbitrary number
**.udpApp[*].serverAddress = ""
**.udpApp[*].serverPort = 3088

;;Default TCP apps
**.numTcpApps = 0
**.tcpAppType = "TCPSessionApp"

;; Default TCP settings
**.tcp.sendQueueClass="TCPVirtualDataSendQueue"
**.tcp.receiveQueueClass="TCPVirtualDataRcvQueue"
**.tcp.tcpAlgorithmClass="TCPReno"
**.tcp.recordStats=false

;UDPNode requires IPForward perhaps for IPv4 layer?
**.IPForward = false

**.max_longitude = 1640
**.max_latitude = 1640
**.wlan_speed = 11

;ProcessorManager emulate serial CPU processing
**.numOfProcessors = 1

;;Processing delays (applies to Router6 only)
**.ICMP.procDelay = 0 s
**.fragmentation.procDelay = 0.2 us
```

```
**.multicast.procDelay = 0.5 us
**.output[*].procDelay = 1 us
**.send.procDelay = 0 s
**.tunneling.procDelay = 0 s
**.localDeliver.procDelay = 0 s
**.preRouting.procDelay = 0.2 us
**.forwarding.procDelay = 1 us
**.inputQueue.procDelay = 0 s
;;*.router*.icmp.procDelay = 0 s
;;*.ha*.icmp.procDelay = 0 s
;;*.ar*.icmp.procDelay = 0 s
**.networkLayer.proc.forwarding.routingInfoDisplay = true

;;Routing modules
**.routing6.ra[*].RoutingAlgorithmType="RoutingAlgorithmStatic"
**.routing6.rp[*].RoutingProtocolType="RoutingProtocolNone"
**.routing6.RACount = 0
**.routing6.RPCount = 0
**.routingTable6.displayIfconfig = true

;Dynamic Topology Models
**.dtb.topoFilename = "retrieve via XML config unless used separately from IPv6Suite"

;Default AP parameter
**.chann = intuniform(1, 14)
**.beaconPeriod = 0.1
**.authWaitEntryTimeout = 2;
**.authEntryTimeout = 2;
**.assEntryTimeout = 120;
**.consecFailedTransLimit = 3;
```

```
**.ds[*].NWIName="EtherModuleAP"
**.beginCollectionTime = 0
**.endCollectionTime = 0
**.consecFailedTransLimit = 3

;;Default link type
**.linkLayers[*].NWIName="EtherModule"

;;Default physical layer type
**.PHYName="PHYSimple"

;;Default mobility handler type
**.MobilityName="MobilityStatic"

;;default global movement info
**.mobilityHandler.moveXmlConfig=xmldoc("empty.xml", "netconf")

;;default global wireless network info
**.networkInterface.nwiXmlConfig=xmldoc("empty.xml", "netconf")
**.networkInterface[*].nwiXmlConfig=xmldoc("empty.xml", "netconf")

;;Default empty xml config file
**.IPv6routingFile = xmldoc("empty.xml")

;;Default wireless parameters. Values come from netconf2.dtd.
**.networkInterface.ssid = "default"
**.networkInterface.pathLossExponent = 2.75
**.networkInterface.pathLossStdDev = 14.1
**.networkInterface.txPower = 100
;;Range of 838 meters
```

```
**.networkInterface.thresholdPower = -93
;;Handover range of 492 meters
**.networkInterface.hoThresholdPower = -90
**.networkInterface.probeEnergyTimeout = 0.01
**.networkInterface.probeResponseTimeout = 0.035
**.networkInterface.authenticationTimeout = 2000
**.networkInterface.associationTimeout = 2000
**.networkInterface.retry = 7
**.networkInterface.fastActiveScan = true
**.networkInterface.scanShortCircuit = true
**.networkInterface.crossTalk = true
**.networkInterface.shadowing = false
**.networkInterface.signalStrengthMaxSample = 1
**.networkInterface.channelsNotToScan = ""
**.networkInterface.address = ""
**.networkInterface.dataRate = 11000000
**.networkInterface.bandwidthRequirements = 0.5
**.networkInterface.recordStatisticVector = true
**.networkInterface.activeScan = true
**.networkInterface.channelScanTime = 0.2
**.networkInterface.bufferSize = 100000
**.networkInterface.linkUpTrigger = false
**.networkInterface.linkDownTrigger = false
**.networkInterface.registerInterface = true
**.networkInterface.queueType = "WESingleQueue"
**.networkInterface.errorRate = 0

;Default BR values
**.recordStats=false
**.brSrcModel.msgType=0
```

```
**.brSrcModel.destAddr=""
**.brSrcModel.tStart=0
**.brSrcModel.bitRate=0
**.brSrcModel.fragmentLen=0

;Default Dynamic IPv6 CBR Loader
**.numNodes = 0
**.rangeMinX = 0
**.rangeMinY = 0
**.rangeMaxX = 0
**.rangeMaxY = 0
**.srcPrefix = ""
**.destPrefix= ""
```

ภาคผนวก ค

ผลการทดลอง

ในภาคผนวกนี้แสดงผลการทดลองของทั้ง 5 กรณี ที่ทดลองทั้งหมด 10 ครั้งดังนี้

1) วิธี DAD แบบมาตรฐาน
2) วิธี A-DAD แบบไม่มีความผิดพลาด (A-DAD Ideal case)
3) วิธี A-DAD แบบไม่ได้รับเลขหมาย NCoA (A-DAD missing NCoA case) หรือแบบไม่มีวิธี A-DAD ในเครือข่าย (No A-DAD method case)
4) วิธี FR-DAD แบบไม่มีความผิดพลาด (FR-DAD Ideal case)
5) วิธี FR-DAD แบบไม่ได้รับเลขหมาย NCoA (FR-DAD missing NCoA case) หรือแบบไม่มี P-Server ในเครือข่าย (No P-Server case)

Standard DAD

	1	2	3	4	5	6	7	8	9	10	Average
Home Network											
Signal up	0.918562	0.234069	0.896445	0.276404	1.003720	0.249794	1.031740	0.737738	0.421281	0.289164	0.605892
Receive Link Local Address (FE80::)	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
Receive RA + prefix and test DAD	1.011070	1.010870	1.010890	1.108900	1.010770	1.011270	1.053310	1.010870	1.010770	1.011030	1.024975
DAD Success	2.011070	2.010870	2.010890	2.108900	2.010770	2.011270	2.053310	2.010870	2.010770	2.011030	2.024975
DAD Delay	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
Foreign network 1											
Signal up	46.586900	46.540600	46.693800	46.648800	46.521000	46.541200	46.609600	46.628100	46.674900	46.733200	46.617810
Receive RA + prefix and test DAD	46.607500	46.561800	46.714300	46.670100	46.541800	46.562800	46.631000	46.649200	46.696100	46.754100	46.638870
DAD Success	47.607500	47.561800	47.714300	47.670100	47.541800	47.562800	47.631000	47.649200	47.696100	47.754100	47.638870
Send BU	47.607500	47.561800	47.714300	47.670100	47.541800	47.562800	47.631000	47.649200	47.696100	47.754100	47.638870
Receive BU Ack	49.209300	49.164400	49.316200	49.272200	49.143400	49.165500	49.233500	49.251700	49.298300	49.356900	49.241140
DAD Delay	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
Handover Layer 3 Delay	2.622400	2.623800	2.622400	2.623400	2.622400	2.624300	2.623900	2.623600	2.623400	2.623700	2.623330
Foreign network 2											
Signal up	142.879000	142.886000	142.965000	142.986000	143.011000	142.925000	142.928000	143.016000	143.009000	142.974000	142.957900
Receive RA + prefix and test DAD	142.901000	142.907000	142.986000	143.007000	143.032000	142.946000	142.949000	143.037000	143.031000	142.995000	142.979100
DAD Success	143.901000	143.907000	143.986000	144.007000	144.032000	143.946000	143.949000	144.037000	144.031000	143.995000	143.979100
Send BU	143.901000	143.907000	143.986000	144.007000	144.032000	143.946000	143.949000	144.037000	144.031000	143.995000	143.979100
Receive BU Ack	143.942000	143.948000	144.027000	144.048000	144.073000	143.987000	143.990000	144.077000	144.072000	144.035000	144.019900
DAD Delay	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
Handover Layer 3 Delay	1.063000	1.062000	1.062000	1.062000	1.062000	1.062000	1.062000	1.061000	1.063000	1.061000	1.062000
Return to Home Network											
Signal up	239.752000	239.784000	239.783000	239.797000	239.729000	239.738000	239.831000	239.834000	239.830000	239.865000	239.794300
Receive RA + prefix and send BU	239.775000	239.805000	239.803000	239.818000	239.749000	239.759000	239.852000	239.855000	239.850000	239.886000	239.815200
Receive BU Ack	239.825460	239.855671	239.854373	239.868618	239.798765	239.808484	239.902997	239.906763	239.902134	239.937678	239.866094
Handover Layer 3 Delay	0.073460	0.071671	0.071373	0.071618	0.069765	0.070484	0.071997	0.072763	0.072134	0.072678	0.071794

A-DAD Ideal case

	1	2	3	4	5	6	7	8	9	10	Average
Home Network											
Signal up	0.912936	0.241709	0.922845	0.281844	1.012220	0.257663	1.031740	0.670074	0.428501	0.289084	0.604862
Receive Link Local Address (FE80::)	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
Receive RA + prefix and test DAD	1.321170	1.320710	1.321130	1.320890	1.320550	1.321410	1.320490	1.320990	1.320910	1.319070	1.320732
DAD Success	1.323090	1.322970	1.322610	1.322930	1.322650	1.323630	1.322910	1.322990	1.322970	1.321530	1.322828
DAD Delay	0.001920	0.002260	0.001480	0.002040	0.002100	0.002220	0.002420	0.002000	0.002060	0.002460	0.002096
Foreign network 1											
Signal up	46.532300	46.587200	46.633500	46.588700	46.590300	46.567800	46.513400	46.711000	46.606300	46.561500	46.589200
Receive RA + prefix and test DAD	46.553200	46.608100	46.654600	46.610000	46.611500	46.587700	46.534100	46.691800	46.608500	46.582400	46.604190
DAD Success	46.556300	46.611800	46.657300	46.612900	46.613800	46.590500	46.536200	46.694100	46.611500	46.585300	46.606970
Send BU	46.556300	46.611800	46.657300	46.612900	46.613800	46.590500	46.536200	46.694100	46.611500	46.585300	46.606970
Receive BU Ack	48.158500	48.213700	48.259300	48.215000	48.215700	48.192800	48.137500	48.296300	48.213500	48.186500	48.208880
DAD Delay	0.003100	0.003700	0.002700	0.002900	0.002300	0.002800	0.002100	0.002300	0.003000	0.002900	0.002780
Handover Layer 3 Delay	1.605300	1.605600	1.604700	1.605000	1.604200	1.605100	1.603400	1.604500	1.605000	1.604100	1.604690
Foreign network 2											
Signal up	142.854000	142.891000	143.001000	142.919000	142.941000	143.000000	143.017000	142.872000	142.937000	142.886000	142.931800
Receive RA + prefix and test DAD	142.875000	142.913000	143.022000	142.941000	142.962000	143.021000	143.038000	142.893000	142.958000	142.907000	142.953000
DAD Success	142.878000	142.915000	143.025000	142.944000	142.965000	143.024000	143.040000	142.895000	142.961000	142.910000	142.955700
Send BU	142.878000	142.915000	143.025000	142.944000	142.965000	143.024000	143.040000	142.895000	142.961000	142.910000	142.955700
Receive BU Ack	142.939000	142.977000	143.087000	143.006000	143.027000	143.085000	143.082000	142.937000	143.002000	142.952000	143.009400
DAD Delay	0.003000	0.002000	0.003000	0.003000	0.003000	0.003000	0.002000	0.002000	0.003000	0.003000	0.002700
Handover Layer 3 Delay	0.064000	0.064000	0.065000	0.065000	0.065000	0.064000	0.044000	0.044000	0.044000	0.045000	0.056400
Return to Home Network											
Signal up	239.703000	239.813000	239.720000	239.698000	239.715000	239.719000	239.702000	239.772000	239.863000	239.706000	239.741100
Receive RA + prefix and send BU	239.724000	239.835000	239.741000	239.719000	239.735000	239.740000	239.723000	239.793000	239.884000	239.727000	239.762100
Receive BU Ack	239.775171	239.886163	239.793124	239.770940	239.787310	239.789531	239.772939	239.845019	239.935894	239.778894	239.813499
Handover Layer 3 Delay	0.072171	0.073163	0.073124	0.072940	0.072310	0.070531	0.070939	0.073019	0.072894	0.072894	0.072399

A-DAD Missing NCoA case

	1	2	3	4	5	6	7	8	9	10	Average
Home Network											
Signal up	0.918562	0.243069	0.898985	0.290124	1.012220	0.262763	1.025320	0.671268	0.420621	0.295653	0.603859
Receive Link Local Address (FE80::)	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
Receive RA + prefix and test DAD	1.320550	1.320930	1.320750	1.320890	1.320550	1.320910	1.320590	1.303580	1.320910	1.320510	1.319017
DAD Success	2.620550	2.620930	2.620750	2.620890	2.620550	2.620910	2.620590	2.603580	2.620910	2.620510	2.619017
DAD Delay	1.300000	1.300000	1.300000	1.300000	1.300000	1.300000	1.300000	1.300000	1.300000	1.300000	1.300000
Foreign network 1											
Signal up	46.672200	46.651300	46.573300	46.507400	46.542600	46.590600	46.553200	46.613600	46.627800	46.525600	46.585760
Receive RA + prefix and test DAD	46.693200	46.672600	46.594300	46.528600	46.563500	46.611700	46.574300	46.634400	46.632000	46.546900	46.605150
DAD Success	47.993200	47.972600	47.894300	47.828600	47.863500	47.911700	47.874300	47.934400	47.932000	47.846900	47.905150
Send BU	47.993200	47.972600	47.894300	47.828600	47.863500	47.911700	47.874300	47.934400	47.932000	47.846900	47.905150
Receive BU Ack	49.595200	49.574500	49.496700	49.431300	49.465400	49.513800	49.476100	49.537000	49.534600	49.449200	49.507380
DAD Delay	1.300000	1.300000	1.300000	1.300000	1.300000	1.300000	1.300000	1.300000	1.300000	1.300000	1.300000
Handover Layer 3 Delay	2.923000	2.923200	2.923400	2.923900	2.922800	2.923200	2.922900	2.923400	2.906800	2.923600	2.921620
Foreign network 2											
Signal up	142.900000	142.870000	142.881000	142.996000	142.966000	142.971000	142.907000	142.900000	142.870000	143.007000	142.926800
Receive RA + prefix and test DAD	142.922000	142.891000	142.903000	143.017000	142.987000	142.992000	142.929000	142.921000	142.891000	143.027000	142.948000
DAD Success	144.222000	144.191000	144.203000	144.317000	144.287000	144.292000	144.229000	144.221000	144.191000	144.327000	144.248000
Send BU	144.222000	144.191000	144.203000	144.317000	144.287000	144.292000	144.229000	144.221000	144.191000	144.327000	144.248000
Receive BU Ack	144.264000	144.233000	144.244000	144.358000	144.329000	144.334000	144.270000	144.263000	144.232000	144.369000	144.289600
DAD Delay	1.300000	1.300000	1.300000	1.300000	1.300000	1.300000	1.300000	1.300000	1.300000	1.300000	1.300000
Handover Layer 3 Delay	1.364000	1.363000	1.363000	1.362000	1.363000	1.363000	1.363000	1.363000	1.362000	1.362000	1.362800
Return to Home Network											
Signal up	239.704000	239.748000	239.766000	239.778000	239.771000	239.731000	239.768000	239.737000	239.857000	239.775000	239.763500
Receive RA + prefix and send BU	239.707000	239.769000	239.787000	239.799000	239.792000	239.752000	239.789000	239.758000	239.878000	239.796000	239.782700
Receive BU Ack	239.757813	239.819949	239.837291	239.849617	239.844211	239.803007	239.839935	239.809120	239.926783	239.848966	239.833669
Handover Layer 3 Delay	0.053813	0.071949	0.071291	0.071617	0.073211	0.072007	0.071935	0.072120	0.069783	0.073966	0.070169

FR-DAD Ideal case

	1	2	3	4	5	6	7	8	9	10	Average
Home Network											
Signal up	0.912936	0.241709	0.902845	0.281844	1.012220	0.257663	1.031740	0.670074	0.428501	0.289084	0.602862
Receive Link Local Address (FE80::)	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
Receive RA + prefix and test DAD	1.010870	1.010970	1.011350	1.010550	1.034130	1.010730	1.053810	1.010810	1.010790	1.011230	1.017524
DAD Success	1.012770	1.012810	1.012870	1.011910	1.036510	1.012610	1.056470	1.013130	1.012330	1.012730	1.019414
DAD Delay	0.001900	0.001840	0.001520	0.001360	0.002380	0.001880	0.002660	0.002320	0.001540	0.001500	0.001890
Foreign network 1											
Signal up	46.581300	46.656100	46.680700	46.570200	46.581100	46.656500	46.553500	46.635600	46.679400	46.711600	46.630600
Receive RA + prefix and test DAD	46.602500	46.676700	46.701400	46.591400	46.601700	46.677700	46.574600	46.656800	46.700400	46.732600	46.651580
DAD Success	46.605200	46.679400	46.704600	46.593800	46.604500	46.679800	46.577600	46.660000	46.703800	46.734600	46.654330
Send BU	46.605200	46.679400	46.704600	46.593800	46.604500	46.679800	46.577600	46.660000	46.703800	46.734600	46.654330
Receive BU Ack	48.207800	48.281400	48.306300	48.195800	48.206100	48.282000	48.179700	48.261500	48.305800	48.336400	48.256280
DAD Delay	0.002700	0.002700	0.003200	0.002400	0.002800	0.002100	0.003000	0.003200	0.003400	0.002000	0.002750
Handover Layer 3 Delay	1.626500	1.625300	1.625600	1.625600	1.625000	1.625500	1.626200	1.625900	1.626400	1.624800	1.625680
Foreign network 2											
Signal up	143.015000	142.901000	142.917000	142.877000	142.914000	142.910000	142.926000	142.962000	143.015000	143.002000	142.943900
Receive RA + prefix and test DAD	143.036000	142.922000	142.938000	142.898000	142.935000	142.932000	142.946000	142.983000	143.036000	143.023000	142.964900
DAD Success	143.039000	142.925000	142.942000	142.901000	142.938000	142.934000	142.950000	142.986000	143.039000	143.026000	142.968000
Send BU	143.039000	142.925000	142.942000	142.901000	142.938000	142.934000	142.950000	142.986000	143.039000	143.026000	142.968000
Receive BU Ack	143.100000	142.967000	143.004000	142.963000	143.000000	142.996000	142.992000	143.048000	143.081000	143.068000	143.021900
DAD Delay	0.003000	0.003000	0.004000	0.003000	0.003000	0.002000	0.004000	0.003000	0.003000	0.003000	0.003100
Handover Layer 3 Delay	0.085000	0.066000	0.087000	0.086000	0.086000	0.086000	0.066000	0.086000	0.066000	0.066000	0.078000
Return to Home Network											
Signal up	239.809000	239.774000	239.788000	239.697000	239.841000	239.760000	239.699000	239.713000	239.778000	239.811000	239.767000
Receive RA + prefix and send BU	239.829000	239.795000	239.809000	239.718000	239.862000	239.782000	239.719000	239.734000	239.799000	239.832000	239.787900
Receive BU Ack	239.878834	239.846585	239.858527	239.768519	239.911132	239.834162	239.770169	239.785707	239.849517	239.883003	239.838616
Handover Layer 3 Delay	0.069834	0.072585	0.070527	0.071519	0.070132	0.074162	0.071169	0.072707	0.071517	0.072003	0.071616

A-DAD Ideal case

	1	2	3	4	5	6	7	8	9	10	Average
Home Network											
Signal up	0.912936	0.241709	0.922845	0.281844	1.012220	0.257663	1.031740	0.670074	0.428501	0.289084	0.604862
Receive Link Local Address (FE80::)	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
Receive RA + prefix and test DAD	1.321170	1.320710	1.321130	1.320890	1.320550	1.321410	1.320490	1.320990	1.320910	1.319070	1.320732
DAD Success	1.323090	1.322970	1.322610	1.322930	1.322650	1.323630	1.322910	1.322990	1.322970	1.321530	1.322828
DAD Delay	0.001920	0.002260	0.001480	0.002040	0.002100	0.002220	0.002420	0.002000	0.002060	0.002460	0.002096
Foreign network 1											
Signal up	46.532300	46.587200	46.633500	46.588700	46.590300	46.567800	46.513400	46.711000	46.606300	46.561500	46.589200
Receive RA + prefix and test DAD	46.553200	46.608100	46.654600	46.610000	46.611500	46.587700	46.534100	46.691800	46.608500	46.582400	46.604190
DAD Success	46.556300	46.611800	46.657300	46.612900	46.613800	46.590500	46.536200	46.694100	46.611500	46.585300	46.606970
Send BU	46.556300	46.611800	46.657300	46.612900	46.613800	46.590500	46.536200	46.694100	46.611500	46.585300	46.606970
Receive BU Ack	48.158500	48.213700	48.259300	48.215000	48.215700	48.192800	48.137500	48.296300	48.213500	48.186500	48.208880
DAD Delay	0.003100	0.003700	0.002700	0.002900	0.002300	0.002800	0.002100	0.002300	0.003000	0.002900	0.002780
Handover Layer 3 Delay	1.605300	1.605600	1.604700	1.605000	1.604200	1.605100	1.603400	1.604500	1.605000	1.604100	1.604690
Foreign network 2											
Signal up	142.854000	142.891000	143.001000	142.919000	142.941000	143.000000	143.017000	142.872000	142.937000	142.886000	142.931800
Receive RA + prefix and test DAD	142.875000	142.913000	143.022000	142.941000	142.962000	143.021000	143.038000	142.893000	142.958000	142.907000	142.953000
DAD Success	142.878000	142.915000	143.025000	142.944000	142.965000	143.024000	143.040000	142.895000	142.961000	142.910000	142.955700
Send BU	142.878000	142.915000	143.025000	142.944000	142.965000	143.024000	143.040000	142.895000	142.961000	142.910000	142.955700
Receive BU Ack	142.939000	142.977000	143.087000	143.006000	143.027000	143.085000	143.082000	142.937000	143.002000	142.952000	143.009400
DAD Delay	0.003000	0.002000	0.003000	0.003000	0.003000	0.003000	0.002000	0.002000	0.003000	0.003000	0.002700
Handover Layer 3 Delay	0.064000	0.064000	0.065000	0.065000	0.065000	0.064000	0.044000	0.044000	0.044000	0.045000	0.056400
Return to Home Network											
Signal up	239.703000	239.813000	239.720000	239.698000	239.715000	239.719000	239.702000	239.772000	239.863000	239.706000	239.741100
Receive RA + prefix and send BU	239.724000	239.835000	239.741000	239.719000	239.735000	239.740000	239.723000	239.793000	239.884000	239.727000	239.762100
Receive BU Ack	239.775171	239.886163	239.793124	239.770940	239.787310	239.789531	239.772939	239.845019	239.935894	239.778894	239.813499
Handover Layer 3 Delay	0.072171	0.073163	0.073124	0.072940	0.072310	0.070531	0.070939	0.073019	0.072894	0.072894	0.072399

ประวัติการศึกษา และการทำงาน

ชื่อ –นามสกุล	นายพหล โสตถิวิรัช
วัน เดือน ปี ที่เกิด	วันที่ 10 สิงหาคม 2524
สถานที่เกิด	กรุงเทพมหานคร
ประวัติการศึกษา	ระดับปริญญาตรี คณะวิศวกรรมศาสตร์ สาขาไฟฟ้า-สื่อสาร มหาวิทยาลัยเกษตรศาสตร์ วิทยาเขตบางเขน
ตำแหน่งหน้าที่การงานปัจจุบัน	วิศวกรระบบ
สถานที่ทำงานปัจจุบัน	บริษัท วิทยุการบินแห่งประเทศไทย จำกัด
ผลงานดีเด่นและรางวัลทางวิชาการ	
ทุนการศึกษาที่ได้รับ	