

ค่อนໂທຣລເລອ່ວ

ค่อนໂທຣລເລອ່ວປະກອບດ້ວຍແອປພລິເຄັນເພື່ອກາຈັດການ ແລະ ກຳກັບການທຳງານຂອງໂຟລ່ວ ຄືອເປັນຫັວໃຈສຳຄັນຂອງ SDN ທາກເທີບກັບຮະບບຄອມພົວເຕົວເບີຢີບໄດ້ກັບສ່ວນຂອງຊີ່ພືໍ່ (CPU) ຂອງຮະບບເພື່ອຄວບຄຸມອຸປະກອນຕ່າງໆ ທີ່ອຸ່ງກາຍໃນເນື້ອເວີຣີກ ຮວມถึงດາຕ້າເພັນຂອງອຸປະກອນ SDN

ໃນບັນຫຼຸງນີ້ຈະໄດ້ ອົບຍື່ງສຳຄັນການຈັດການຂອງຄອນໂທຣລເລອ່ວ ແລະ ຄອນໂທຣລເລອ່ວທີ່ຖູກພັດນາຂຶ້ນໂດຍຜູ້ຜົລິຕຸປະກອນຕ່າງໆ ຮວມถึงຄອນໂທຣລເລອ່ວແບບໂອເປັນຊອ່ວົງ ໂດຍໃນທີ່ນີ້ ຈະເນັ້ນໂອເປັນເດිලິ່ທ (OpenDaylight) ຄອນໂທຣລເລອ່ວ ປ້ຈຸບັນຄືອັນໝື່ງຂອງໂອເປັນຊອ່ວົງທີ່ໄດ້ຮັບຄວາມນິຍມຍ່າງນາກ

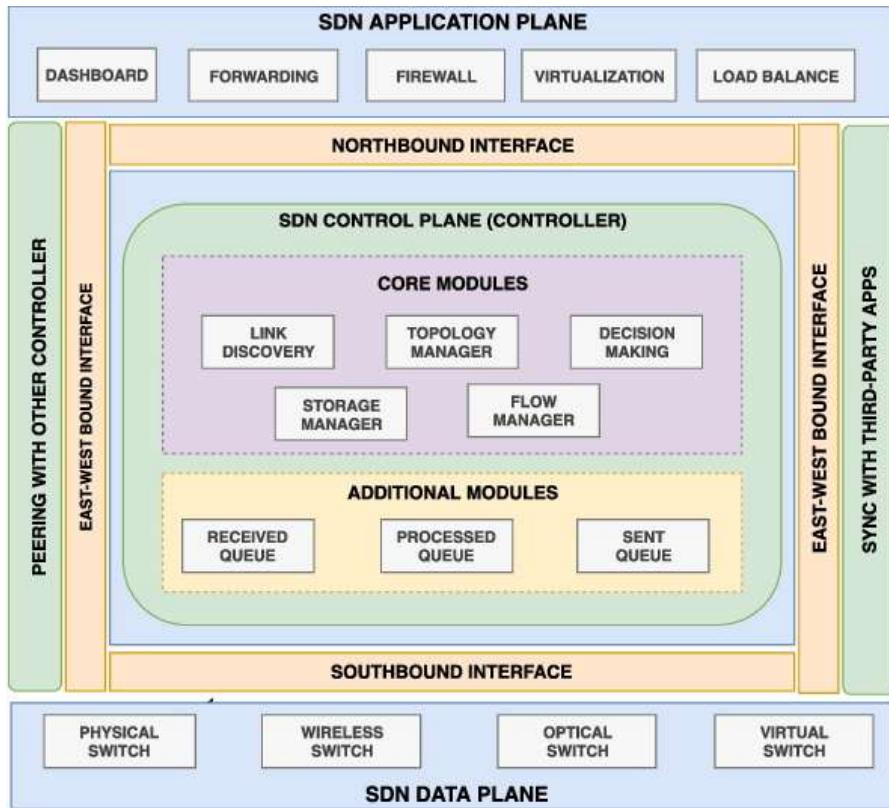
12.1 ສຕາປໍຕຍກຣມຂອງຄອນໂທຣລເລອ່ວ

ຄອນໂທຣລເລອ່ວ ບໍ່ອບາຍຄັ້ງຈາກອ້າງຄື່ງໃນຮູບຂອງ Network Operating System (NOS) ຄືອເປັນສ່ວນສຳຄັນ ໃນການຈັດການກັບຮະບບເນື້ອເວີຣີກປົງແບບ SDN ແນວ່າແຕ່ລະຄອນໂທຣລເລອ່ວຈະມີພິັງກໍ່ຂັ້ນແລະຄວາມສາມາດຖື່ທີ່ແຕກຕ່າງກັນ ພບວ່າວ່ອງຄຳປະກອບຂອງຄອນໂທຣລເລອ່ວສາມາດແບ່ງອອກເປັນ 2 ສ່ວນໜັກ ໄດ້ແກ່ ສ່ວນຂອງແກນໜັກຂອງ ຄອນໂທຣລເລອ່ວ (controller core) ແລະ ສ່ວນຂອງອິນເທຼອຣີ-ຟີ່ (interface) ຕ່າງໆ [173] ດັ່ງແສດງໃນຮູບທີ່ 12.1

- ສ່ວນຂອງແກນໜັກຂອງຄອນໂທຣລເລອ່ວ (controller core) ແກນໜັກຂອງຄອນໂທຣລເລອ່ວທຳມະນຸ້ມາທີ່ສຳຄັນ ໃນການຈັດການໂໂທໂລຢີແລະໂຟລ່ວຂອງທຽບພິກ ຜົ່ງປະກອບດ້ວຍໂມດຸລສຳຄັນຕ່າງໆ ໄດ້ແກ່
 - ການຄັ້ນຫາກາເຊື່ອມຕ່ອ (Link Discovery) ສັງເນັມສເສຈ PACKET_OUT ເປັນໜ່ວງ ແລ້ວ ເພື່ອຕຽບສອບພວົມຕະຫຼາດ
 - ການຈັດກາໂໂທໂລຢີ (Topology Manager) ສ້າງເນື້ອເວີຣີໂໂທໂລຢີຈາກການທີ່ໄດ້ຮັບເນັມສເສຈ PACKET_IN ແລະ ຄອຍຕຽບສອບໂໂທໂລຢີທີ່ສ້າງຂຶ້ນ
 - ການຕັດສິນໃຈ (Discision Making) ທຳມະນຸ້ມາກັ້ນຫາເສັ້ນທາງທີ່ດີທີ່ສຸດຮະຫວ່າງໂທນດໃນເນື້ອເວີຣີ ຈາກອຸ່ງບຸນພື້ນຮູນຂອງຄຸນກາພາກການໃໝ່ບໍລິການ (QoS) ບໍ່ຮັບການປົກກັນ (Security)
 - ການຈັດກາສ່ວນພື້ນທີ່ຈັດເກີບ (Storage Manager) ການຈັດກາເກີນການຈັດເກີບ
 - ການຈັດກາໂຟລ່ວ (Flow Manager) ທຳມະນຸ້ມາທີ່ສຳຄັນຈັດກາກັບໂຟລ່ວເວັ້ນທີ່ແລະໂຟລ່ວທີ່ເປີດຂອງດາຕ້າເພັນ

ນອກຈາກສ່ວນປະກອບທີ່ກຳລ່ວງໄປແລ້ວ ຄອນໂທຣລເລອ່ວຍັງຈາກປະກອບໄປດ້ວຍກາເກີບຂໍ້ມູນທາງສົດຖານີ ຮວມໄປຄື່ງສ່ວນຂອງຄົວແນນເນຈອ່ວ (queue manager) ເພື່ອເກີບຂໍ້ມູນທີ່ເກີນເຫັນກັບປະສິທິພາບ ແລະ ຈັດກາສ່ວນຂອງຄົວໜ້າ (incoming) ແລະ ຂາອອກ (outgoing)

- ອິນເທຼອຣີ-ຟີ່ (Interface) ເຫັນທີ່ມາດີວ່າ ຂາທີ່ມາດີວ່າ ອິນເທຼອຣີ-ຟີ່ແລະ ນອർທີ່ມາດີວ່າ ອິນເທຼອຣີ-ຟີ່ຄືອເປັນສອງອິນເທຼອຣີ-ຟີ່ທີ່ລັກຂອງຄອນໂທຣລເລອ່ວ ໂດຍໂອເປັນໂຟລ່ວ ຄືອເປັນເຫັນທີ່ມາດີວ່າ ອິນເທຼອຣີ-ຟີ່ທີ່ລັກ ໃນຂະນະທີ່ສ່ວນຂອງນອർທີ່ມາດີວ່າ ອິນເທຼອຣີ-ຟີ່ຂອງຍຸ່ນພື້ນຮູນຂອງ REST API ເປັນໜັກ



รูปที่ 12.1: สถาปัตยกรรมทั่วไปของคอนโทรลเลอร์ [173]

นอกจากนี้ ในเน็ตเวิร์กที่มีมากกว่า 1 คอนโทรลเลอร์ การติดต่อระหว่างคอนโทรลเลอร์จะอาศัยการสื่อสารผ่านเวสท์บันด์(Westbound) อินเทอร์เฟซซึ่งไม่มีมาตรฐานหลัก แต่ละคอนโทรลเลอร์ใช้การสื่อสารที่แตกต่างกัน ทำให้คอนโทรลเลอร์ที่ต่างกันไม่สามารถทำงานร่วมกันได้ อินเทอร์เฟซสุดท้ายได้แก่ อีสท์บันด์(Eastbound) อินเทอร์เฟซทำหน้าที่ในการติดต่อกับเราเตอร์ที่ถูกใช้งานเดิม (legacy router) โดยผ่าน BGP เป็นโพรโทคอลหลักที่ใช้

ปัจจุบันถือว่ามีความหลากหลายของคอนโทรลเลอร์เป็นอย่างมาก โดยที่แต่ละคอนโทรลเลอร์มีคุณสมบัติที่แตกต่างกัน ตั้งแต่ประสิทธิภาพ (high-performance) การรองรับการขยายตัว (scalable) โปรแกรมที่ใช้ในการพัฒนา และผู้ผลิต (vendor) ดังนั้น การเลือกคอนโทรลเลอร์จะต้องคำนึงถึงความสามารถและความเหมาะสมในด้านต่าง ๆ ทั้งนี้ จากการศึกษาของ Ashton, Metzler & Associates พบว่าในการเลือกใช้คอนโทรลเลอร์จะต้องพิจารณาใน 10 สิ่งต่อไปนี้ [172]

1. การรองรับการทำงานของโอเพนโฟล์ว (OpenFlow Support): โอเพนโฟล์วถือเป็นส่วนสำคัญที่ทำให้เกิดรูปแบบการสื่อสารของ SDN เรียกว่าเป็นเข้าท์บานด์อินเทอร์เฟซแรกก็ได้ที่ทำให้สามารถควบคุมการสื่อสารของด้าต้าเพลนของสวิตช์ ปัจจุบันจากที่ได้กล่าวมาแล้วโอเพนโฟล์วแต่ละเวอร์ชันมีความสามารถแมตซ์ยอดเดือร์ของแพ็กเก็ตที่แตกต่างกัน เช่น ในช่วงเวอร์ชัน 1.0 ไม่สามารถที่จะรองรับ IPv6 ได้ การรองรับ QoS ด้วยมิเตอร์เบิลต้องเริ่มที่เวอร์ชัน 1.3 ทำให้การพิจารณาคอนโทรลเลอร์ที่ต้องการ ต้องคำนึงในส่วนของฟังก์ชันที่ต้องการ และเข้าใจถึงทิศทางการพัฒนาที่เกิดขึ้น (roadmap) ของคอนโทรลเลอร์ที่จะนำมาใช้
2. เน็ตเวิร์กเวร์ชวลไลเซชัน (Network Virtualization) การทำงานในรูปแบบเวร์ชวลไลเซชันถือว่าเป็น

ส่วนสำคัญของเน็ตเวิร์กปัจจุบัน ตัวอย่างที่รู้จักอย่างแพร่หลายได้แก่ Virtual LAN (VLAN) ซึ่ง VLAN ทำให้สามารถที่จะแบ่งอีเทอร์เน็ตเวิร์กออกเป็น 4,094 บรรดาศักดิ์โดยมีการกำหนดแยกทรัพย์สินของแต่ละกลุ่มผู้ใช้ออกจากกันโดยสิ้นเชิง แม้ว่าจะทำงานอยู่บนสวิตช์เดียวกัน อีกหนึ่งตัวอย่างได้แก่ Virtual Routing and Forwarding (VRF) เพื่อทำเน็ตเวิร์กเวอร์ชวลໄລ່ເຊັ່ນ ให้ร้าເຕົວສາມາດสร้างເວຼິວໜ້າເຕົວອືນສແພນສິ້ນນາ ຈຶ່ງແຕ່ລະອືນສແພນສີ້ນທີ່ກຳນົດກັບເຮົາຕິ່ງໂປຣໂຕຄອລຂອງພ້ອມກັບຈັດການກັບພົກລົງວິເວີຣິດິງເທິບີລຂອງທຸນເອງ

นอกเหนือ VLAN และ VRF แล้ว ส่วนหนึ่งที่สำคัญคือ ความสามารถเพื่อรับการสร้างເວຼິວໜ້າເນື້ນ-ເວິຣິກຈາກເນື້ນ-ເວິຣິກທີ່ມີຢູ່ຕື່ອເປັນສ່ວນສຳຄັນມາກ ຈຶ່ງການທຳມາດໃນຮູບແບບດັ່ງກ່າວ ທຳໄຫ້ເກີດໂຄຮສ້າງເນື້ນ-ເວິຣິກທີ່ຫາກຫາຍ ເພື່ອຮອງຜູ້ໃຊ້ (tenant) ຈຳນວນນາກ ໂດຍແຕ່ລະຜູ້ໃຊ້ເສີມອນມີເນື້ນ-ເວິຣິກຂອງທຸນເອງ ການທຳມາດໃນຮູບແບບນີ້ ເປັນການແກ້ໄຂໂຄຮສ້າງຂອງເວຼິວໜ້າເນື້ນ-ເວິຣິກຈາກໂຄຮສ້າງເນື້ນ-ເວິຣິກທີ່ໃຊ້ ທຳໄຫ້ສະດວກຕ່ອກການປັບປຸງເປົ້າຍືນຫຼືວິເກີໄຂເນື້ນ-ເວິຣິກ ສະດວກແກ້ການຂໍາຍາໂຄຮສ້າງເນື້ນ-ເວິຣິກໃນອານັດ ໂດຍໄໝກະທບກັບໂຟລົງທີ່ມີຢູ່ແລ້ວ

3. ຮອງຮັບຝັກໜັນຕ່າງ ໆ ຂອງເນື້ນ-ເວິຣິກ (Network Fuctionality) ຄອນໂທຣເລອ່ຽທີ່ຕ້ອງສາມາດຮອງຮັບການສ້າງເວຼິວໜ້າເນື້ນ-ເວິຣິກຈຶ່ງໃຫ້ແຕ່ລະເນື້ນ-ເວິຣິກທຳມາດອີສະຕ່ຕ່ອກກັນໂດຍສິ້ນເຊີງ ແລະທີ່ສຳຄັນຄອນໂທຣເລອ່ຽທີ່ຕ້ອງສາມາດຮັບການທີ່ຈະສ້າງໂຟລົງຕາມຄວາມຕ້ອງການຂອງຄຸນພາບ (QoS) ການໃຫ້ບັນດາທີ່ຕ້ອງການ ທັນນີ້ຄອນໂທຣເລອ່ຽຄວ່າທີ່ຈະກຳນົດເສັ້ນທາງການສ້ອສານໄດ້ມາກກ່າວໜຶ່ງເສັ້ນທາງ ແລະສາມາດຮັບການທີ່ຈະແບ່ງທຽບພິກຈາກໂຟລົງໃນຫຍາເສັ້ນທາງ ຄວາມສາມາດຮັບການນີ້ຈະໃຫ້ສາມາດເພີ່ມປະສິທິພາບ ແລະລດຄວາມຈຳເປັນໃນການໃໝ່ໂປຣໂຕຄອລໃໝ່ ເຊັ່ນ Transparent Interconnection of Lots of Links (TRILL) ຢີ້ວິ Shortest Path Bridging (SPB)

4. ການຂໍາຍາຕ້ວຂອງຮະບບ (Scalability) ການທຳມາດຂອງ SDN ທຳໄຫ້ຜູ້ອຸດ ສາມາດເພີ່ມເນື້ນ-ເວິຣິກຝັກໜັນຕ່າງໆ ໄດ້ຕາມຕ້ອງການ ເພີ່ມອຸນນີ້ເພື່ອເນື້ນ-ເວິຣິກເດືອຍ ປັບປຸງສຳຄັນຂອງການຂໍາຍາຕ້ວຂອງ SDN ອີ່ ຄວາມສາມາດຮັບການທີ່ຈະຮອງຮັບຈຳນວນສົວົງທີ່ເພີ່ມຂຶ້ນໃນຮະບບໄດ້ ການທຳມາດແບບຮັບແກ້ໄຂ (reactive) ເມື່ອພົບວ່າໄມ່ສາມາດຕັດສິນໄຈໄດ້ ສິ່ງທີ່ຕ້ອງທຳອົາ ການຮັບແກ້ໄຂຂອງພັກເກີດຕັດກ່າວ່າໄປຍັງຄອນໂທຣເລອ່ຽຈຶ່ງນີ້ເມື່ອມີຈຳນວນນັກຂຶ້ນ ພາກໃຫ້ໃຫ້ເກີດຄອງຫຸດ (bottleneck) ຂຶ້ນທີ່ຄອນໂທຣເລອ່ຽໄດ້ ອີ່ປັບປຸງທີ່ອຈາກເກີດຂຶ້ນ ອີ່ ການຮັບແກ້ໄຂຂອງໂຟລົງເທິບີເລື່ອນທີ່ ເນື່ອຈາກການທຳມາດແບບຮັບແກ້ໄຂໃໝ່ ໃນເນື້ນ-ເວິຣິກທີ່ມີຂັາດໃຫຍ່ ຄອນໂທຣເລອ່ຽຈະຕ້ອງສາມາດປະມາລພັດຈຳນວນໂຟລົງໄດ້ເປັນຈຳນວນຫຼັກລ້ານໂຟລົງຕ່ອງວິນາທີ

5. ປະສິທິພາບ (Performance) ການກຳນົດໂຟລົງຂອງຄອນໂທຣເລອ່ຽອີ່ເປັນປັຈຢັງສຳຄັນຂອງປະສິທິພາບ ປະກອບດ້ວຍ 2 ສ່ວນ ໄດ້ໄກ້ ເວລາທີ່ໃໝ່ເພື່ອຕິດຕັ້ງໂຟລົງ (flow set up time) ແລະຄວາມເງື່ອງການກຳນົດຕັ້ງ ຕັ້ງຈຳນວນໂຟລົງຕ່ອງວິນາທີ ໂດຍທີ່ຄ່າທັງ 2 ມີຜລຍ່າງມາກຕ່ອງການເພີ່ມຈຳນວນຄອນໂທຣເລອ່ຽທາງຈຳເປັນ ເຊັ່ນກຣັນທີ່ການໃຫ້ຕິດຕັ້ງໂຟລົງຈຳນວນນັກໃຫ້ກົດສົວົງເປົ້າຍືນໄວ້ຈະຮອງຮັບໄດ້ ທຳໄໝມີຄວາມຈຳເປັນຕ້ອງເພີ່ມຈຳນວນຄອນໂທຣເລອ່ຽ

ນອກເහັນຈາກນີ້ ຕ້ອງຄຳນີ້ຄົງຮູບແບບຂອງການຕິດຕັ້ງໂຟລົງເປັນແບບໂປຣ໌ (proactive) ຢີ້ວິຮັ່ງ໌ (reactive) ການຕິດຕັ້ງແບບໂປຣ໌ ຈະໃຫ້ການຕິດຕັ້ງໂຟລົງເກີນຂຶ້ນກ່ອນທີ່ສົວົງຈະໄດ້ຮັບພັກເກີດ ທຳໄໝສົວົງສາມາດຮັບກັບພັກເກີດໄດ້ທັນທີ່ ຕຽບກັນຂ້າມທາກເປັນແບບຮັ່ງ໌ ສົວົງທີ່ຕ້ອງສັງພັກເກີດໄປຍັງຄອນໂທຣເລອ່ຽເພື່ອຕຽບສັງພັກເກີດທີ່ເກີດຂຶ້ນອຸນນີ້ຈາກເວລາຫົວ່ວ່າທີ່ເກີດຂຶ້ນ ຍັງສັງພັກເກີດຕ່ອງການທຳມາດຂອງສົວົງແລະຄວາມສາມາດຮັບການສ້ອສານແລະການປະມາລຂອງຄອນໂທຣເລອ່ຽ ເຊັ່ນ ຄອນໂທຣເລອ່ຽທີ່ເສີ່ນດ້ວຍພາກພາກ C ຈະມາຄວາມເງື່ອງວ່າຄອນໂທຣເລອ່ຽທີ່ເສີ່ນດ້ວຍພາກ Java ອ່າງມາກ ທັນນີ້ຄາດຫວັງວ່າຄອນໂທຣເລອ່ຽຈະໄໝເກີດເປັນຈຸດຄອງຫຸດຂອງການສ້າງແລະລັບໂຟລົງເລື່ອນທີ່

6. ความสามารถในการโปรแกรมของเน็ตเวิร์ก (Network Programmability) โดยทั่วไป การโปรแกรมอุปกรณ์เน็ตเวิร์กจะทำในรูปแบบตัวต่อตัว ซึ่งการโปรแกรมในรูปแบบดังกล่าวค่อนข้างจะใช้เวลา มีโอกาสเกิดความผิดพลาดที่สูง และอาจทำให้ระบบโดยรวมขาดเสียหาย หากจากนี้การโปรแกรมในรูปแบบดังกล่าว ยังไม่ปรับเปลี่ยนตามสถานะของระบบที่เปลี่ยนแปลง

การทำงานของ SDN การโปรแกรมจะถูกกำหนดโดยคอนโทรลเลอร์ ซึ่งทำให้สามารถต่อสนองต่อสถานะการณ์ที่เกิดขึ้นอย่างรวดเร็ว เนื่องจากในปัจจุบันมีการเชื่อมต่อของอุปกรณ์เป็นจำนวนมาก และมีการเกิดขึ้นของเซอร์วิสใหม่ๆ การทำงานของ SDN คอนโทรลเลอร์ยังสามารถรองรับการโปรแกรมในรูปแบบ scrip ปท (script) ของ CLI ซึ่งต่างจากการโปรแกรมทั่วไป ทำให้การใช้งานโดยคอนโทรลเลอร์มีความยืดหยุ่นมากกว่า

นอกจากนี้ การใช้งานคอนโทรลเลอร์ยังทำให้สามารถโปรแกรมผ่านอินเทอร์เฟซ API ทำให้สามารถบริหารจัดการการทำงานของเน็ตเวิร์กได้ละเอียดขึ้น เช่น การจัดการ QoS ตามความต้องการของแอปพลิเคชัน รวมไปถึงการให้บริการไฟร์วอลล์และการทำโหลดбалานซ์ เป็นต้น

7. ความน่าเชื่อถือ (Reliability): การใช้งานคอนโทรลเลอร์ก่อให้เกิดปัญหาสำคัญ คือ จุดเดียวของความผิดพลาด (Single point of Failure) หากเกิดความผิดพลาดเกิดขึ้นจะทำให้ระบบไม่สามารถดำเนินการได้อย่างต่อเนื่อง นอกจากนี้การทำงานของคอนโทรลเลอร์ต้องสามารถตอบสนองต่อปัญหาการเชื่อมระหว่างการสื่อสารได้อย่างรวดเร็ว เช่น หากพบว่ามีเส้นทางเสียหาย (failure) คอนโทรลเลอร์ต้องสามารถที่จะค้นหาเส้นทางใหม่ (reroute) ส่งต่อทรัพฟิกไปยังเส้นทางอื่น การทำเช่นนี้ส่งผลให้คอนโทรลเลอร์ต้องค่อยตรวจสอบโถโพโลยีของเน็ตเวิร์กอย่างต่อเนื่อง รวมถึงสามารถรองรับการทำงานการทำงานของไฟร์วอลล์ เพื่อความน่าเชื่อถือของเน็ตเวิร์กเช่น Vrrp Redundancy Protocol (VRRP) และ Multi-Chassis Link Aggregation Group (MC-LAG) เป็นต้น

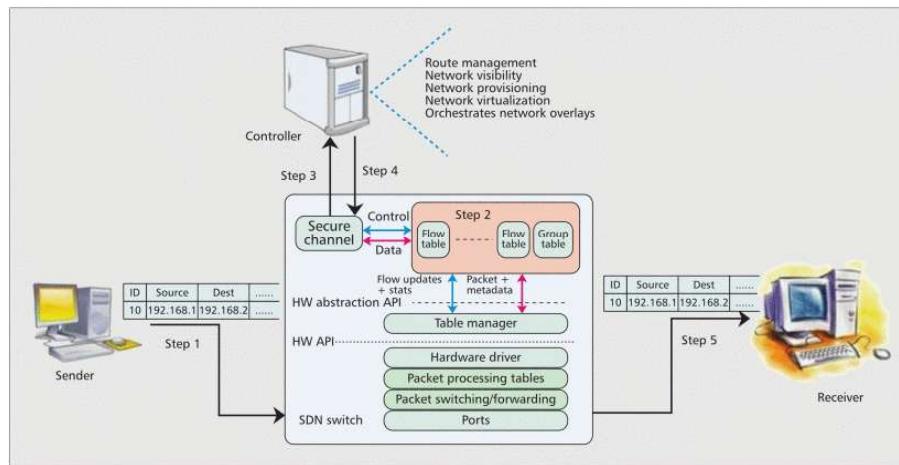
เพื่อเพิ่มความน่าเชื่อถือ การใช้งานคอนโทรลเลอร์ควรรองรับการทำงานในรูปแบบคลัสเตอร์ เช่น มี 2 คอนโทรลเลอร์หรือมากกว่าทำงานร่วมกันภายใต้คลัสเตอร์ คอนโทรลเลอร์ที่ไม่ได้ใช้งานสามารถอยู่ในโหมดสแตนด์บาย (stand by) โดยทั้งนี้แต่ละคอนโทรลเลอร์จะต้องประสานข้อมูลกัน (synchronization) ตลอดเวลา ถือเป็นอีกส่วนสำคัญของการทำงานในรูปแบบคลัสเตอร์

8. ความปลอดภัยของเน็ตเวิร์ก (Security of the Network) ความปลอดภัยถือเป็นส่วนสำคัญของคอนโทรลเลอร์ ดังนั้นคอนโทรลเลอร์ต้องมีระบบการพิสูจน์ทราบตัวตน (authentication) และ การอนุญาตใช้งาน (authorization) ในระดับสูง รวมไปถึงความสามารถกำหนด ACLs ที่ดี นอกจากนี้ คอนโทรลเลอร์ต้องสามารถแยกผู้ใช้ (tenant) แต่ละคนออกจากกันอย่างสิ้นเชิง แม้ว่าจะทำงานอยู่บนโครงสร้างพื้นฐานเดียวกัน ที่สำคัญ คอนโทรลเลอร์ต้องแจ้งเตือนผู้ดูแลได้อย่างทันท่วงที

9. การดูแล ตรวจสอบ และแสดงผลส่วนกลาง (Centralized Monitoring and Visualization) ความสามารถของการตรวจสอบการทำงานของแต่ละไฟล์ตั้งแต่ต้นทางไปยังปลายทาง (end-to-end) เป็นอีกปัจจัยสำคัญของการดูแลระบบเน็ตเวิร์ก การทำงานของเน็ตเวิร์กในรูปแบบเดิมโดยปราศจาก SDN ทำให้สามารถตรวจสอบการทำงานดังกล่าวเป็นไปได้ ทั้งนี้การทำงานของ SDN อาศัยการบันทึกเควนเชอร์ของไฟล์ ทำให้คอนโทรลเลอร์สามารถศักย์ข้อมูลจากไฟล์เพื่อตรวจสอบปัญหาที่เกิดขึ้น และแก้ไขปัญหาของไฟล์นั้น ๆ ได้

การทำงานเน็ตเวิร์กปัจจุบันประกอบด้วยการสื่อสารบนพิสิคอลลิงก์และแบบเวอร์ชวลเน็ตเวิร์ก ทำให้คอนโทรลเลอร์ต้องสามารถแสดงผลการทำงานของทั้งสองตัว และควรรองรับการทำงานกับไฟร์วอลล์ SNMP เพื่อสามารถตรวจสอบสภาพการทำงานของอุปกรณ์ต่าง ๆ ได้

10. ผู้ผลิตค่อนโตรลเลอร์ (The SDN Controller Vendor) เนื่องจากปัจจุบันได้เกิดผู้เล่นหลายรายที่เข้าสู่การพัฒนา SDN ค่อนโตรลเลอร์ ซึ่งการเลือกค่อนโตรลเลอร์ต้องคำนึงความต่อเนื่องของการใช้งาน และการพัฒนาฟังก์ชันใหม่ การพัฒนาอย่างต่อเนื่องถือเป็นส่วนสำคัญของการเลือก SDN ค่อนโตรลเลอร์ บริษัท CISCO, Linux Foundation, Intel, IBM, Juniper และ อื่นๆ เป็นตัวอย่างของบริษัทที่ร่วมหนึ่งของผู้เล่นที่สำคัญในตลาด SDN



รูปที่ 12.2: The operation of SDN[?]

12.2 โอเพนซอร์สค่อนโตรลเลอร์

จากการความต้องการของ SDN ทำให้มีการพัฒนาจำนวนมาก ในที่นี้จะนำเสนอส่วนของค่อนโตรลเลอร์ที่เกิดขึ้นเพียงบางส่วน ที่เป็นส่วนหลักด้านค่อนโตรลเลอร์ในปัจจุบัน และค่อนโตรลเลอร์ที่ยังมีการใช้งานอยู่

- **น็อก/พ้อก (NOX/POX)** POX เป็น SND ค่อนโตรลเลอร์ที่พัฒนาขึ้นจาก NOX[98] ในภาษา C++ เป็นภาษาไฟรอน ซึ่ง NOX ถือเป็น SDN แรกๆ ที่ถูกพัฒนาขึ้น โดย Nicira และมีสถาบันวิจัย (research community) ทำให้มีการพัฒนาค่อนโตรลเลอร์ต่อยอดจาก NOX เช่น NOX-MT[99] เพื่อรองรับการทำงานแบบมัลติ-thread และ QNOX[100] เพื่อรองรับคุณภาพการให้บริการ (QoS) จนสุดท้ายถูกพัฒนาเป็น พ้อก (POX)[101] บนไฟรอน และปรับปรุงให้มีประสิทธิภาพที่ดีขึ้น มีการใช้งานส่วนใหญ่อยู่บนกลุ่มนักวิจัย
- **บีคอน (Beacon)** ถูกพัฒนาขึ้นที่มหาวิทยาลัยสแตนฟอร์ด และบริษัท Big Switch Network ในปี ค.ศ. 2010 ทำงานแบบมัลติเดดด้วยภาษา Java ทำให้สามารถทำงานในรูปแบบหลายแพลตฟอร์ม บีคอน ถือว่าได้รับความนิยมอย่างมากในกลุ่มนักวิจัย โดยมีจุดประสงค์หลักของการออกแบบคือ ให้นักพัฒนาสามารถสร้างผลงานที่ดีด้วยการสนับสนุนไลบรารีต่างๆ เพื่อการพัฒนา มีสิทธิภาพที่สูง และความสามารถที่สามารถเริ่มหรือหยุดแอปพลิเคชันระหว่างที่ทำงานอยู่ได้
- **ฟลัดไลน์ (Floodlight)** เป็นอีกหนึ่งค่อนโตรลเลอร์ที่ถูกพัฒนาบน Java ต่อยอดมาจากบีคอน ได้รับการสนับสนุนจากบริษัท Big Switch Networks สถาบัตยกรรมของฟลัดไลน์ อยู่บนพื้นฐานของ Big Network Controller (BNC)

- **เมสโตร (Maestro)** ອຸກພັນນາເປັນພາຫາຈາວແບບ multi-thread ໂດຍມາວິທາລີໄຣສ ເພື່ອໃໝ່ປະສິທ-
ຈິກາພທີ່ສູງດ້ວຍການທຳມານແບບຂານ (parallelism) ຈາກເຕັໂນໂລຢີຂອງຊື່ພິໍແບບ multi-core ນອກຈາກ
ນີ້ Meastro ຍັງຮອບການໂຟລ່ວທີ່ມາພ້ອມຈັກນ
- **ຮີຢູ (RYU)** ເປັນການອົກແບບຄອນໂທຣລເລອ່ວແບບແກກສ່ວນ (component-based) ການອົກແບບໃໝ່ API
ທີ່ຈ່າຍສໍາຫັ້ນຜູ້ພັນນາ ເພື່ອສ້າງແອປລີເຄີ້ນໃນການຄວບຄຸມແລະຈັດການເນື້ອເວົ້າໄດ້ຈ່າຍ ສາມາດຮອງຈັບເຫຼົ່າ
ທົບການດີໂພໂທໂຄລ່ອ ແຫ່ນ ໂອເປັນໂຟລ່ວ ແລະ NETCONF ເປັນຕົ້ນ ສາມາດຮອງຈັບໂອເປັນໂຟລ່ວຕັ້ງແຕ່ເວົ້ວ
1.0 ຄຶ້ງເວົ້ວຊັ້ນ 1.5 ເປັນໂອເປັນຂອ້ງຮອຍໆກາຍໄດ້ລື້ສີທີ່ Apache 2.0
- **ໂອເປັນເດີໄລ່ (OpenDaylight)** ເປັນ SDN ດອນໂທຣລເລອ່ວແບບໂອເປັນຂອ້ງທີ່ມີລັກຂະນະເປັນແພລຕິໂຮມ
ແບບໂມໂດລ ເພື່ອໃຫ້ສາມາດປັບແຕ່ເພື່ອຮອງຈັບເນື້ອເວົ້າທຸກໆຂາດ ໂອເປັນເດີໄລ່ຕ່າງຈາກ SDN ອື່ນທີ່ມີອູ່
ໂດຍມີການເນັ້ນສ່ວນການທຳມານຂອງຄວາມສາມາດຂອງການໂປຣແກຣມເນື້ອເວົ້າ ອຸກອົກແບບຕັ້ງແຕ່ເຮີມຕົ້ນເພື່ອ
ຮອງຈັບການໃຊ້ເຈິ້ງພານີຍກັບເນື້ອເວົ້າທີ່ມີອູ່
- **ໂອນອສ (ONOS)** ຄື່ອເປັນໂອເປັນຂອ້ງແຮກ ທີ່ອົກແບບເພື່ອຮອງຈັບ SDN ສໍາຫັກລຸ່ມຜູ້ໃຫ້ບໍລິການອິນເຕອຣ-
ເນື້ອເວົ້າ ໂດຍມີຈຸດປະສົງທີ່ຈະສັບສົນການທຳມານເນື້ອເວົ້າແບບຕ່ອນເນື່ອງ (high availability, HA) ກາຮຍາຍ
ຕ້ວ່າ ປະສິທິພາຫາພທີ່ສູງ ໂອນອສພັນນາສ່ວນຂອງນອർທົບການດີແອັບສແຕຣກຂັ້ນແລະ API ເພື່ອໃຫ້ຜູ້ພັນນາສາ-
ມາມາດທຳໄດ້ຈ່າຍ ແລະເຫົ່າທົບການດີແອັບສແຕຣກຂັ້ນແລະການເຂື່ອມຕ່ອງເພື່ອຮອງຈັບອຸປະກອນເນື້ອເວົ້າທີ່ພ້ອມ
ສໍາຫັກໂອເປັນໂຟລ່ວແລະອຸປະກອນເນື້ອເວົ້າເດີມ ໂດຍທີ່ ໂອນອສໄດ້ຮັບການພັນນາຮ່ວມກັບຜູ້ໃຫ້ບໍລິການເຄືອຂ່າຍ
ເຂົ່າ AT&T ແລະ NTT Communications ຜູ້ຜົດອຸປະກອນຕ່າງໆ ເຂົ່າ Ciena, Ericsson Huawei ແລະ
Intel ຮ່ວມຄົງການທຳມານກັບ ONF

ຊື່	ສາຕັປຍກຣມ	ພາຫາ	ໂຟລ່ວ/ວິນາທີ	ນອർທົບການດີ API	ໂນໂດລ	ຜູ້ສັບສົນ	ລື້ສີທີ່
Beacon	Conctrailized, Multithread	C++	1.8M	Ad-hoc API	ຕໍ່າ	Nicira	GPLv3
Beacon	Conctrailized, Multithread	Java	12.8M	Ad-hoc API	ປານກລາງ	Stanford University	GPLv2
ONOS	Distributed	Java	1M	RESTful API	ປານກລາງ Ericsson, Hauwei, Intel	AT&T, Ciena, Cisco	-
Beacon	Conctrailized, Multithread	Java	4.8M	Ad-hoc API	ປານກລາງ	RICE, NSF	LGPLv2.1
OpenDaylight	Conctrailized	Java	106K	REST RESTCONF	ປານກລາງ ມາກ	Linux ແລະບ.oື່ນ ຈົ່ງ Cisco, IBM, NEC	EPLv1.0

ຕາரາງທີ 12.1: ຕ້າວອຍ່າງເປົ້າໃຫ້ການໂອເປັນໂຟລ່ວດ້ວຍດອນໂທຣລເລອ່ວ[39]

12.3 ການທົດສອບການທຳມານໂອເປັນໂຟລ່ວດ້ວຍດອນໂທຣລເລອ່ວ POX

ເພື່ອຫາຜູ້ອ່ານເຂົ້າໃຈຄວາມສັນພັນການທຳມານຂອງດອນໂທຣລເລອ່ວ ແລະການທຳມານຂອງໂອເປັນໂຟລ່ວ ທັງນີ້ຈະໃໝ່
ດອນໂທຣລເລອ່ວ POX ເພື່ອທົດສອບການທຳມານອ່າງຍໍາຍ ດ້ວຍການທົດສອບການທຳມານໄຟໄຟໃນເຄື່ອງຕົນເອງ (IP=127.0.0.1)
ເນື່ອງຈາກດອນໂທຣລເລອ່ວ POX ທຳມານນີ້ໄຟໂຮມໂວຣ່ຂັ້ນ 2.7 ຜູ້ສາມາດຮັດຕິຕັ້ງເພີ່ມເຕີມໄດ້ຕັ້ງນີ້

```
sudo apt update
sudo apt install python2-minimal
```

1. ເຮັດວຽກທຳມານດອນໂທຣລເລອ່ວ ໃນທີ່ນີ້ຈະໃໝ່ໂມໂດລ ເພື່ອເຮັດວຽກຕົ້ນ ຈະປາກກູ້ຂອງຄວາມທັງນີ້ ຈາກຂ້ອຄວາມທີ່ປາກ
ກູ້ຈະພບວ່າເປັນການໃຊ້ໂອເປັນໂຟລ່ວເວົ້ວຊັ້ນ 1.0 ແລະໃຊ້ດີໂພລ່ວທີ່ 6633 ຜູ້ພັນນາໄດ້ເຫັນທາກຕ້ອງການ

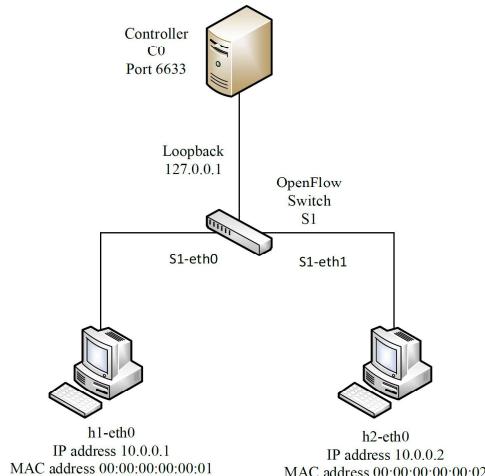
เปลี่ยนแปลงหมายเลขพอร์ต หมายเลข IP address หรือการเข้ารหัส โดยการใส่ --port=หมายเลขที่ต้องการ หรือการใช้ SSL ในการสื่อสาร --private-key=คีย์ที่ต้องการ (ปกติไม่มี) เป็นต้น

```
root@vagrant:/home/ck/pox# ./pox.py --verbose forwarding
  12_learning
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
DEBUG:core:POX 0.5.0 (eel) going up...
DEBUG:core:Running on CPython (2.7.18/Jul 1 2022 10:30:50)
DEBUG:core:Platform is Linux-5.15.0-56-generic-x86_64-with-Ubuntu
-22.04-jammy
INFO:core:POX 0.5.0 (eel) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
```

2. ให้เปิดโปรแกรม Wireshark เพื่อบันทึกการสื่อสารระหว่างคอนโทรลเลอร์และสวิตช์ หากผู้อ่านไม่แน่ใจ ให้ตรวจสอบเทอร์เฟซที่จะตรวจจับก่อน โดยใช้คำสั่ง ifconfig หรือ ip address list

3. ทดสอบการสร้างเน็ตเวิร์กในมินิเน็ตอย่างง่าย พร้อมกำหนด IP address ให้คอนโทรลเลอร์ และหมายเลขพอร์ต POX กำหนดหมายเลขพอร์ตที่ 6633 จะได้เน็ตเวิร์กดังแสดงในรูปที่ 12.3

```
ck@vagrant:~$ sudo mn --mac --switch=ovsk --controller=remote,ip
=127.0.0.1,port=6633
```



รูปที่ 12.3: รูปแบบเน็ตเวิร์กของมินิเน็ตที่ใช้ทดสอบ

พร้อมกันนี้ ให้สังเกตการทำงานของคอนโทรลเลอร์จะปรากฏข้อความแสดงถึงการเชื่อมต่อที่เกิดขึ้น

```
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
DEBUG:forwarding.12_learning:Connection [00-00-00-00-00-01 2]
DEBUG:openflow.of_01: connection aborted
```

4. ตรวจสอบโฟล์วภายในสวิตช์และเทเบิล arp จะเห็นว่าไม่ปรากฏข้อมูลใด ๆ

```
mininet> dpctl dump-flows
*** s1
```

```
mininet> h1 arp
```

5. ทดสอบการ ping เพื่อไม่ให้สับสนกับข้อมูลที่ได้ กำหนดให้ ping เพียง 1 ครั้งเท่านั้น (-c1) mininet>

```
h1 ping -c1 h2
```

```
mininet> h1 ping -c2 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=26.7 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.840 ms

--- 10.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 0.840/13.772/26.705/12.932 ms
```

6. ตรวจสอบโฟล์กายในสวิตช์และเทเบิล arp จะปรากฏข้อมูล ดังนี้

```
mininet> dpctl dump-flows
*** s1
-----
cookie=0x0, duration=2.812s, table=0, n_packets=2, n_bytes=196,
    idle_timeout=10, hard_timeout=30, priority=65535,icmp,in_port
    ="s1-eth1",vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst
    =00:00:00:00:00:02,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,
    icmp_type=8,icmp_code=0 actions=output:"s1-eth2"
cookie=0x0, duration=2.810s, table=0, n_packets=2, n_bytes=196,
    idle_timeout=10, hard_timeout=30, priority=65535,icmp,in_port
    ="s1-eth2",vlan_tci=0x0000,dl_src=00:00:00:00:00:02,dl_dst
    =00:00:00:00:00:01,nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,
    icmp_type=0,icmp_code=0 actions=output:"s1-eth1"
mininet> h1 arp
Address      HWtype  HWaddress      Flags Mask   Iface
10.0.0.2      ether   00:00:00:00:00:02 C          h1-eth0
```

7. ตรวจสอบคอนโทรลเลอร์จะปรากฏข้อมูลที่เกิดจากการติดต่อของโหนดในมินิเน็ต ดังนี้

```
DEBUG:forwarding.12_learning:installing flow for
00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
DEBUG:forwarding.12_learning:installing flow for
00:00:00:00:00:01.1 -> 00:00:00:00:00:02.2
DEBUG:forwarding.12_learning:installing flow for
00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
DEBUG:forwarding.12_learning:installing flow for
00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
DEBUG:forwarding.12_learning:installing flow for
00:00:00:00:00:01.1 -> 00:00:00:00:00:02.2
```

12.3.1 รายละเอียดข้อมูลภายในสวิตช์

โปรแกรม dpctl (administer OpenFlow datapaths) เป็นโปรแกรมสำหรับตรวจสอบและการจัดการ (administering) กับโอเพนโฟล์ด้าพาร์ค[63] สามารถใช้เพื่อแสดงสถานะของดาต้าพาร์คุณสมบัติที่เกี่ยวข้อง คอนฟิกุเลชัน และเทเบิลเอ็นทรี เมื่อใช้เครื่องเหลื่องโอเพนโฟล์ โปรแกรม dpctl สามารถเพิ่ม ลบ แก้ไข และตรวจสอบ ดาต้าพาร์คได้ จากผลที่ได้ตัวแปรที่สำคัญมีดังนี้

- in_port หมายเลขพอร์ตขาเข้า
- dl_src หมายเลข MAC address ของต้นทาง

- dl_dst=mac หมายเลข MAC address ของปลายทาง
- dl_type ประเภทของໂປຣໂຕຄອລ ເຊັ່ນ ethertype หมายเลข ອືເກວ່ຽນເນີຕໂປຣໂຕຄອລ
- nw_src หมายเลข IP address ຕິ່ນທາງ ເປັນ IPv4
- nw_proto ໂປຣໂຕຄອລ
- tp_src หมายเลขພອຣີຕິ່ນທາງ
- tp_dst หมายเลขພອຣີປ່າຍທາງທາງ
- icmp_type ປະເກທ (TYPE) ຂອງເມສເສຈ ICMP
- icmp_code ໂດັບ (CODE) ຂອງເມສເສຈ ICMP
- table หมายเลขຂອງເທັບເລີດທີ່ແສດງຄໍາ

ນອກຈາກນີ້ ຈະມີການໃໝ່ໃນຮູບແບບຍ່ອ ດັ່ງນີ້

- ip ຈະມີຄໍາເທົ່າກັບ dl_type=0x0800.
- icmp ຈະມີຄໍາເທົ່າກັບ dl_type=0x0800, nw_proto=1.
- tcp ຈະມີຄໍາເທົ່າກັບ dl_type=0x0800, nw_proto=6.
- udp ຈະມີຄໍາເທົ່າກັບ dl_type=0x0800, nw_proto=17.
- arp ຈະມີຄໍາເທົ່າກັບ dl_type=0x0806.

ຈາກຂໍ້ມູນຂ້າງຕັນ ທຳໃຫ້ທ່ານວ່າ ຈາກຄໍາສ້າງ dpctl dump-flows ປະກອບດ້ວຍ 3 ເມສເສຈຂ້າງຕັນ ປະກອບ ດ້ວຍ

- ເມສເສຈ ທີ່ 1 ເກີດຈາກການສ່າງທ່ານຂອງໂປຣໂຕຄອລ ARP
- ເມສເສຈ ທີ່ 2 ເກີດຈາກການຄໍາສ້າງ Echo Request (ping) icmp_type=8, icmp_code=0
- ເມສເສຈ ທີ່ 2 ເກີດຈາກການຄໍາສ້າງ Echo Reply (ping) Ping icmp_type=0, icmp_code=0

ນອກການໃໝ່ຄໍາສ້າງ dpctl ຜູ້ອ້ານສາມາດໃໝ່ຄໍາສ້າງ ovs-ofctl ໄດ້ ເພື່ອດູ້ຂໍ້ມູນໃນສວິຕ່ຈະໃຫ້ຜລເໜືອນກັບການໃໝ່ ຄໍາສ້າງ dpctl ທີ່ແສດງໄປແລ້ວ

```
mininet> sh ovs-ofctl dump-flows s1
```

12.3.2 ຂໍ້ມູນຈາກໂປຣແກຣມ Wireshark

- ການສື່ອສາරະໜ່າງສວິຕ່ ແລະ ຄອນໂທຣເລອ່ວ ເມື່ອເຮີ່ມຕັນທ່ານ ການທ່ານພິລເຕອຣີໂປຣໂຕຄອລ ໂອເພນໂຟລ່ວ ເນື່ອຈາກໂລເພນວິສວິຕ່ ອົງຮັບເວຼ່ອຮັບສູງສຸດທີ່ 1.4 (openflow_v5) ໃນຂະໜາດທີ່ ຄອນໂທຣເລອ່ວ ອອງຮັບເວຼ່ອຮັບສູງສຸດທີ່ 1.0 (openflow_v1) ໄກ້ກໍາທັນພິລເຕອຣີທັງສອງເວຼ່ອຮັບເວຼ່ອໃຫ້ຄໍາທີ່ ຄົວວັນ
- ການສື່ອສາຮີເກີດຂຶ້ນຈາກການ ping ສັງເກດວ່າຈະເປັນການໃໝ່ ໂອເພນໂຟລ່ວເວຼ່ອຮັບ 1.0 ເນື່ອຈາກຄອນໂທຣເລອ່ວ ສາມາດຮັບເທົ່າສູງສຸດທີ່ເວຼ່ອຮັບນີ້ ທຳມະເກີດການສື່ອສາຮີທັງຈາກເມສເສຈ HELLO ເປັນເວຼ່ອຮັບ 1.0 ທັງໝົດ

No.	Time	Source	Destination	Protocol	Length	Info
15	2.519327438	192.168.198.246	192.168.198.230	OpenFlow	74	Type: OFPT_HELLO
17	2.545394867	192.168.198.230	192.168.198.246	OpenFlow	74	Type: OFPT_HELLO
19	2.546293361	192.168.198.230	192.168.198.246	OpenFlow	86	Type: OFPT_STATS_REQUEST
21	2.546630568	192.168.198.246	192.168.198.230	OpenFlow	242	Type: OFPT_FEATURES_REPLY
22	2.546682825	192.168.198.246	192.168.198.230	OpenFlow	11...	Type: OFPT_STATS_REPLY
24	2.548651019	192.168.198.230	192.168.198.246	OpenFlow	78	Type: OFPT_SET_CONFIG
26	2.549170263	192.168.198.230	192.168.198.246	OpenFlow	138	Type: OFPT_FLOW_MOD
28	2.549629769	192.168.198.230	192.168.198.246	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
30	2.549785961	192.168.198.246	192.168.198.230	OpenFlow	74	Type: OFPT_BARRIER_REPLY

< >

- > Internet Protocol Version 4, Src: 192.168.198.246, Dst: 192.168.198.230
- > Transmission Control Protocol, Src Port: 33866, Dst Port: 6633, Seq: 1, Ack: 1, Len: 8
- ✓ OpenFlow 1.4
 - Version: 1.4 (0x05)
 - Type: OFPT_HELLO (0)
 - Length: 8
 - Transaction ID: 125

รูปที่ 12.4: การสื่อสารระหว่างพอร์ตโคลอปโนเเพนโน่ล์กับสวิตช์

No.	Time	Source	Destination	Protocol	Length	Info
126	16.933636813	10.0.0.1	10.0.0.2	OpenFlow	182	Type: OFPT_PACKET_IN
→ 128	16.936866431	10.0.0.1	10.0.0.2	OpenFlow	276	Type: OFPT_PACKET_OUT
130	16.937072793	192.168.198.246	192.168.198.230	OpenFlow	74	Type: OFPT_BARRIER_REPLY
← 131	16.937561159	10.0.0.2	10.0.0.1	OpenFlow	182	Type: OFPT_PACKET_IN
133	16.940253162	10.0.0.2	10.0.0.1	OpenFlow	276	Type: OFPT_PACKET_OUT
135	16.940472983	192.168.198.246	192.168.198.230	OpenFlow	74	Type: OFPT_BARRIER_REPLY

< >

- > Internet Protocol Version 4, Src: 192.168.198.230, Dst: 192.168.198.246
- > Transmission Control Protocol, Src Port: 6633, Dst Port: 33866, Seq: 1857, Ack: 3065, Len: 210
- ✓ OpenFlow 1.0
- ✓ OpenFlow 1.0
- ✓ OpenFlow 1.0
 - .000 0001 = Version: 1.0 (0x01)
 - Type: OFPT_PACKET_OUT (13)
 - Length: 122
 - Transaction ID: 28
 - Buffer Id: 0xffffffff
 - In port: 1
 - Actions length: 8
 - Actions type: Output to switch port (0)
 - Action length: 8
 - Output port: 65529
 - Max length: 0
- > Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:00:02 (00:00:00:00:00:02)
- > Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.2
- > Internet Control Message Protocol

รูปที่ 12.5: การสื่อสารระหว่างพอร์ตโคลอปโนเเพนโน่ล์กับสวิตช์ ระหว่างการ ping

12.4 Datapath OVS

โมดูลเครื่องเนลของ Open vSwitch ช่วยให้สามารถควบคุมการประมวลผลแพ็กเก็ตในระดับไฟล์จากผู้ใช้ (userspace) ได้อย่างมีศักยภาพ บนอุปกรณ์เครือข่ายที่เลือกใช้งาน โมดูลนี้สามารถนำไปใช้เพื่อสร้างสวิตซ์อิเล็กทรอนิกส์แบบพื้นฐาน การทำ bonding ของอุปกรณ์เครือข่าย การประมวลผล VLAN การควบคุมการเข้าถึงเครือข่าย (network access control) การควบคุมเครือข่ายแบบอิงไฟล์ (flow-based network control) และการใช้งานอื่น ๆ อีกมากมาย

โมดูลเครื่องเนลนี้รองรับการทำงานของ “datapath” หลายตัว (ซึ่งเปรียบได้กับ bridge) โดยแต่ละ datapath สามารถมี “vport” ได้หลายตัว (เปรียบเสมือนพอร์ตภายใน bridge) นอกจากนี้ แต่ละ datapath ยังมี “ตารางไฟล์” (flow table) ที่ผ่าน userspace จะเป็นผู้ติดตามข้อมูลด้วย “ไฟล์” ซึ่งทำหน้าที่แยกจัดคีย์ที่ได้จากส่วนหัวของแพ็กเก็ตและเมทาดาทา ไปยังชุดของแอ็อกชันต่าง ๆ แอ็อกชันที่พบบ่อยที่สุดคือการส่งต่อแพ็กเก็ตไปยัง vport อื่น แต่ก็ยังมีแอ็อกชันประเภทอื่น ๆ ที่รองรับเช่นกัน

เมื่อแพ็กเก็ตเข้ามาทาง vport โมดูลเครื่องเนลจะประมวลผลโดยการดึงคีย์ของไฟล์ (flow key) ออกมานะ และค้นหาในตารางไฟล์ หากพบไฟล์ที่ตรงกัน ก็จะดำเนินการตามแอ็อกชันที่ผูกไว้กับไฟล์นั้น หากไม่พบไฟล์ที่ตรงกัน แพ็กเก็ตจะถูกส่งต่อไปยัง userspace เพื่อให้ทำการประมวลผลต่อไป (ซึ่งในระหว่างการประมวลผลนั้น userspace มักจะตั้งค่าไฟล์ใหม่ เพื่อให้แพ็กเก็ตประเภทเดียวกันในอนาคตสามารถถูกจัดการได้ทั้งหมด ภายใต้เครื่องเนล)

เพื่อรองรับความเข้ากันได้ทั้งแบบย้อนกลับและไปข้างหน้า (backward และ forward compatibility) ทุกครั้งที่โมดูลเครื่องเนลส่งแพ็กเก็ตไปยัง userspace จะมีการส่ง flow key ที่เครื่องเนลแยกออกจากแพ็กเก็ตไปด้วย จากนั้น userspace จะสกัด flow key ตามมุ่งมองของตนเองจากแพ็กเก็ต และนำมาเปรียบเทียบกับเวอร์ชันที่เครื่องเนลส่งมา ดังนี้

- หาก flow key ตามมุ่งมองของ userspace ตรงกับ flow key ของเครื่องเนล ก็ไม่จำเป็นต้องดำเนินการพิเศษใด ๆ
- หาก flow key ของเครื่องเนลมีฟิลด์มากกว่า flow key ของ userspace ตัวอย่างเช่น เครื่องเนลสามารถถอดรหัสส่วนหัว IPv6 ได้ แต่ userspace หยุดอยู่แค่ระดับ Ethernet type (เนื่องจากไม่เข้าใจ IPv6) กรณีนี้ก็ยังไม่จำเป็นต้องทำอะไรเป็นพิเศษ userspace ยังคงสามารถตั้งค่าไฟล์ได้ตามปกติ ตราบใดที่ใช้ flow key ที่เครื่องเนลส่งมาให้
- หาก flow key ของ userspace มีฟิลด์มากกว่า flow key ของเครื่องเนล ตัวอย่างเช่น userspace ถอดรหัสส่วนหัว IPv6 ได้ แต่เครื่องเนลหยุดอยู่ที่ Ethernet type ในกรณีนี้ userspace สามารถส่งต่อแพ็กเก็ตด้วยตนเอง โดยไม่ต้องตั้งค่าไฟล์ในเครื่องเนล วิธีนี้จะส่งผลเสียต่อประสิทธิภาพ เนื่องจากแพ็กเก็ตทุกตัวที่เครื่องเนลมองว่าเป็นส่วนหนึ่งของไฟล์นั้นจะต้องถูกส่งไปยัง userspace แต่พฤษติกรรมการส่งต่อแพ็กเก็ตจะยังคงถูกต้อง (หาก userspace สามารถพิจารณาได้ว่าค่าของฟิลด์เพิ่มเติมเหล่านี้ไม่ส่งผลต่อพฤษติกรรมการส่งต่อ ก็อาจเลือกตั้งค่าไฟล์ในเครื่องเนลได้เช่นกัน)

12.5 พื้นฐานการโปรแกรมคอนโทรลเลอร์ POX

ผู้อ่านสามารถเข้าไปตรวจสอบการทำงานของไฟรอนแอปพลิเคชันได้ที่ /pox/forwarding/l2_learning.py รายละเอียดการใช้งานอย่างละเอียดสามารถศึกษาได้ที่เว็บไซต์ของคอนโทรลเลอร์ POX ที่ตาม URL นี้ <https://openflow.stanford.edu/display/ONL/POX+Wiki.html>

เพื่อศึกษาการทำงานของแอปพลิเคชัน ในที่นี่จะแบ่งโค้ดออกเป็นส่วน ๆ เพื่อแสดงให้เห็นถึงการทำงานของแอปพลิเคชันที่สอดคล้องกับกับผลการทำงานที่เกิดขึ้น ทั้งนี้แอปพลิเคชัน l2_learning.py จะทำให้ออpenFlow สร้างช่องทางแลกเปลี่ยนข้อมูลระหว่างสวิตช์ในレイเยอร์ 2

12.5.1 ลิขสิทธิ์ของโปรแกรม

ส่วนเริ่มต้นของโปรแกรม เป็นส่วนแสดงลิขสิทธิ์ของโปรแกรม ซึ่งการนำไปใช้ผู้ใช้ต้องแนบส่วนนี้ด้วย

```
# Copyright 2011-2012 James McCauley
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at:
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied.
# See the License for the specific language governing permissions and
# limitations under the License.
```

12.5.2 ไลบรารีที่ต้องใช้

```
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpid_to_str, str_to_dpid
from pox.lib.util import str_to_bool
import time
%end{lstlisting}
```

- POX Core ถือเป็นส่วนสำคัญของ POX เพื่อให้ components อื่นๆ ในโปรแกรม register และ component ที่อื่นสามารถเข้ามาตรวจสอบได้
- โอเพนFlowライบรารีเวอร์ชัน 1.0
- ดาต้าพาร์ ID (DPID) แต่ละสวิตช์ในโอเพนFlow จะประกอบด้วยหมายเลขของดาต้าพาร์ขนาด 64 บิต เรียกว่า ดาต้าพาร์ ID หรือ DPID เพื่อให้สวิตช์สื่อสารกับคอนโทรลเลอร์ระหว่างการทำ handshaking ของเมสเซจ ofp_switch_features ดังนั้น POX ใช้ pox.lib.util.dpid_to_str() เพื่อจัดรูปแบบ (format) ของ DPID และเปลี่ยนค่ากลับโดยใช้ str_to_dpid() โดยมีรูปคล้ายค่าของ MAC address แต่ใช้เครื่องหมายขีดกลาง (-) แทน ตามด้วยขีดลง เช่น "00-00-00-00-00-05|123"
- ค่าทางตรรกะ POX กำหนด pox.lib.util's str_to_bool() เพื่อกำหนดให้คำเช่น "on", "true" หรือ "enable" มีค่าเป็นจริงหมวด (True) ส่วนอย่างอื่นให้เป็นเท็จ (False)

12.5.3 รีจิสเตอร์คอมโพเนนต์

```

class l2_learning (object):
    """
    Waits for OpenFlow switches to connect and makes them learning
    switches.
    """
    def __init__ (self, transparent, ignore = None):
        """
        Initialize

        See LearningSwitch for meaning of 'transparent'
        'ignore' is an optional list/set of DPIDs to ignore
        """
        core.openflow.addListener(self)
        self.transparent = transparent
        self.ignore = set(ignore) if ignore else ()

    def _handle_ConnectionUp (self, event):
        if event.dpid in self.ignore:
            log.debug("Ignoring connection %s" % (event.connection,))
            return
        log.debug("Connection %s" % (event.connection,))
        LearningSwitch(event.connection, self.transparent)

    def launch (transparent=False, hold_down=_flood_delay, ignore = None):
        """
        Starts an L2 learning switch.
        """
        try:
            global _flood_delay
            _flood_delay = int(str(hold_down), 10)
            assert _flood_delay >= 0
        except:
            raise RuntimeError("Expected hold-down to be a number")

        if ignore:
            ignore = ignore.replace(',', ' ').split()
            ignore = set(str_to_dpid(dpid) for dpid in ignore)

        core.registerNew(l2_learning, str_to_bool(transparent), ignore)

```

- การรีจิสเตรอร์คอมโพเนนต์สามารถทำโดยพิมพ์ชื่อ registerNew() คอมโพเนนต์ที่ถูกสร้างขึ้นในกรณีนี้ คือ คลาส l2_learning หากใน __init__ เมธอด (method) ของคลาสที่อ้างถึง มีอาร์กูเมนต์ (argument) สามารถส่งผ่านค่าได้เลย สังเกตุการรีจิสเตรอร์ที่ใช้
- หลังจากรีจิสเตรอร์เมื่อมีการเชื่อมต่อเกิดขึ้น พิมพ์ชื่อ _handle_ConnectionUp จะถูกเรียก (invoke) ในที่นี่เรียกว่าปั้ยงคลาส LearningSwitch ต่อไป

12.5.4 การจัดการแพ็คเก็ต

ส่วนของการจัดการแพ็คเก็ตคลาส LearningSwitch

```

class LearningSwitch (object):
    def __init__ (self, connection, transparent):
        ...
    def _handle_PacketIn (self, event):
        ...

```

การทำงานของ LearningSwitch ประกอบด้วย 2 ส่วนสำคัญ คือ

- กำหนดค่าเริ่มต้นต่าง ๆ รวมถึงกำหนด event listen เมื่อได้รับแพ็กเก็ตด้วยคำสั่ง `connection.addListener(self)`
- จัดการกับแพ็กเก็ตที่ได้รับ `_handle_PacketIn` โดยโปรแกรมพังก์ชันที่จำเป็นต่าง ๆ

การกำหนดค่าเริ่มต้น

```
def __init__(self, connection, transparent):
    self.connection = connection
    self.transparent = transparent

    self.macToPort = {}

    connection.addListener(self)
    self.hold_down_expired = _flood_delay == 0
```

การจัดการแพ็กเก็ตที่ได้รับ

```
def _handle_PacketIn(self, event):
    packet = event.parsed
    def flood(message = None):
        msg = of.ofp_packet_out()
        if time.time() - self.connection.connect_time >= _flood_delay:

            if self.hold_down_expired is False:
                self.hold_down_expired = True
                log.info("%s: Flood hold-down expired -- flooding",
                         dpid_to_str(event.dpid))
                msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
            else:
                pass
        msg.data = event.ofp
        msg.in_port = event.port
        self.connection.send(msg)

    def drop(duration = None):
        """
        Drops this packet and optionally installs a flow to continue
        dropping similar ones for a while
        """
        if duration is not None:
            if not isinstance(duration, tuple):
                duration = (duration, duration)
            msg = of.ofp_flow_mod()
            msg.match = of.ofp_match.from_packet(packet)
            msg.idle_timeout = duration[0]
            msg.hard_timeout = duration[1]
            msg.buffer_id = event.ofp.buffer_id
            self.connection.send(msg)
        elif event.ofp.buffer_id is not None:
            msg = of.ofp_packet_out()
            msg.buffer_id = event.ofp.buffer_id
            msg.in_port = event.port
            self.connection.send(msg)
```

```

self.macToPort[packet.src] = event.port # 1

if not self.transparent: # 2
    if packet.type == packet.LLDP_TYPE or packet.dst.isBridgeFiltered():
        drop() # 2a
        return

if packet.dst.is_multicast:
    flood()
else:
    if packet.dst not in self.macToPort:
        flood("Port for %s unknown -- flooding" % (packet.dst,))
    else:
        port = self.macToPort[packet.dst]
        if port == event.port:

            log.warning("Same port for packet from %s -> %s on %s.%s.
                         Drop."
                         % (packet.src, packet.dst, dpid_to_str(event.dpid), port)
                         )
            drop(10)
            return

        log.debug("installing flow for %s.%i -> %s.%i" %
                  (packet.src, event.port, packet.dst, port))
        msg = of.ofp_flow_mod()
        msg.match = of.ofp_match.from_packet(packet, event.port)
        msg.idle_timeout = 10
        msg.hard_timeout = 30
        msg.actions.append(of.ofp_action_output(port = port))
        msg.data = event.ofp # 6a
        self.connection.send(msg)

```

การจัดการกับแพ็คเก็ตประกอบด้วยการใช้งาน 2 เมสเซจที่สำคัญ ได้แก่

- ofp_packet_out เป็นเมสเสจที่ส่งไปยังสวิตช์เพื่อให้ทำงานตามแอ็คชั่นที่กำหนด ในกรณีนี้ให้สวิตช์ส่งแพ็คเก็ตออกไปทุกพอร์ต of.OFPP_FLOOD
 - buffer_id หมายเลขบัฟเฟอร์ที่แพ็คเก็ตจัดเก็บในดาต้าพาร์ ไม่ต้องมีหากเป็นการส่งใหม่
 - in_port หมายเลขพอร์ตของสวิตช์ที่ได้รับแพ็คเก็ต หากมีการส่งแพ็คเก็ตออกไป ต้องมีค่าเป็น OFPP_NONE
 - actions รายการแอ็คชั่น ที่ต้องทำ (oft_action_XXXX) ในตัวอย่าง คือ ofp_action_output
 - data ข้อมูลที่ต้องการให้สวิตช์ส่ง
- ofp_flow_mod แก้ไขโฟล์วเทเบิล ถือเป็นส่วนสำคัญของผลลัพธ์ที่เกิดขึ้นจากคำสั่ง dpctl ที่ใช้ ประกอบด้วย
 - cookie หมายเลขของโฟล์วอินทรีที่ใช้ ไม่จำเป็นต้องมีเป็นออบชัน
 - command รูปแบบการทำงาน ได้แก่
 - * OFPFC_ADD เพิ่มรูปแบบการแมช (match rule) ลงในดาต้าพาร์
 - * OFPFC_MODIFY แก้ไขการแมช (match) เขตเดอร์

- * OFPFC_MODIFY_STRICT แก้ไขการแมช (match) เขตเดอร์เป็นแบบ wildcard เท่านั้น
- * OFPFC_DELETE ลบรูปแบบการแมช
- * OFPFC_DELETE_STRICT - ลบการแมช (match) เขตเดอร์เป็นแบบ wildcard
- idle_timeout - รูปแบบการแมชจะถูกลบทิ้งตามเวลาที่กำหนดนี้หน่วยเป็นวินาที โดยดีฟอลต์จะไม่มีค่า (OFP_FLOW_PERMANENT)
- priority ค่าความสำคัญของการแมชที่จะใช้ ค่าสูงมากยิ่งสำคัญ แต่ไม่ครอบคลุมถึงการแมชที่ตรงกันพอดี (exact match)
- buffer_id บัฟเฟอร์ของดาต้าแพทท์ไฟล์ใหม่จะใช้ ให้มีค่าเป็น None หากไม่ระบุ
- out_port ใช้กรณีรอนเมมีชคำสั่งลบ (DELETE) หากไม่มีการทำหนดอย่างไร ให้ค่าเป็น OFPP_NONE
- flags กำหนดค่า หากมีการใช้งานต่อไปนี้
 - * OFPFF_SEND_FLOW_REM ส่งเมสเจสไปยังคอนโทรลเลอร์แจ้งให้ลับไฟล์ออก หากไฟล์นั้นหมดอายุ (expire)
 - * OFPFF_CHECK_OVERLAP แจ้งว่ามีความซ้ำซ้อนของไฟล์อื่นหรือไม่
 - * OFPFF_EMERG สำหรับไฟล์ใช้กรณีฉุกเฉิน เมื่อมีการเชื่อมต่อระหว่างสวิตช์และคอนโทรลเลอร์เสียหาย
- actions รายการแอคชั่นที่ต้องทำ ตามลำดับที่ถูกเพิ่มเข้ามา
- match (ofp_match) โครงสร้างการแมช (Match Structure) สำหรับรูปแบบการแมช (รายละเอียดด้านไป)

โครงสร้างการแมช (Match Structure)

การแมชเขตเดอร์ของโอเพนไฟล์กำหนดรูปแบบไว้ในคลาสของ ofp_match เพื่อให้สามารถกำหนดเขตเดอร์ที่ต้องการแมชได้ หากไม่มีการทำหนดทุกฟิลด์สามารถเป็นได้ทุกค่า (wildcard, *) ค่าที่กำหนดเพื่อใช้กับโอเพนไฟล์เวอร์ชัน 1.0 ของ POX มีดังนี้

พารามิเตอร์	ความหมาย
dl_dst	Ethernet destination address
dl_src	Ethernet source address
dl_type	Ethertype / length (e.g. 0x0800 = IPv4)
nw_dst	IP destination address
nw_proto	IP protocol (e.g., 6 = TCP) or lower 8 bits of ARP opcode
nw_src	IP source address
nw_tos	IP TOS/DS bits
in_port	Switch port number the packet arrived on
tp_dst	TCP/UDP destination port
tp_src	TCP/UDP source port
dl_vlan	VLAN ID
dl_vlan_pcp	VLAN priority

ตารางที่ 12.2: แมชฟิลด์สำหรับโอเพนไฟล์เวอร์ชัน 1.0

12.6 Open Virtual Network (OVN)

OVN makes use of OVS, a production quality multilayer switch platform that opens the forwarding functions to programmatic extension and control, which provides further integration down to hardware acceleration through its support for multiple data path providers and flow APIs. This includes at the kernel level with SwitchDev / TC, and in the userspace with eBPF XDP and DPDK rte_flow.

12.7 สถาปัตยกรรมและพื้นฐานของ OVN

OVN หรือ Open Virtual Network เป็นระบบที่สนับสนุนการทำ abstraction ของเครือข่ายเชิงตรรกะ (logical network abstraction) ในสภาพแวดล้อมของเครื่องเสมือน (virtual machine) และค่อนเทนเนอร์ OVN ทำงานเสริมความสามารถเดิมของ OVS เพื่อเพิ่มการรองรับแบบเน็ฟสำหรับ abstraction ของเครือข่ายเชิงตรรกะ เช่น logical L2 และ L3 overlays รวมถึง security groups นอกจากนี้ บริการอย่าง DHCP ก็ถือเป็นคุณสมบัติที่พึงประสงค์ เช่นกัน เช่นเดียวกับ OVS เป้าหมายการออกแบบของ OVN คือการมีการติดตั้งใช้งานที่มีคุณภาพระดับ production และสามารถทำงานได้ในสเกลขนาดใหญ่

เครือข่ายกายภาพ (physical network) ประกอบด้วยสายสัญญาณ สวิตช์ และเราเตอร์จริง เครือข่ายเสมือน (virtual network) จะขยายเครือข่ายกายภาพเข้าไปยังแพลตฟอร์มไฮเปอร์ไวเซอร์หรือค่อนเทนเนอร์ โดยเชื่อมต่อ VM หรือค่อนเทนเนอร์เข้ากับเครือข่ายกายภาพ เครือข่ายเชิงตรรกะของ OVN คือเครือข่ายที่ถูกนำໄไปใช้งานด้วยซอฟต์แวร์ และถูกแยกออกจากเครือข่ายกายภาพ (และตั้งนั้นรวมถึงเครือข่ายเสมือน) ด้วยท่อสัญญาณ (tunnels) หรือกลไกการห่อหุ้มข้อมูล (encapsulation) รูปแบบอื่น วิธีนี้ทำให้ address space ของ IP และโปรโตคอลอื่น ๆ ที่ใช้ในเครือข่ายเชิงตรรกะสามารถซ้อนทับกับ address space ที่ใช้ในเครือข่ายกายภาพได้โดยไม่ก่อให้เกิดความขัดแย้ง 拓扑โล耶ชื่อของเครือข่ายเชิงตรรกะสามารถจัดวางได้โดยไม่ต้องคำนึงถึง拓扑โล耶ชื่อของเครือข่ายกายภาพที่รันอยู่ ดังนั้น VM ที่เป็นส่วนหนึ่งของเครือข่ายเชิงตรรกะจะสามารถรับข้อมูลจากเครือข่ายกายภาพหนึ่งไปยังอีกเครื่องหนึ่งได้โดยไม่เกิดการหยุดชะงักของเครือข่าย ดูหัวข้อ Logical Networks ด้านล่างสำหรับข้อมูลเพิ่มเติม

ชั้น encapsulation จะป้องกันไม่ให้ VM และค่อนเทนเนอร์ที่เชื่อมต่อกับเครือข่ายเชิงตรรกะสื่อสารกับโลกภายนอก เนื่องจากเครือข่ายกายภาพได้สำหรับการทำคลัสเตอร์ของ VM และค่อนเทนเนอร์ การจำกัดเช่นนี้อาจยอมรับได้หรือแม้กระทั่งเป็นสิ่งที่ต้องการ แต่ในหลายกรณี VM และค่อนเทนเนอร์จำเป็นต้องมีการเชื่อมต่อกับเครือข่ายกายภาพ OVN ซึ่งมีรูปแบบของเกตเวย์หลายประเภทเพื่อรับรับส่งข้อมูล ดูหัวข้อ Gateways ด้านล่างสำหรับข้อมูลเพิ่มเติม

การติดตั้งใช้งาน OVN หนึ่งชุดประกอบด้วยองค์ประกอบหลักสี่ ได้แก่

- ระบบจัดการคลาวด์ (Cloud Management System: CMS) ซึ่งเป็นโคลอนต์หลักของ OVN (ผู้ดูแลและผู้ดูแลระบบ) การผ่าน OVN เข้ากับ CMS จำเป็นต้องติดตั้งปลั๊กอินเฉพาะของ CMS และซอฟต์แวร์ที่เกี่ยวข้อง (ดูด้านล่าง) ในระยะแรก OVN มุ่งเป้าไปที่ OpenStack ในฐานะ CMS

โดยทั่วไปเรามักกล่าวถึง CMS เพียง "ตัวเดียว" แต่ความสามารถจินตนาการถึงสถานการณ์ที่มี CMS หลายตัวจัดการส่วนต่าง ๆ ของการติดตั้ง OVN เดียวกันได้

- โหนดฐานข้อมูล OVN (OVN Database) ซึ่งอาจเป็นโหนดกายภาพหรือเสมือน (หรือในอนาคตเป็นคลัสเตอร์) ที่ติดตั้งอยู่ในตำแหน่งศูนย์กลาง

- ไฮเปอร์ไวเซอร์หนึ่งตัวหรือมากกว่า (โดยปกติมักมีจำนวนมาก) ไฮเปอร์ไวเซอร์ต้องรัน Open vSwitch และต้องติดตั้งอินเทอร์เฟซตามที่อธิบายไว้ในไฟล์ Documentation/topics/integration.rst ในชอร์สโค้ดของ Open vSwitch แพลตฟอร์มไฮเปอร์ไวเซอร์ได้กิตามที่ Open vSwitch รองรับถือว่ายอมรับได้
- เกตเวย์จำนวนศูนย์ตัวหรือมากกว่า เกตเวย์ทำหน้าที่ขยายเครือข่ายเชิงตรรกะที่อิงกับท่อสัญญาณ (tunnel-based logical network) ไปยังเครือข่ายภายนอก โดยการส่งต่อแพ็กเก็ตแบบสองทิศทางระหว่างท่อสัญญาณและพอร์ต Ethernet ภายนี้ช่วยให้เครื่องที่ไม่ได้ถูกทำให้เป็นสมือนสามารถเข้าร่วมในเครือข่ายเชิงตรรกะได้ เกตเวย์อาจเป็นโญาสต์ภายนอก เครื่องเสมือน หรือสวิตซ์ชาร์ดแวร์แบบ ASIC ที่รองรับสคีมา vtep(5)

ไฮเปอร์ไวเซอร์และเกตเวย์จะถูกเรียกรวมกันว่าโหนดขนส่ง (transport node) หรือ chassis

แผนภาพด้านล่างแสดงให้เห็นถึงการทำงานร่วมกันขององค์ประกอบหลักของ OVN และซอฟต์แวร์ที่เกี่ยวข้อง โดยเริ่มจากด้านบนของแผนภาพ เราจะมี:

- ระบบจัดการคลาวด์ (CMS) ตามที่ได้อธิบายไว้ข้างต้น
- ปลั๊กอิน OVN/CMS ซึ่งเป็นส่วนประกอบของ CMS ที่ทำหน้าที่เชื่อมต่อ กับ OVN ใน OpenStack ปลั๊กอินนี้คือปลั๊กอินของ Neutron วัตถุประสงค์หลักของปลั๊กอินคือการแปลงแนวคิดการกำหนดค่าเครือข่ายเชิงตรรกะของ CMS ซึ่งถูกเก็บไว้ในฐานข้อมูลการกำหนดค่าของ CMS ในรูปแบบเฉพาะของ CMS ให้เป็นตัวแทนกลาง (intermediate representation) ที่ OVN เข้าใจได้

องค์ประกอบนี้จำเป็นต้องเป็นแบบเฉพาะของ CMS ดังนั้นจึงต้องพัฒนาปลั๊กอินใหม่สำหรับ CMS แต่ละตัวที่ต้องการผ่านเข้ากับ OVN ส่วนประกอบทั้งหมดที่อยู่ด้านล่างของปลั๊กอินในแผนภาพจะไม่ขึ้นกับ CMS

- ฐานข้อมูล OVN Northbound ทำหน้าที่รับตัวแทนกลางของการกำหนดค่าเครือข่ายเชิงตรรกะที่ถูกส่งลงมาจากปลั๊กอิน OVN/CMS โครงสร้างสคีมาของฐานข้อมูลถูกออกแบบให้ ``impedance matched'' กับแนวคิดที่ใช้ใน CMS เพื่อรับแนวคิดของ logical switches, routers, ACLs และองค์ประกอบอื่น ๆ โดยตรง รายละเอียดเพิ่มเติมได้ที่ ovn-nb(5)

ฐานข้อมูล OVN Northbound มีโคลอนต์เพียงสองตัว คือปลั๊กอิน OVN/CMS ที่อยู่ด้านบน และ ovn-northd ที่อยู่ด้านล่าง

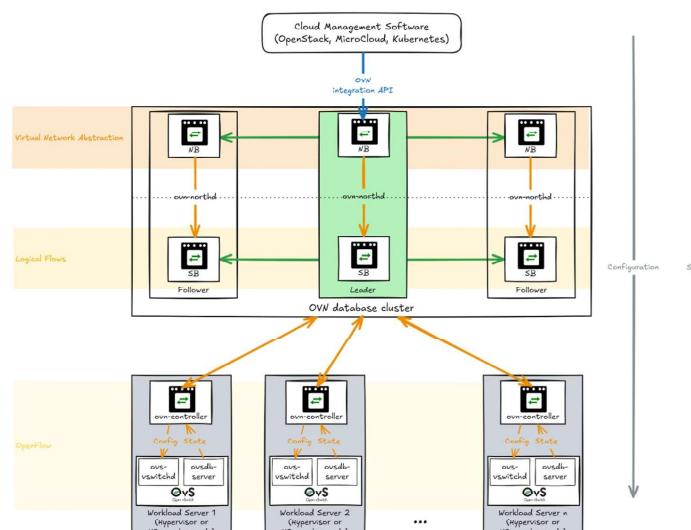
- ovn-northd(8) เชื่อมต่อกับฐานข้อมูล OVN Northbound ที่อยู่ด้านบน และฐานข้อมูล OVN Southbound ที่อยู่ด้านล่าง โดยทำหน้าที่แปลงการกำหนดค่าเครือข่ายเชิงตรรกะในรูปของแนวคิดเครือข่ายแบบตั้งเดิมจากฐานข้อมูล OVN Northbound ให้เป็น logical datapath flows ในฐานข้อมูล OVN Southbound
- ฐานข้อมูล OVN Southbound เป็นศูนย์กลางของระบบ โดยมีโคลอนต์คือ ovn-northd(8) ที่อยู่ด้านบน และ ovn-controller(8) บนโหนดขนส่งทุกโหนดที่อยู่ด้านล่าง

ฐานข้อมูล OVN Southbound ประกอบด้วยข้อมูลสามประเภท ได้แก่ ตาราง Physical Network (PN) ที่ระบุวิธีการเข้าถึงไฮเปอร์ไวเซอร์และโหนดอื่น ๆ ตาราง Logical Network (LN) ที่อธิบายเครือข่ายเชิงตรรกะในรูปของ ``logical datapath flows'' และตาราง Binding ที่เชื่อมโยงตำแหน่งขององค์ประกอบเครือข่ายเชิงตรรกะเข้ากับเครือข่ายภายนอก ไฮเปอร์ไวเซอร์จะเป็นผู้เพิ่มข้อมูลในตาราง PN และ Port_Binding ในขณะที่ ovn-northd(8) จะเป็นผู้เพิ่มข้อมูลในตาราง LN

ประสิทธิภาพของฐานข้อมูล OVN Southbound ต้องสามารถขยายตามจำนวนโหนดชนส่งได้ ซึ่งอาจต้องมีการปรับปรุงใน ovsdb-server(1) เมื่อพบความต้องการที่ต้องการเพิ่มความสามารถในการทำคลัสเตอร์เพื่อความพร้อมใช้งาน

ส่วนประกอบที่เหลือจะถูกทำขึ้นໄอเปอร์ไวเซอร์แต่ละตัว ได้แก่

- ovn-controller(8) ซึ่งเป็นเอเจนต์ของ OVN บนໄอเปอร์ไวเซอร์และซอฟต์แวร์เกตเวย์แต่ละตัว ทางฝั่ง Northbound จะเชื่อมต่อกับฐานข้อมูล OVN Southbound เพื่อเรียนรู้การกำหนดค่าและสถานะของ OVN และเพื่อเตรียมข้อมูลในตาราง PN และคอลัมน์ Chassis ในตาราง Binding ด้วยสถานะของໄอเปอร์ไวเซอร์ ทางฝั่ง Southbound จะเชื่อมต่อกับ ovs-vswitchd(8) ในฐานะ OpenFlow controller เพื่อควบคุมทรัพย์สินเครือข่าย และเชื่อมต่อกับ ovsdb-server(1) ภายในเครื่อง เพื่อผ่านติดตามและควบคุมการกำหนดค่าของ Open vSwitch
- ovs-vswitchd(8) และ ovsdb-server(1) ซึ่งเป็นองค์ประกอบมาตรฐานของ Open vSwitch



รูปที่ 12.6: OVN distributed architecture.

OVN ใช้ ovsdb-server เป็นฐานข้อมูลโดยตรงอย่างสะดวก โดยไม่ต้องเพิ่มการพิ่งพาซอฟต์แวร์ใหม่ใด ๆ แต่ละอินสแตนซ์ของ ovn-controller จะทำงานกับสแกนปัจจุบันของฐานข้อมูลที่มีความสอดคล้องกันอยู่เสมอ โดยจะรักษาการเชื่อมต่อกับฐานข้อมูลในลักษณะอิงตามการอัปเดต (update-driven connection) หากการเชื่อมต่อถูกขัดจังหวะ ovn-controller จะซิงก์ตามให้ทันกับสแกนปัจจุบันสุดที่สอดคล้องกันของเนื้อหาฐานข้อมูลที่เกี่ยวข้อง และประมวลผลข้อมูลเหล่านั้นต่อไป

12.8 การไฟลของข้อมูลการกำหนดค่า (Configuration data flow)

ในการออกแบบของ OVN การกำหนดค่าเครือข่ายจะให้ลงทาง southbound จาก CMS ไปยังโหนดชนส่ง (transport nodes) แต่ละตัว CMS มีหน้าที่นำเสนองานส่วนติดต่อผู้ใช้สำหรับการกำหนดค่าให้กับผู้ดูแลระบบ การเปลี่ยนแปลงใด ๆ ที่ทำผ่านส่วนติดต่อตั้งกล่าวจะถูกสื่อสารไปยังฐานข้อมูล Northbound (NB) ผ่าน API จากนั้น ovn-northd จะกำหนดรายละเอียดระดับล่างและส่งต่อไปยังฐานข้อมูล Southbound (SB) พร้อมทั้งยังอัปเดตหมายเลขเวอร์ชันของการกำหนดค่าที่ต้องการ

ต่อมา การเปลี่ยนแปลงใน SB จะถูกสื่อสารไปยังเอเจนต์ ovn-controller ที่ทำงานอยู่บนโหนดชนส่ง ซึ่งเอเจนต์เหล่านี้จะทำการอัปเดตการกำหนดค่าของ VM คอนเทนเนอร์ และ Open vSwitch ให้สอดคล้องกัน

12.9 การไหลของข้อมูลสถานะ (Status information flow)

ในทางกลับกัน การรวบรวมข้อมูลสถานะจะเหลือขึ้นทาง northbound จากโหนดชนส่งไปยัง CMS เอเจนต์ ovn-controller แต่ละตัวที่ทำงานบนโหนดชนส่งจะอัปเดตหมายเลขเวอร์ชันของการกำหนดค่าใน SB เพื่อสังเกตว่าการเปลี่ยนแปลงการกำหนดค่าที่ร้องขอได้ถูกนำไปใช้แล้ว

ovn-northd จะตรวจสอบหมายเลขเวอร์ชันของการกำหนดค่าของโหนดชนส่งแต่ละตัวใน SB และคัดลอกค่าต่ำสุดไปยัง NB เพื่อแสดงความคืบหน้าของการเปลี่ยนแปลง ผู้สังเกตการณ์ NB ใด ๆ รวมถึง CMS จึงสามารถตรวจสอบได้ว่าโหนดชนส่งทั้งหมดที่เกี่ยวข้องได้รับการอัปเดตการกำหนดค่าเรียบร้อยแล้วหรือไม่

12.10 OVN และผลิตภัณฑ์ของ Canonical

OVN สามารถทำหน้าที่เป็น SDN ให้กับ CMS ได้หลายระบบ เช่น OpenStack, LXD และ Kubernetes โดย Canonical ได้รวม OVN ไว้กับ Charmed OpenStack, Sunbeam และ MicroCloud เพื่อทำให้การติดตั้งและการกำหนดค่าเริ่มต้นง่ายขึ้น นอกจากนี้ Canonical Kubernetes ยังสามารถใช้ประโยชน์จาก OVN ได้เช่นกัน โดยให้สถาปัตยกรรมและอินเทอร์เฟซบรรทัดคำสั่งที่คุ้นเคย ผ่านการเสริมด้วยปลั๊กอิน CNI (Container Network Interface ซึ่งเป็นสเปกของ CNCF สำหรับการเขียนต่อคอนเทนเนอร์) ที่นำ OVN มาใช้งาน เช่น KubeOVN หรือ OVN-Kubernetes

OVN-Kubernetes ได้รับประโยชน์จากการความสามารถด้าน hardware acceleration และ offloading ทั้งหมด และได้รับการดูแลโดยองค์กร OVN เอง

แม้ในทางทฤษฎีจะเป็นไปได้ที่ CMS หลายตัวจะใช้ฐานข้อมูล OVN แบบกระจายร่วมกันเพียงชุดเดียว แต่ไม่ควรทำเช่นนั้น เนื่องจาก CMS แต่ละตัวมีมุมมองต่อความต้องการด้านเครือข่ายที่แตกต่างและอาจไม่เข้ากัน หากแต่ละ CMS เขียนรายละเอียดการกำหนดค่าเครือข่ายของตนเองลงในโมเดลเครือข่ายเดียวกันโดยตรงโดยไม่มีขั้นกลางสำหรับการประสานงาน ผลลัพธ์จะเกิดความไม่สอดคล้องและการข้อนับทับที่ไม่เป็นประสงค์

ดังนั้น การติดตั้ง OVN แต่ละชุดควรถูกแยกให้เฉพาะกับ CMS ของตนเอง การแลกเปลี่ยนข้อมูลเส้นทาง และการควบคุมระหว่างกันอย่างสอดคล้องสามารถทำได้โดยใช้โปรโตคอลเราท์ติงมาตรฐานของอุตสาหกรรม เช่น Border Gateway Protocol (BGP) ในความเป็นจริง แนวปฏิบัติที่ดีที่สุดในปัจจุบันคือการใช้ BGP เป็น underlay ของดาต้าเซ็นเตอร์ เพื่อให้ได้ประโยชน์จากความน่าเชื่อถือ ความสามารถในการขยายตัว และคุณสมบัติด้านความปลอดภัยของ “โปรโตคอลเราท์ติงของอินเทอร์เน็ต” แต่รายละเอียดนี้จะเป็นหัวข้อของบล็อกในอนาคต

12.11 Step-by-Step OVN Testbed Using Alpine Containers

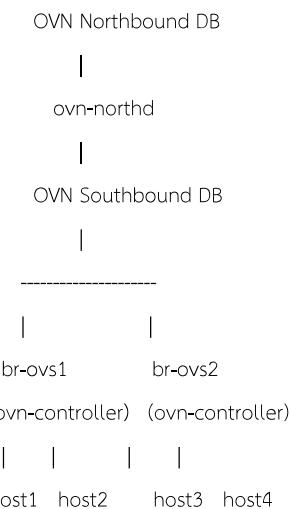
12.12 Objective

This document demonstrates how to build a minimal OVN (Open Virtual Network) test environment using Alpine Linux containers. The lab includes:

- 4 Alpine container hosts
- 2 Open vSwitch (OVS) bridges
- OVN Northbound and Southbound databases
- ovn-northd
- ovn-controller

No Kubernetes or OpenStack is required.

12.12.1 Target Topology



12.12.2 Host Requirements

The following software must be installed on the Linux host:

```

sudo apt update
sudo apt install -y openvswitch-switch ovn-host docker.io
sudo systemctl enable --now openvswitch-switch ovn-central docker
  
```

Verify installation:

```

ovs-vsctl show
ovn-nbctl show
  
```

12.12.3 Start OVN Databases

Enable OVN central services:

```

sudo systemctl enable --now ovn-central
  
```

Expose Northbound and Southbound databases:

```

ovn-nbctl set-connection ptcp:6641
ovn-sbctl set-connection ptcp:6642
  
```

12.12.4 Create Logical Network

Create Logical Switch

```
ovn-nbctl ls-add ls-test
```

Create Logical Ports

```
ovn-nbctl lsp-add ls-test lp-host1
ovn-nbctl lsp-add ls-test lp-host2
ovn-nbctl lsp-add ls-test lp-host3
ovn-nbctl lsp-add ls-test lp-host4
```

Assign IP and MAC Addresses

```
ovn-nbctl lsp-set-addresses lp-host1 "02:00:00:00:00:01 10.0.0.1/24"
ovn-nbctl lsp-set-addresses lp-host2 "02:00:00:00:00:02 10.0.0.2/24"
ovn-nbctl lsp-set-addresses lp-host3 "02:00:00:00:00:03 10.0.0.3/24"
ovn-nbctl lsp-set-addresses lp-host4 "02:00:00:00:00:04 10.0.0.4/24"
```

Create OVS Bridges

First OVS Bridge

```
sudo ovs-vsctl add-br br-ovs1
sudo ovs-vsctl set open . external-ids:system-id=ovs1
sudo ovs-vsctl set open . external-ids:ovn-remote=tcp:127.0.0.1:6642
sudo ovs-vsctl set open . external-ids:ovn-encap-type=geneve
sudo ovs-vsctl set open . external-ids:ovn-encap-ip=127.0.0.1
```

Second OVS Bridge

```
sudo ovs-vsctl add-br br-ovs2
sudo ovs-vsctl set open . external-ids:system-id=ovs2
```

Start OVN Controller

```
sudo ovn-controller &
```

Bind Logical Ports to Chassis

```
ovn-nbctl lsp-set-options lp-host1 requested-chassis=ovs1
ovn-nbctl lsp-set-options lp-host2 requested-chassis=ovs1
ovn-nbctl lsp-set-options lp-host3 requested-chassis=ovs2
ovn-nbctl lsp-set-options lp-host4 requested-chassis=ovs2
```

Verify bindings:

```
ovn-sbctl show
```

12.12.5 Create Alpine Containers

Create veth Pairs

```
for i in 1 2 3 4; do
    sudo ip link add veth${i} type veth peer name veth${i}-c
done
```

Attach veth to OVS Bridges

```
sudo ovs-vsctl add-port br-ovs1 veth1
sudo ovs-vsctl add-port br-ovs1 veth2
sudo ovs-vsctl add-port br-ovs2 veth3
sudo ovs-vsctl add-port br-ovs2 veth4
```

Run Alpine Containers

```
docker run -dit --name host1 --net=none alpine sh
docker run -dit --name host2 --net=none alpine sh
docker run -dit --name host3 --net=none alpine sh
docker run -dit --name host4 --net=none alpine sh
```

Move Interfaces into Containers

```
for i in 1 2 3 4; do
    pid=$(docker inspect -f .State.Pid host${i})
    sudo ip link set veth${i}-c netns $pid
done
```

12.12.6 Configure Container Networking

Example for host1:

```
docker exec host1 ip link set veth1-c name eth0
docker exec host1 ip addr add 10.0.0.1/24 dev eth0
docker exec host1 ip link set eth0 up
```

Repeat with appropriate IP addresses for other hosts.

12.12.7 Connectivity Test

```
docker exec host1 ping -c 3 10.0.0.4
docker exec host3 ping -c 3 10.0.0.2
```

Successful replies confirm OVN logical switching.

12.12.8 Useful Debug Commands

```
ovn-nbctl show
ovn-sbctl show
ovs-vsctl show
ovs-ofctl dump-flows br-ovs1
```

12.12.9 Conclusion

This lab demonstrates a complete OVN logical network using lightweight Alpine containers. It validates OVN control-plane translation, logical switching, and OpenFlow-based dataplane forwarding without relying on Kubernetes or OpenStack.

สรุป

Canonical OVN มอబระบบเครือข่ายแบบซอฟต์แวร์ (Software-Defined Networking) ที่ใช้งานง่าย มีความน่าเชื่อถือ และให้ผลลัพธ์ที่คาดเดาได้ SDN นี้ช่วยทำ abstraction ของรายละเอียดโครงสร้างพื้นฐานเครือข่ายในดาต้าเซ็นเตอร์ และเข้มต่อวิร์กโหลดของคุณเข้าด้วยกันได้อย่างราบรื่น ไม่ว่าจะใช้ CMS ใดก็ตาม

ด้วยการรองรับการแยกผู้ใช้อย่างปลอดภัยในตัว และประสิทธิภาพที่เพิ่มขึ้นจาก hardware acceleration และ offloading OVN จึงเป็นรากฐานสำคัญของวิสัยทัศน์ด้านเครือข่ายของ Canonical ติดตามเนื้อหาในอนาคต ซึ่งรวมถึง white paper เชิงลึกที่จะอธิบายโครงสร้างภายในและการทำงานของ OVN อย่างละเอียดมากขึ้น เช่น วิธีการรัน OVN บน NVIDIA BlueField DPU และการนำไปใช้เพื่อบังคับและจัดการการควบคุมความปลอดภัยแบบกระจายศูนย์

12.13 สรุป

จากความพยายามตอบสนองความต้องการของเน็ตเวิร์กในรูปแบบ SDN ทำให้มีค่อนໂທຣລເລອ່ວเกิดขึ้นเป็นจำนวนมาก ปัจจุบันโน้ตเคนได้เลิฟ์และโอนอสຄอนໂທຣລເລອ່ວหลักที่ถือได้ว่ารับใช้งานอย่างแพร่หลาย ซึ่งในบทต่อไปจะแสดงตัวอย่างการทำงานของໂພຣໂທຄອລທັ້ງສອງ

12.14 คำถาม

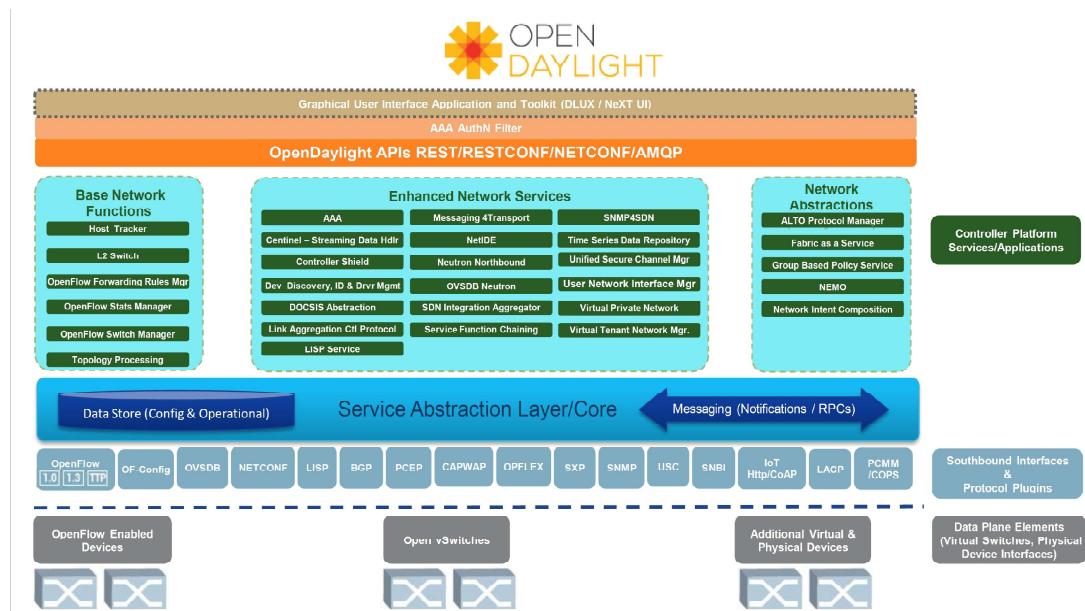
โอเพนเดไลท์ (OpenDaylight)

โอเพนเดไลท์เริ่มการพัฒนาตั้งแต่ปี ค.ศ.2013 โดยบริษัท IBM และ CISCO โอเพนเดไลท์เป็น SDN แรกที่ถูกออกแบบเพื่อรองรับขาเข้าบานด์โปรดักโคลที่นอกเหนือจากโอเพนพอร์ตทำให้โอเพนเดไลท์สามารถใช้งานกับเน็ตเวิร์กที่มาจากการผู้ผลิตที่หลากหลาย โอเพนเดไลท์สามารถทำหน้าที่เป็นคอนโทรลเลอร์สำหรับ distributed คอนโทรลเลอร์ ประกอบด้วยการติดต่อ กับผู้ใช้ และ API สำหรับการบริหารจัดการแบบรวมศูนย์ พร้อมทั้งทำหน้าที่เป็นօเพนเดสต์เทอร์สำหรับ Virtual Network Functions (VNF)

13.1 สถาปัตยกรรมของโอเพนเดไลท์

โอเพนเดไลท์เวอร์ชันแรกถูกปล่อยออกมายังเดือนกุมภาพันธ์ ค.ศ.2014 โดยใช้ชื่อว่า Hydrogen จากนั้นตามมาด้วย Helium ในเดือนกันยายน ค.ศ.2014 ซึ่ง Helium ถือเป็นเวอร์ชันสำคัญของการพัฒนาแพลตฟอร์มในเวอร์ชันต่อๆ มา

ปัจจุบันโอเพนเดไลท์ออกมายังเวอร์ชันที่ 11 มีชื่อเป็นทางการว่า Sodium จากการตั้งชื่อตามตารางธาตุดังแสดงในรูปที่ 13.1



รูปที่ 13.1: สถาปัตยกรรมทั่วไปของ Opendaylight

โอเพนเดไลท์แบ่งการทำงานออกเป็น 3 เลเยอร์หลัก ได้แก่ 1.เน็ตเวิร์กแอปพลิเคชันและเซอร์วิส2. ส่วนของคอนโทรลเลอร์แพลตฟอร์มและ 3. ส่วนของขาเข้าบานด์ฯลฯ

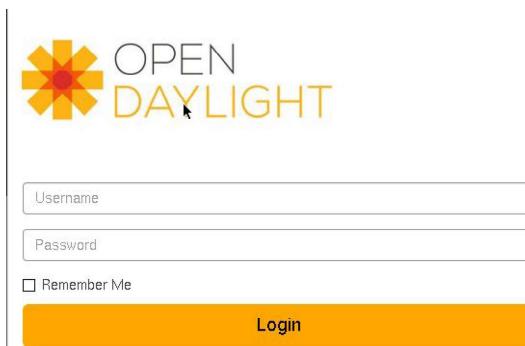
13.1.1 เน็ตเวิร์กแอปพลิเคชันและเซอร์วิส

ส่วนของเน็ตเวิร์กแอปพลิเคชันและเซอร์วิสเป็นส่วนบนสุดของโอเพนเด้ไอที เพื่อควบคุม จัดการ และค่อยตรวจสอบการทำงานของเน็ตเวิร์ก แอปพลิเคชันและเซอร์วิสต่างๆ ขึ้นอยู่กับแพลตฟอร์มที่ใช้งาน โดยจะประกอบด้วย ส่วนของนอร์ทบาวน์ด API เพื่อรับการใช้งานของแอปพลิเคชัน ในส่วนของนอร์ทบาวน์ดยังประกอบด้วย ส่วนօโคสเตเทอร์ของแอปพลิเคชันเพื่อการองรับความซ้ำของระบบอย่างคลาวด์หรือ NFV ทั้งนี้ โอเพนเด้ไอที รองรับทั้งการใช้งาน OSGi framework ในกรณีที่ทำงานบน address space เดียวกัน หรือ REST API ในกรณีที่ทำงานเป็นคอนโทรลเลอร์ภายในเครื่องเดียวกัน หรือการใช้งานที่เครื่องอื่น เน็ตเวิร์กแอปพลิเคชันที่สำคัญ ได้แก่



รูปที่ 13.2: ส่วนของเน็ตเวิร์กแอปพลิเคชันและเซอร์วิส

- openDayLight User eXperience (DLUX) เป็นส่วนที่ใช้ติดต่อกับผู้ใช้งานทางหน้าเว็บ แสดงโ拓โพโลยี ของเน็ตเวิร์กที่ทำงาน และแสดงรายละเอียดของ YANG



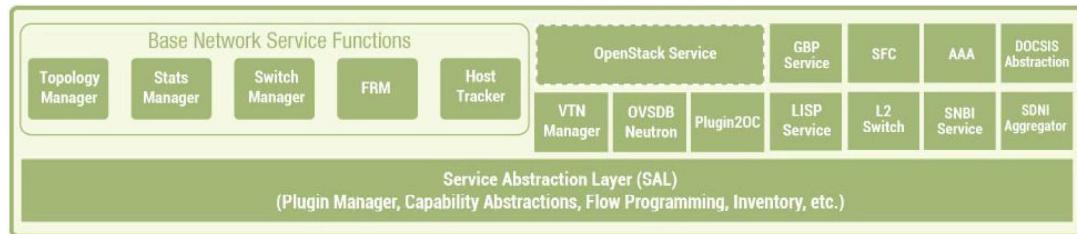
รูปที่ 13.3: เว็บสำหรับการผู้ใช้งาน

- DDos protection เป็นแอปพลิเคชันเพื่อป้องกันการรุกรุกโดย Denial of Service (DDoS) เป็นการใช้ส่วนของ นอร์ทบาวน์ด REST API เพื่อค่อยตรวจสอบพฤติกรรมของทราฟฟิก
- VTN coordinator เป็นแอปพลิเคชันที่ใช้เพื่อกำหนด VTN และการประสานการทำงานร่วมกันกับหลายคอนโทรลเลอร์

13.1.2 คอนโทรลเลอร์แพลตฟอร์ม

ส่วนของคอนโทรลเลอร์แพลตฟอร์มถือเป็นหัวใจสำคัญของโอเพนเด้ไอที โดยส่วนนี้จะเชื่อมต่อกับนอร์ทบาวน์ด API ไปยังแอปพลิเคชันให้สามารถกำหนดการทำงานของอุปกรณ์เน็ตเวิร์กทั้งรูปแบบเวอร์ชวลและเป็นเครื่องทำงานจริง ประกอบด้วยส่วนย่อย ได้แก่ ส่วน Base Network Service Functions (BNSFs), Platform Network Service Function และ Service Abstraction Layer (SAL)

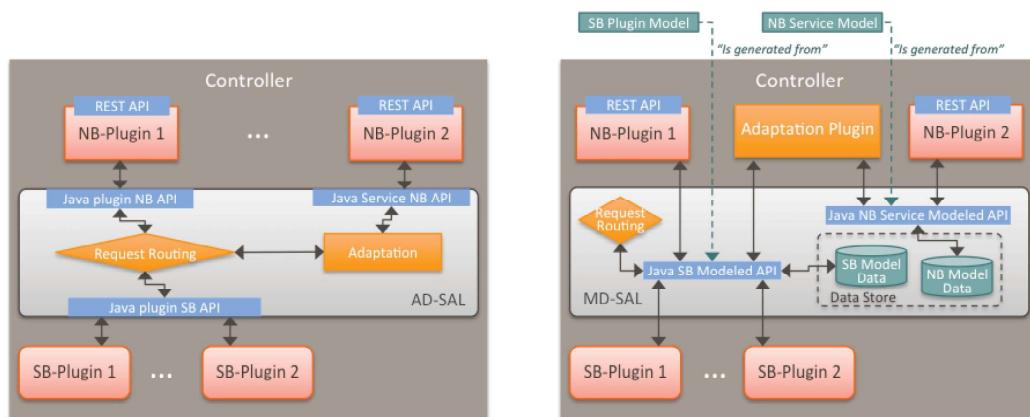
- Base Network Service Functions (BNSFs) ทำหน้าที่เก็บข้อมูลทางสถิติ รวบรวมข้อมูลของอุปกรณ์ต่างๆ ที่ประมวลในเน็ตเวิร์ก และประสิทธิภาพของอุปกรณ์เหล่านั้น ประกอบด้วยส่วนการจัดเก็บข้อมูลของโ托โพโลยี (Topology Manager) ข้อมูลทางสถิติของสวิตช์ (Statistics Manager) ข้อมูลของตัว



รูปที่ 13.4: องค์ประกอบของคอนโทรลเลอร์แพลตฟอร์ม

สวิตช์ (Switch Manager) อัพเดทไฟล์ แก๊ซไฟล์ที่ขัดແย়েঁกন (Forward Rules Manager) ติดตาม ตำแหน่งของไอดีในเน็ตเวิร์ก (Host tracker)

- Platform Network Service Function ทำหน้าที่ที่เกี่ยวข้องกับการทำงานของเน็ตเวิร์กและส่วนของ เซอร์วิสที่ต้องการเพิ่มเติม ประกอบด้วย นอร์ทบานด์API เพื่อให้ออกพลิกเซ็นระบุความต้องการ คอน- โทรลเลอร์อาจจัดเตรียมเพื่อตอบสนองต่อความต้องการ (Affinity Metadata Service) สร้างเวอร์ชวล เน็ตเวิร์กแบบอิจิกคลอให้กับผู้ใช้แต่ละคน (Virtual Tenant Network, VTN) รองรับการทำงานของ สวิตช์ในเลเยอร์ 2 และการให้บริการแบบต่อเนื่อง (Service Function Chain) เป็นต้น
- Service Abstraction Layer (SAL) เพื่อสนับสนุนการใช้งานเช้าที่บานด์และจัดเตรียมเซอร์วิสเพื่อ รองรับการทำงานของโมดูลและเน็ตเวิร์กแอปพลิเคชัน เดิมเวอร์ชัน Hydrogen ใช้รูปแบบการทำงาน แบบ API-driven Service Abstraction Layer (AD-SAL) ทำให้เกิดข้อจำกัดที่คอนโทรลเลอร์ต้องรู้ถึง ทุกอุปกรณ์ในเน็ตเวิร์กและมีไดร์เวอร์จากอุปกรณ์เพื่อรับ Helium ได้เปลี่ยนรูปแบบเป็น Model- driven Service Abstraction Layer (MD-SAL) ทำให้คอนโทรลเลอร์ไม่จำเป็นต้องรู้ถึงอุปกรณ์ที่ทั้งหมด ทำให้สามารถรองรับอุปกรณ์ได้หลายรูปแบบ รวมไปถึงเช้าที่บานด์พร้อมกันที่ใช้



รูปที่ 13.5: เปรียบเทียบ SAL และ MD-SAL

13.1.3 เช้าที่บานด์อินเทอร์เฟซ (Southbound Interface) และโปรโตคอลเสริม (Protocol Plugins)

เพื่อรับเช้าที่บานด์อินเทอร์เฟซที่หลากหลาย รวมถึงความปลอดภัยของการสื่อสารระหว่างคอนโทรลเลอร์ กับอุปกรณ์เน็ตเวิร์ก โดยเช้าที่บานด์ที่สำคัญ ได้แก่



รูปที่ 13.6: เขาท์บานด์อินเทอร์เฟช (Southbound Interface) และโปรโตคอลเสริม (Protocol Plugins)

- ส่วนเสริมโอเพนโฟล์ว (OpenFlow Plugin) รองรับโอเพนโฟล์วโปรโตคอล เวอร์ชัน 1.0 และ 1.3 รวมถึงรองรับรูปแบบ Table Type Patterns (TTPs) เพื่อให้ออเพนโฟล์วคอนโทรลเลอร์สามารถสื่อสารกับสวิตซ์เพื่อใช้วีร์ชันดังต่อไปนี้
- ส่วนเสริม โอเพนวีสวิตซ์ (Open vSwitch Plugin) รองรับการทำงานของโปรโตคอล OVSDB เพื่อคอนฟิกภาระน้ำหนักและชั้นโอเพนวีสวิตซ์
- ส่วนเสริม SNMP (SNMP Plugin) เพื่อให้สามารถใช้งาน Simple Network Management Protocol (SNMP) กับอุปกรณ์เน็ตเวิร์กที่ว่าไป การคอนฟิกภาระน้ำหนักและชั้นโอเพนโฟล์วจะทำผ่านฟอร์แมตเดียวกัน เช่น ACL หรือ VLAN
- ส่วนเสริม NETCONF (NETCONF Plugin) เพื่อให้ออเพนเด้ลایท์สามารถรองรับการทำงานของโปรโตคอล NETCONF

13.2 ติดตั้งโอเพนเด้ลัยท์

คอนโทรลเลอร์โอเพนเด้ลัยท์เป็น Java Virtual Machine (JVM) ทำให้สามารถรองรับทุกรอบปัญญาติการและชาร์ดแวร์หากสามารถรองรับ Java การพัฒนาของโอเพนเด้ลัยท์อาศัยเครื่องมือที่สำคัญ ดังนี้

- Maven รองรับการติดตั้งอัตโนมัติสำหรับโปรเจกต์ของ Java โดยโมดูล Maven จะใช้สคริปต์ไฟล์ pom.xml (Project Object Model) เพื่อระบุไฟล์ที่ต้องการและการเริ่มการทำงาน
- Open Service Gateway Initiative (OSGi) เป็นเฟรมเวิร์กทำงานรองรับเบื้องหลังของโอเพนเด้ลัยท์ ให้สามารถเรียกใช้ส่วนโปรแกรมเพิ่มเติม (JAR file) ได้
- Java เพื่อใช้พัฒนาแอปพลิเคชันของโอเพนเด้ลัยท์ และ feature ต่าง ๆ ของโอเพนเด้ลัยท์คอนโทรลเลอร์
- Karaf เป็นแอปพลิเคชันคอนโทรลเลอร์เนอร์เริมบน OSGi เพื่อให้สะดวกต่อการติดตั้งแอปพลิเคชันเพิ่มเติม
- REST APIs ส่วนของนอร์ทบานด์ API เช่น topology manager, host tracker, flow programmers และ อื่น ๆ

โอเพนเด้ลัยท์จะจัดเตรียมส่วนของนอร์ทบานด์ API เพื่อรับรู้การร้องขอจากแอปพลิเคชัน โดยมีส่วนของ OSGi และ REST เพื่อรับการสื่อสารแบบสองทางสำหรับนอร์ทบานด์ API โดยที่ OSGi ใช้สำหรับแอปพลิเคชันที่ทำงานบนเน็ตเวิร์กเดียวกัน ในขณะที่ REST ในรูปแบบเว็บ สำหรับแอปพลิเคชันที่อยู่บนเน็ตเวิร์กอื่น สำหรับในส่วนของเขาท์บานด์ประกอบไปด้วยโปรโตคอลที่สามารถนำมาใช้ในรูปแบบ plugin เช่น โอเพนโฟล์ว เวอร์ชัน 1.0 และ 1.3 โดยการเชื่อมต่อผ่าน Service Abstraction Layer (SAL)

13.2.1 การติดตั้งโอเพนเด้ลัยท์

การทดสอบจะเป็นการติดตั้งโอเพนเด้ลัยท์บน Ubuntu เวอร์ชัน 18.04

- อัปเดทส่วนของระบบปฏิบัติการ

```
odl@ubuntu:~$ sudo apt-get update
odl@ubuntu:~$ sudo apt-get upgrade
```

2. ติดตั้ง Java JDK

```
odl@ubuntu:~$ sudo apt-get install openjdk-8-jdk
odl@ubuntu:~$ export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

3. ติดตั้งโอเพนเดไลท์ ในที่นี้เลือกเวอร์ชัน Boron ผู้อ่านสามารถเปลี่ยนโหลดเวอร์ชันอื่นได้ตามต้องการ

```
wget https://nexus.opendaylight.org/content/repositories/public/
      org/opendaylight/integration/distribution-karaf/0.5.4-Boron-SR4
      /distribution-karaf-0.5.4-Boron-SR4.zip
```

4. จัดเตรียมพื้นที่ เพื่อติดตั้ง บायไฟล์ไปไว้ที่เดียวกัน

```
sudo mkdir /usr/local/karaf
mv distribution-karaf-0.5.4-Boron-SR4.zip /usr/local/karaf/
sudo mv distribution-karaf-0.5.4-Boron-SR4.zip /usr/local/karaf/
sudo unzip /usr/local/karaf/distribution-karaf-0.5.4-Boron-SR4.zip
      -d /usr/local/karaf/
```

5. ติดตั้งให้ karaf สามารถทำงานบน user space เพื่อสะดวกต่อการเข้าถึง

```
sudo update-alternatives --install /usr/bin/karaf karaf /usr/
      local/karaf/distribution-karaf-0.5.4-Boron-SR4/
      bin/karaf 1
```

6. เรียกใช้ karaf

```
sudo -E karaf
```

7. ผลที่ได้เข้าสู่การทำงานของโอเพนเดไลท์

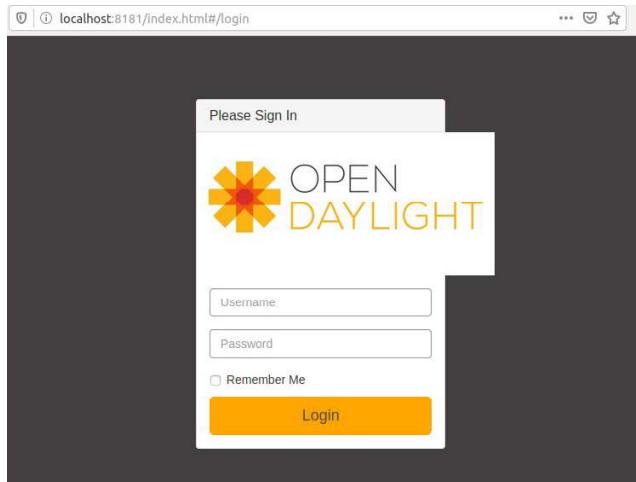
```
100%
[=====]
```

```
Hit <tab> for a list of available commands
and [cmd] --help for help on a specific command.
Hit <ctrl-d> or type system:shutdown or logout to shutdown
OpenDaylight.
```

8. ทดสอบการติดตั้งฟีเจอร์และแสดงรายการฟีเจอร์ของโอเพนเดีลท์ผ่าน Karaf (เนื่องจากบางฟีเจอร์หากไม่สามารถทำงานพร้อมกันได้ จึงไม่แนะนำให้ติดตั้งฟีเจอร์พร้อมกัน)

```
opendaylight-user@root>feature:install odl-dlux-all
opendaylight-user@root>feature:list -i
Name           | Version     | Installed | Repository
| Description
-----
standard       | 3.0.7       | x         | standard-3.0.7
               | Karaf standard feature
config        | 3.0.7       | x         | standard-3.0.7
               | Provide OSGi ConfigAdmin support
region        | 3.0.7       | x         | standard-3.0.7
               | Provide Region Support
...
...
```

9. จากการที่ได้ลงฟีเจอร์ DLUX ทำให้สามารถเพิ่มส่วนของการติดต่อกับผู้ใช้ผ่านทางเบราว์เซอร์ได้ ที่หมายเลข 8181 ดังแสดงในรูปที่



รูปที่ 13.7: ฟีเจอร์ DLUX ของโอเพนเดีลท์

13.2.2 Karaf

การทำงานของโปรแกรมที่ว่าไปประกอบด้วยส่วนย่อยหลายส่วน การใช้งานของ Karaf ช่วยให้การจัดการกับแอปพลิเคชันสะดวกขึ้น อาศัยการทำงานในรูปแบบฟีเจอร์ (feature) รูปที่ 13.8 แสดงการทำงานของ Karaf การใช้งานของ Karaf ค่อนข้างสะดวก ผ่านหน้าของ console ของแอปพลิเคชัน คำสั่งสำคัญดังนี้

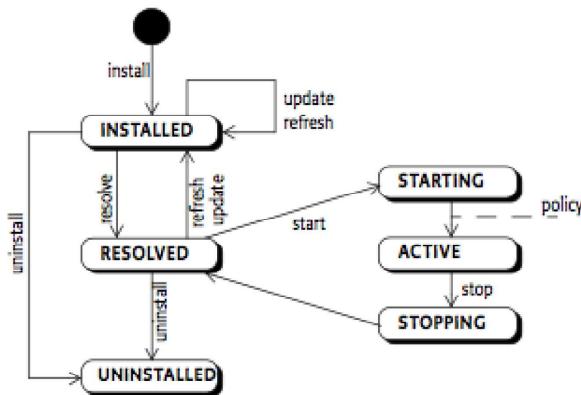
- ติดตั้งฟีเจอร์เพิ่ม

```
feature:install <feature-name>
```

- ติดตั้งหลายฟีเจอร์เพิ่มพร้อมกัน

```
feature:install <feature1-name> <feature2-name> ... <featureN-name>
```

- แสดงรายการ (list) ของฟีเจอร์ที่มีอยู่



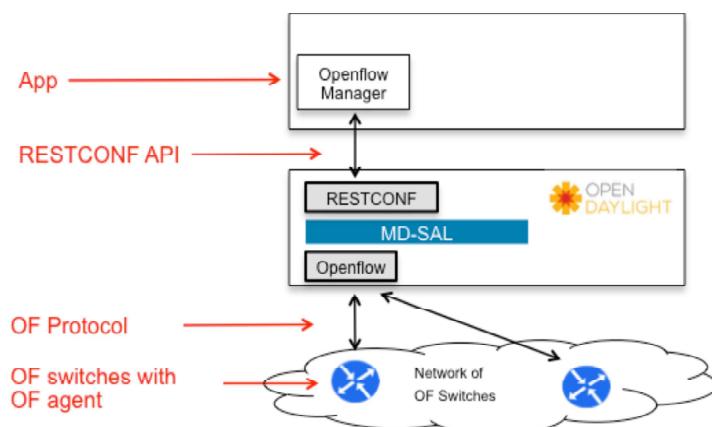
รูปที่ 13.8: การทำงานของ Karaf

`feature:list`

- แสดงรายการ (list) ของฟีเจอร์ที่ได้ติดตั้งไปแล้ว

`feature:list -i`

นอกจากการติดตั้งโปรแกรมโอเพนเดไฟล์แล้ว อีกหนึ่งโปรแกรมที่แนะนำคือ OpenDaylight OpenFlow Manager (OFM) เป็นโปรแกรมที่ทำงานบนโอเพนเดไฟล์เพื่อคูโอเพนโพลาร์โน้ติ กำหนดเส้นทาง (path) ของโอเพนโพลาร์และเก็บค่าสถิติ [86] โดยมีรูปแบบการทำงานดังแสดงในรูปที่ 13.9



รูปที่ 13.9: สถาปัตยกรรมของ OFM [86]

13.2.3 การทดสอบการทำงานเบื้องต้น

การทดสอบโอเพนเดไฟล์ในส่วนนี้จะใช้เวอร์ชัน Boron โดยการเรียกชื่อของโอเพนเดไฟล์จะเป็นตามตารางธาตุเริ่มจาก Hydrogen จนกระทั่งเวอร์ชันล่าสุดคือ Magnesium เดือนมีนาคม ค.ศ. 2020 openDaylight User eXperience (DLUX) Dlux ถือเป็น feature แรกที่ผู้อ่านควรเริ่มต้นด้วย ใช้แสดงโพลาร์และโหนดที่อยู่ในเน็ตเวิร์ก

ชื่อของไฟจ่อร์	รายละเอียด	ชื่อไฟจ่อร์ใน Karaf	
การพิสูจน์ทราบตัวตน	สำหรับการทำการพิสูจน์ทราบตัวตน การอนุญาตใช้งานและ Accounting	odl-base-all odl-aaa-authn	
RESTCONF API DLUX	เพื่อต่อสื่อสารระหว่าง REST API และ MD-SAL ယุชอร์อินเทอร์เฟซ	odl-restconf odl-dlux-core odl-dlux-node	
โอเพนโฟล์ว L2 Switch	การใช้งานร่วมกับโอเพนโฟล์ว สำหรับการทำงานในเลเยอร์ 2 หรืออีเทอร์เน็ต ผ่านการทำงานกับโอเพนโฟล์ว	odl-openflowplugin-all odl-l2switch-all odl-l2switch-switch-ui	

ตารางที่ 13.1: Physical features of LoRa and NB-IoT

13.3 สรุป

...

13.4 คำถ้าม