WEEK-7:

Lecture 33: Software-Defined Networking (SDN)-Part I

Introduction

- SDN is a way to restructure networks by centralizing control and separating it from packet forwarding, making networks easier to program and manage.
- Motivation scenario: In a normal L3 network running OSPF, each switch/route decides locally; when a switch is attacked or fails, alternate routing lacks a single coordinating "brain." SDN addresses this by centralizing decisions.

Key Features / Concepts

- Planes and separation
    - Data Plane: forwarding devices (switches/routers) apply rules in flow tables.
    - Control Plane: centralized controller (network OS) computes policies and installs rules network-wide.
    - Application Plane: apps (security, routing, traffic engineering, etc.) drive requirements.
- APIs and interfaces
    - Southbound API: controller ↔ data plane; OpenFlow used here to add/modify/delete flow rules.
    - Northbound API: controller ↔ applications; exposes abstractions/services to apps.
    - East-/Westbound APIs: coordination among multiple controllers.
- Flow-rule model (flow table = "match → action" entries)
    - Match fields (examples shown): priority, protocol (e.g., TCP), IP destination, ports.
    - Actions: forward to specific port, send to controller, "normal" processing.
    - Fields are fixed within an OpenFlow version; different versions provide different field sets.
- Miss handling and setup delay
    - If no rule matches, switch sends PACKET-IN to controller; controller chooses a rule and installs it; typical new-rule delay ≈ 3–5 ms.

- Rule lifetimes (timeouts)
    - Hard timeout: delete rule after a fixed time (often used to reset/clean a switch).
    - Soft timeout: delete rule if no matching packets arrive for a while (frees limited rule space).
- Origins and adoption (context in slides)
    - 2006 SANE (centralized security idea) → 2008 OpenFlow project (SIGCOMM) → 2009 OpenFlow v1.0; ONF formation; industry announcements to incorporate SDN.
- Implementations (named in slides)
    - OpenFlow switch software: Indigo, LINC, Pantou, of13softswitch, Open vSwitch (popular).
- Clarification
    - SDN (concept/architecture) ≠ OpenFlow (a southbound protocol used to realize controller–switch communication).

Advantages

- Centralized global view enables fast, coherent reactions to failures/attacks; controller can reprogram paths across the whole network.
- Breaks vendor lock-in of "hardware+OS+apps" stacks; supports dynamic, application-specific configuration through software.
- Practical note from slides: large operators (e.g., data centers) adopt SDN in phases to reduce operational cost and link-failure impact.

Limitations / Challenges

- Traditional (problem statement from slides)
    - Distributed forwarding, no central control; vendor-specific stacks hinder agility; per-device local view only.
- SDN design challenges (emphasized in Part-I)
    - Rule Placement
        - Flow tables live in TCAM (specialized fast memory) which is small and expensive—only limited rules fit.
        - Objective: place rules to fit TCAM and reduce PACKET-IN frequency/latency.

- Controller Placement
    - Raised for next lecture; placement/load impacts responsiveness and control traffic.
- Operational constraints
    - OpenFlow match fields cannot be arbitrarily changed within a version; new matches require different versions/approach.
    - New rule installation adds setup delay (≈ 3–5 ms).

Applications

- Data-center networking (called out in slides) with phased migration to SDN.
- Network applications above the controller: security, routing, traffic engineering, other control logic.

Comparisons (Traditional vs. SDN)

Aspect	**Traditional network**	**SDN approach**
Control location	Distributed in each L3 switch/router (e.g., OSPF).	Centralized controller with global view.
Forwarding basis	Per-device routing tables built by distributed protocols.	Flow tables with controller-defined rules (match→action).
Vendor coupling	Tight coupling: hardware + OS + apps per device.	Decoupled planes; apps drive policies via APIs.
Failure handling	Re-routing is decentralized; no single coordinator.	Controller recomputes rules and pushes network-wide.
Miss handling	N/A (local protocol decisions).	PACKET-IN to controller; installs rule (≈3–5 ms).
Rule storage	N/A (routing table memory).	TCAM-backed flow tables (fast but limited/costly).

Recap / Summary

- SDN centralizes control (controller), separates planes, and programs forwarding via flow rules using southbound APIs (OpenFlow).
- Operational mechanics: flow matches, actions, PACKET-IN on miss, hard/soft timeouts, version-specific match fields.
- Benefits: programmability, global optimization, faster coordinated recovery; Key issues to handle: rule placement under TCAM limits and controller placement (covered next).

Lecture 34: Software-Defined Networking (SDN) Part II

Introduction

- Focus shifts from rule placement to the second core problem in SDN: controller placement and related control-plane design.
- Quick recap: SDN separates control from data plane; rules reside in switch flow tables with TCAM limits; misses trigger PACKET-IN and add ≈3–5 ms setup delay; OpenFlow is used on the southbound interface.

Key Features / Concepts

- APIs in SDN
    - Southbound API: between control plane and data plane; OpenFlow used to program switches.
    - Northbound API: between control plane and application plane; standard application interfaces continue to work.
    - East/Westbound APIs: between multiple controllers for coordination in multi-controller deployments.
- Controller responsibilities
    - Define flow rules per application requirements; handle incoming requests from switches; minimize control delay during rule setup.
    - Practical note: single-threaded controllers handle roughly a few hundred requests/sec; multithreading increases capacity.
- Logical vs physical proximity
    - Controller appears logically one hop from switches, though physically it may be multiple hops away.

Advantages

- Multi-controller architectures (ring/mesh/tree) improve resilience and scalability versus a single controller, while keeping a logical one-hop view.
- SDN enables enhanced security policy chains (e.g., Firewall → IDS → Proxy) to be enforced coherently through controller logic.

Limitations / Challenges

- Few controllers in a large network can congest control channels with many PACKET-IN messages; controller placement must balance load and latency.

- Even with centralized logic, new flow setup still incurs control latency; designs must minimize rule-misses and avoid excessive controller round-trips.

Applications

- Security service chaining: sequence traffic through functions like Firewall, IDS, Proxy under controller guidance to reduce data-plane ambiguity.
- Education/experimentation: emulate SDN topologies, try flat vs hierarchical controller designs, and measure control overheads and failover.

Controller–Switch Topology Options

- Flat architecture
    - Each switch sends PACKET-IN directly to a controller and receives flow rules in return; simplest logical view.
- Hierarchical (tree)
    - Controllers arranged in tiers; switches attach to lower-tier controllers; higher tiers coordinate.
- Ring
    - Multiple controllers connected in a ring; each switch associates with one controller in the ring.
- Mesh
    - Controllers interconnect in a mesh; switches may have alternative controller paths improving fault-tolerance and reliability.

Control Mechanisms

- Centralized control
    - Single controller governs the entire network; simpler coordination but potential bottleneck/failure point.
- Distributed control
    - Multiple controllers manage different subnetworks; reduces latency and scales control handling.
- Backup controller

        - Standby replica of main controller; takes over seamlessly upon failure to ensure uninterrupted management.

Security Notes

- SDN can enhance security by explicitly steering flows through ordered middleboxes (policy chain), mitigating data-plane ambiguity; example chain: HTTP → IDS → Proxy → Firewall with controller-installed rules.
- The lecture cites a SIGCOMM'13 approach showcasing policy-chain enforcement using SDN.

Experimentation & Tools

- Emulation/simulation
    - Use emulators/simulators to evaluate SDN performance; infrastructure deployment and controller placement must support OpenFlow.
    - Controllers can be local or remote (identified by IP and port).
- Mininet
    - Build virtual OpenFlow networks on a single machine; supports remote/local controllers; Python-based; ideal for SDN experiments.
- Controller software
    - POX, NOX, Floodlight, OpenDaylight, ONOS; OpenDaylight and ONOS highlighted as popular options.

Comparisons

Topic	Single controller	Multiple controllers
Scalability	Limited by one node's capacity; higher PACKET-IN load.	Load spread across controllers; better throughput.
Fault tolerance	Single point of failure unless backup used.	Ring/mesh allows alternate paths and failover.
Latency to switches	May be high if far physically.	Potentially lower with regional controllers.
Coordination need	Minimal.	Requires East/Westbound APIs among controllers.

Recap / Summary

- SDN Part-II centers on controller placement and control-plane structuring: APIs (north/south/east-west), logical one-hop view, and architectures (flat, tree, ring, mesh).
- Control strategy can be centralized or distributed; backup controllers provide resilience; security policy chaining benefits from SDN orchestration.
- Practical work uses Mininet and controller frameworks (e.g., OpenDaylight/ONOS) to deploy, test, and measure designs; overall SDN performance hinges on both rule placement and controller placement while managing added PACKET-IN overhead.

Lecture 35: Software-Defined IoT Networking – Part I

Introduction

- Goal: show how SDN can be applied to IoT to solve long-standing networking problems and make IoT more efficient.

- Reference context: IoT reference models layer the stack from physical devices and connectivity up to data accumulation/abstraction and applications; SDN will be mapped onto this landscape.

Key Features / Concepts

- IoT layering snapshots
    - Forum model layers: Physical devices & controllers → Connectivity → Edge computing → Data accumulation → Data abstraction → Application → Collaboration & processes.
    - Tiered view: Context-aware tier, Network tier, Application tier with multiple modules per tier.
- Why integrate SDN into IoT?
    - Intelligent routing decisions: centralized controller computes routes/policies for heterogeneous IoT traffic.
    - Simpler information collection/analysis/decision-making: controller has a global view to orchestrate data flows.
    - Visibility of network resources: manage per user/device/application requirements more easily.
    - Intelligent traffic-pattern analysis and coordinated actions: controller drives policy-based responses.
- SDN-enabled IoT data path (high-level)
    - IoT devices (sensors/actuators/RFID) send data via mobile/fixed access to an aggregation layer → transport network → packet gateways/data center.
    - SDN controller overlays this end-to-end path to orchestrate protocols, control devices, and enforce service logic.
- Where SDN adds control in IoT
    - End device control: centralized coordination of sensors, actuators, RFID tags.
    - Access: rule placement while considering device heterogeneity and mobility.
    - Transport/backbone: rule placement and traffic engineering.
    - Data center: flow classification and enhanced security.

Advantages

- Centralized intelligence makes heterogeneous IoT networks manageable and policy-driven at scale.

- SDN improves orchestration among devices/protocols across access, transport, and data-center segments for end-to-end behavior.
- Enables dynamic rule placement and traffic engineering tuned to mobility and application demands.

Limitations / Challenges

- Sensor networks (core IoT substrate) present constraints:
    - Real-time reprogramming of nodes is generally infeasible in traditional WSNs.
    - Vendor-specific stacks with no single standard; heterogeneous layers per vendor.
    - Resource constraints: limited CPU and memory; heavy computation and many control programs are not viable on nodes.
- These challenges motivate SDN-based designs that externalize control and keep nodes simple.

Applications

- Applying SDN to WSNs for IoT control:
    - Centralized rule computation for forwarding (threshold-based, ID-based, etc.) and policy enforcement.
    - End-to-end service logic (e.g., access policy + backbone engineering + DC flow classification) coordinated by the controller.

Comparisons (Approaches within SDN-WSN landscape)

- Sensor OpenFlow (SOF)
    - Idea: flow-based forwarding in WSNs controlled by a central entity; two forwarding modes:
        - Value-centric: forward only if a sensed value exceeds a threshold.
        - ID-centric: forward based on source node ID.
    - Note: real-life deployment not reported in the lecture.
- Soft-WSN (proposed by the lecturer's group)
    - Components: Sensor Device Management (sensor management; delay management for dynamic sensing intervals; active–sleep management for power states).
    - Topology Management:

        - Node-specific: modify forwarding logic of a specific node.
        - Network-specific: forward all traffic of a node, or drop all traffic of a node.
    - Experimental evidence (real hardware):
        - Higher packet delivery ratio versus traditional WSN.
        - Fewer replicated data packets than traditional WSN.
        - More control packets due to PACKET-IN interactions for new flows to obtain controller logic.
- SDN-WISE
    - A software-defined WSN platform with a flow table designed for sensor nodes and APIs to program nodes in real time.
    - Stack overview: IEEE 802.15.4 + MCU; above MAC/PHY a forwarding layer with flow rules and an INPP (in-network packet processing) layer; sinks and emulated nodes supported.

Recap / Summary

- SDN brings centralized routing, orchestration, and policy control to IoT across access, transport, and data-center segments, improving manageability and efficiency.
- For sensor networks, SDN variants (SOF, Soft-WSN, SDN-WISE) address real-time control, forwarding logic, and programmability while coping with constrained resources.
- Empirical results (Soft-WSN) show improved delivery and reduced replication at the cost of more control messages, reflecting controller-driven operation.
- Overall, SDN-based approaches significantly improve IoT/WSN performance compared to traditional methods and set the stage for SDN-assisted mobile IoT in the next lecture.