CAN201 - Week 9

Network Layer - Control Plane (AS, OSPF, BGP, SDN, ICMP, SNMP)

InubashiriLix (Github)

0. 总体心智模型：这一讲在说什么?

0.1 已经知道的 & 现在要补的

- 前几讲你已经有的“层次大图”：
  - 传输层：TCP/UDP → 端到端逻辑连接、可靠性、拥塞控制；
  - 网络层：IP → * 每一跳怎么转发 *（数据平面）；
  - 路由算法：LS / DV → * 全网怎么选路 *（控制平面，理想化模型）。
- **这一讲**在干什么：
  1. 把“理想化的路由算法”放回真实 Internet：
     - Internet 是 * 很多网络拼起来的网络 *（AS）；
     - AS 内部：OSPF 等 **intra-AS** 协议；
     - AS 之间：BGP **inter-AS** 协议；
     - 二者共同决定路由表。
  2. 展示另一种控制平面实现思路：*SDN（集中控制）*：
     - 把“算路由”的脑袋搬到一个控制器里；
     - 底下的交换机只是执行 match→action。
  3. 最后补上两个“配角”但考试必考：
     - ICMP：IP 的“吐槽/报错/诊断”信道（ping、traceroute）；
     - SNMP：运维视角的 **network management** 协议。

0.2 一张逻辑链

```
IP 数据平面：一跳一跳转发 → 需要路由表

路由表从哪来？
  → AS 内：OSPF 等 intra-AS 协议
  → AS 间：BGP inter-AS 协议
  → 也可以从 SDN 控制器下发（集中控制）

跑着跑着会出错、会坏：
  → ICMP 用来报告“路由/转发层面的错误、诊断信息”

整网日常运维、监控：
  → SNMP：管理端拉 MIB，设备上报 trap
```

接下来按这条线展开。

1. Internet 是一堆 AS 拼起来的：为什么需要层次化路由?

1.1 单一“扁平网络”的不可行性

- 之前的 LS / DV 模型默认：
  - “全网所有路由器都跑同一种协议，视图都是一张扁平的大图”；
  - 每个路由器要么：
    - LS：知道全网所有链路状态；
    - DV：跟所有邻居交换到所有目的的距离。
- 现实问题：
  1. 规模：目的网络 = * 十亿级 *，不能每个 prefix 都进每个路由表；
  2. 控制权：每个运营商/组织想 **自己控制**自己网里的路由策略；
  3. 链路负载：若整网都互相广播 LS/DV 信息，控制报文本身就能挤爆链路。

1.2 解决方案：Autonomous Systems（AS）

- *AS（Autonomous System）*：
  - 一块由同一个管理者控制的网络；
  - 有自己的内部路由协议（IGP：OSPF、RIP、IS-IS…）；
  - 对外表现为一个统一的“整体”，通过少数 **gateway routers** 接别的 AS。
- 于是设计变成两层：
  1. *Intra-AS routing*（域内，intra-domain）
     - 在一个 AS 里面怎么路由；
     - 协议如：OSPF，IS-IS，EIGRP…
  2. *Inter-AS routing*（域间，inter-domain）
     - AS 与 AS 之间怎么路由；
     - 协议：唯一主角 BGP。

1.3 Forwarding table 谁填的?

- 在路由器里：

```
[Intra-AS 协议]      → 学到 AS 内部各网段路径
[Inter-AS 协议=BGP] → 学到“外部前缀怎么走、经哪个 gateway”
    ↓
  共同决定 → Forwarding Table（目的前缀 → 下一跳/接口）
```

- 对“AS 内部目的”：只靠 intra-AS；
- 对“外部网络”：先靠 BGP 知道“走哪个 gateway AS / AS path”，再靠内部 OSPF 算“到这个 gateway 的内部路径”。

2. Intra-AS：OSPF / 分层 OSPF 的心智模型

2.1 OSPF = 在 AS 内跑 Dijkstra

- OSPF 特点：
  - *Open*：开放标准；
  - *Link-State*：每个路由器通过 LSA（Link-State Advertisement）泛洪全网；
  - 每个路由器本地：
    - 拥有 AS 内完整拓扑图；
    - 对每个目的跑 Dijkstra → 得到最短路、填 forwarding table；
  - 可以用多种 metric：带宽/时延等；
  - 所有 OSPF 报文直接跑在 IP 之上（不是 TCP/UDP），有认证字段防攻击。
- 心智模型：
  - AS 内每个路由器手里都有同一张“高精地图”，只是在各自位置上往外跑 Dijkstra。

2.2 Hierarchical OSPF：Area + Backbone

- 大型 AS 内部再分层：
  - *Area*：多个局部区域（area 1, area 2…）；

    - *Backbone*: area 0，为所有区域间通信的骨干。
- 路由器角色:
    - *local (internal) router*:
        - 只在一个 area 内;
        - 泛洪 LSA 只在本 area 内进行;
        - 算本 area 内所有目的的最短路。
    - *area border router*:
        - 接在若干 area 和 backbone 之间;
        - 对本 area 里的前缀进行 “汇总” (summarization) 再通告给 backbone。
    - *backbone router*:
        - 只在 backbone，负责跨 area 路由。
    - *boundary router*:
        - 接入外部 AS;
        - 同时参与 OSPF (intra) 和 BGP (inter)。
- 直观理解:
    - 整个 AS 再次被做成 “小 Internet”: 骨干 + 多个区域;
    - 每个 area 内细节自己知道，对其它 area 只需要知道 “往哪个方向走”。

3. Inter-AS: BGP = Internet 的 “粘合剂”

3.1 BGP 做的四件事

- BGP = Border Gateway Protocol:
    - “The glue that holds the Internet together”。
- 给每个 AS 边界路由器提供的能力:
    1. 从邻居 AS 获得 “我能到哪些前缀” 的信息 (eBGP);
    2. 基于这些 reachability + **policy** 选路 (不是单纯最短路);
    3. 将选好的路由通过 iBGP 广泛告诉本 AS 中所有路由器;
    4. 再向外部邻居 AS 广播自己的前缀和路径 (继续传播)。

3.2 eBGP & iBGP

- 两类 BGP 会话:
    - *eBGP* (external BGP):
        - 不同 AS 之间的 gateway 之间的 session;
        - 用来 “跨 AS 交换前缀 reachability 信息”。
    - *iBGP* (internal BGP):
        - 一个 AS 内所有 BGP speaker (主要是 gateway) 之间;
        - 用来在 AS 内传播从外部学到的路径。
- 逻辑视图:

```
AS1----AS2----AS3

AS 边界路由器:
  3a ↔ 2c 之间跑 eBGP
  2c ↔ 2a 等在 AS2 内跑 iBGP
  1c ↔ 2a eBGP, 1c ↔ 1a/1b/1d iBGP
```

3.3 BGP 是 path-vector 协议: path = prefix + attributes

- 一个 BGP “route” = *prefix + attributes*:
    - *prefix*: 一个目的网络，如 X = 138.76.0.0/16;
    - 关键属性:
        1. *AS-PATH*: 此前缀公告从哪个 AS 链接一路传过来:
            - 如:‘AS3, AS15, X‘ 表示路径 AS3→AS15→X;
            - 用来:
                - 检测环路 (看到自己 ASID 就丢掉);
                - 左右 “路径长短” 相关决策。
        2. *NEXT-HOP*:
            - 指向 “通往下一个 AS 的具体路由器” 的 IP;
            - 用于转发时选下一跳。
- BGP 会话跑在 **TCP 连接**上 (长期保持):
    - 报文类型:
        - OPEN: 建会话 + 认证;
        - UPDATE: 通告新路径 / 撤销旧路径;
        - KEEPALIVE: 保持心跳、确认 OPEN;
        - NOTIFICATION: 错误报告及关闭连接。

3.4 BGP 的 “基于策略选路” (policy-based routing)

- 和 OSPF 只管 “cost 最小” 不同，BGP 主要遵循 * 策略 *:
    - “不要走 AS W”;
    - “不要走某个国家 Y”;
    - “优先走自家客户/互惠 AS，不给竞争对手白做转发”。
- 例子: 三家 ISP A/B/C + 下游客户 x/y/w:
    - ISP 不想帮别家运 “第三方流量” (不赚钱的 transit);
    - 某 ISP B 可以:
        - 接受来自 A 的路由 Aw;
        - 但 **不**把 BAw 广告给 C;
        - 从而 C 不会认为可以通过 B 抵达 w;
        - 这样 B 不会成了 C→A→w 的中转通道。

3.5 多条路径时如何选择?

- AS1 的 gateway 1c 可能同时学到:
    - Path1: AS3, X (来自 AS3 直接);
    - Path2: AS2, AS3, X (来自 AS2);
- 1c 会根据策略:
    - 可能偏好更短的 AS-PATH (比如选 AS3,X);
    - 也可能考虑 local preference 等其它属性。
- 一旦选好:
    - 通过 iBGP 通知 AS1 内所有 BGP speaker;
    - 这些路由器再与 OSPF 结合:计算 “从本地到出口 gateway 的内部路径”。

3.6 BGP 填 forwarding table: 与 OSPF 配合

- 思路: * 两段路 *:
    1. BGP 告诉你: “到 X 这个前缀，要经过 1c 这个出口”;
    2. OSPF 告诉你: “到 1c 这台路由器，最佳内部路径是走接口 1 或 2”。
- 在 AS1 内任意一个路由器 (如 1d) 的决策:

```
for dest prefix = X:
  next-hop = "towards 1c"  (由 BGP 决定)
 具体接口 = if_to_1c       (由 OSPF 决定)
```

3.7 Hot Potato Routing (“甩手掌柜”)

- 当 AS 内有多个出口都能到 X 时，BGP 决策 +OSPF 组合有一个常见策略:
    - 选择 **intra-AS 代价最小**的那个出口 (离自己最近的 gateway);
    - 至于外面跨多少 AS、路径是否更长，不太在乎;
    - 这种 “把包尽快丢给别人” 的做法叫 *hot potato routing*。
- 直觉:
    - 自己网内资源宝贵，先甩给别人再说;
    - 不一定是 * 全局最短 AS path*，但符合运营商 “只管自己网” 的思路。

3.8 为什么 intra-AS 和 inter-AS 要完全不同?

- *Policy*:
    - inter-AS: 每个运营商对 “走谁的网、谁走我的网” 都有经济/政治考量;

    - intra-AS：都是自己人，策略简单。
- *Scale*：
    - inter-AS：通过 CIDR 汇总前缀，减少全网路由表规模；
    - intra-AS：拓扑更精细，可以用复杂 metric 做性能优化。
- *Performance*：
    - intra-AS：更看重利用率、时延等性能指标；
    - inter-AS：策略（policy）往往压倒纯性能。

4. SDN Control Plane：从“每台路由器算路”到“控制器算路”

4.1 传统 per-router control plane

- 传统模式：
    - 每台路由器本机：
        - 跑 OSPF/RIP/BGP……；
        - 保存 routing table；
        - 算 forwarding table；
    - 整个网络的控制逻辑“散落在每一个设备里”。
- 缺点：
    - 配置复杂，一台一台改容易出错；
    - 想做“全网级别”的流量工程很麻烦；
    - 升级协议/策略：需要动很多设备。

4.2 SDN 的基本想法：逻辑上集中控制平面

- SDN 架构中：
    - * 数据平面 *：大量“笨但快”的交换机，只负责执行 flow table；
    - * 控制平面 *：逻辑集中在一套 SDN controller（可以是分布式实现）；
    - 交换机 ↔ controller 之间：
        - 用 southbound API（如 OpenFlow）收发 control message；
    - controller ↔ 上层“网络控制应用”之间：
        - 用 northbound API（例如 RESTful API）提供抽象接口。

4.3 SDN 的三层角色

- *Data plane switches*：
    - 提供 generalized forwarding（match→action 的 flow table）；
    - 不再自己跑 OSPF/BGP 等复杂协议；
    - 只通过 southbound 协议（如 OpenFlow）收 controller 下发的表项。
- *SDN controller（网络操作系统）*：
    - 维护全网视图（拓扑、链路状态、主机位置、统计信息等）；
    - 提供抽象 API 给上层应用（northbound）；
    - 通过 southbound（如 OpenFlow）与交换机交互：
        - 收集事件/统计；
        - 下发 flow rules。
- *Network-control applications*：
    - 真正的“脑子”：
        - routing app（算路由）；
        - access control / firewall app；
        - load balancing app；
    - 用 controller 提供的 API 实现各自的控制逻辑。

4.4 SDN controller 内部结构（逻辑分层）

- 从底到顶可以分三层：
    1. *Communication layer*：
        - 与交换机通信（OpenFlow/SNMP 等）；
        - 负责收集流量统计、端口状态、拓扑变化。
    2. *Network-wide state management*：
        - 维护一份分布式、一致的“网络状态数据库”：
            - 拓扑图、链路信息、交换机信息、host 信息；
            - flow table 状态、统计数据。
    3. *Interface / Abstraction layer*：
        - 对上层 app 暴露图抽象、intent 抽象等；
        - 提供 RESTful / 库 API。

4.5 OpenFlow 协议：controller 和 switch 之间的语言

- OpenFlow 负责：
    - 在 **控制器 ↔ 交换机**之间携带：
        - Flow table 操作；
        - packet-in / packet-out；
        - 端口状态事件等。
- 消息类型：
    1. *Controller → Switch*：
        - Features：查询交换机能力；
        - Configure：设置交换机配置；
        - Modify-state：增删改 flow entries；
        - Packet-out：让交换机把某个包从指定端口发出去。
    2. *Switch → Controller*：
        - Packet-in：交换机把某个包“送给控制器处理”；
        - Flow-removed：告知 flow entry 被删（超时、计数达限等）；
        - Port-status：链路 up/down 等状态变化。
    3. *Symmetric*：握手、心跳等杂项。

4.6 一个典型“链路失败 → 重新算路 → 下发表”的流程

- 示例（类似 PPT 图）：

1) 交换机 s1 发现某条链路 down
   → 发送 OpenFlow port status 给 controller

2) controller 更新内部链路状态（state management）

3) routing app（比如 Dijkstra link-state app）
   预先注册了“链路状态变化回调” → 被触发

4) routing app 读取当前网络图/链路信息，运行 Dijkstra，
   得到新的最短路结果

5) routing app 把需要修改的路径交给 flow-table computation 模块，
   生成具体 flow entries

6) controller 通过 OpenFlow 把新的 flow entries 下发到相关交换机
   → 完成全网 rerouting

- 对比传统 per-router OSPF：
    - OSPF 是每台路由器自己泛洪 LSA，自行算路；
    - SDN 是“上头算好路，再广播表项”。

5. ICMP：IP 层的“吐槽与诊断”协议

5.1 ICMP 是什么？

- *Internet Control Message Protocol*：
    - 被 hosts 和 routers 用来报告网络层问题：
        - 目的网络/主机/端口不可达；
        - TTL 过期；
        - IP 头错误；
        - 回显请求/应答（echo request/reply，ping）。
    - 逻辑位置：
        - “运行在 IP 之上”的网络层协议；
        - ICMP 报文被封装在 IP 数据报中，protocol number = 1。

- ICMP 报文格式：
  - type + code + 其他字段 + “引发错误的那个 IP 数据报头及前 8 字节”。

5.2 常见 ICMP 类型（type, code）

- 示例（只记最常用的）：
  - (8, 0)：echo request（ping 请求）；
  - (0, 0)：echo reply（ping 回复）；
  - (3, 0/1/2/3…)：destination unreachable：
    - 3,0：网络不可达；
    - 3,1：主机不可达；
    - 3,3：端口不可达；
  - (11, 0)：TTL expired（常用于 traceroute）；
  - (12, 0)：bad IP header。

5.3 Traceroute 怎么用 ICMP 干活?

- traceroute 的基本操作：
  1. 源主机向目的地发送一系列 UDP 段：
     - 第一批 TTL=1，第二批 TTL=2，以此类推；
     - 目的 UDP 端口号设成一个目的端不会用的（特别大的）端口。
  2. 对第 n 批：
     - 报文在第 n 跳路由器 TTL 减到 0，被丢弃；
     - 第 n 跳路由器发回 ICMP “TTL expired”(type=11, code=0)；
     - 该 ICMP 中的 * 源 IP = 第 n 跳的地址 *；
     - 源主机收到后记下 RTT 和该路由器 IP。
  3. 当报文最终到达目的主机：
     - 目的主机发现 UDP 端口不可达 → 发回 ICMP (3,3) port unreachable；
     - 源主机收到后，知道 “到达终点了”，停止 traceroute。
- 心智模型：
  - 利用 TTL 逐跳 “碰瓷”，用 ICMP 的报错消息把路径上的每一跳 IP 抖出来。

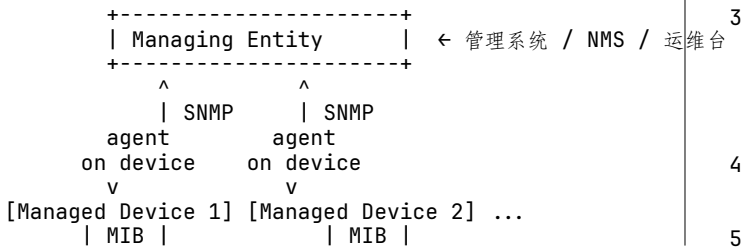
6. Network Management & SNMP：运维视角的网络层

6.1 Network Management 是什么问题?

- 一整个 AS（网络）= 上千个互联的硬件/软件组件：
  - 路由器、交换机、防火墙、服务器、链路、接口……；
  - 都需要监控、配置、告警、统计。
- 1996 年 Saydam 的定义（简化解读）：
  - **部署、集成和协调**硬件/软件/人；
  - **监控、测试、轮询、配置、分析、评估和控制**网络及其元素资源；
  - 目的是在 **合理成本**下满足实时、性能、QoS 需求。

6.2 管理框架结构：managed device + agent + manager + MIB

- 抽象结构：



- 各组件：
  - *Managed device*：
    - 被管理的设备（路由器、交换机等）；
    - 包含大量 *managed objects*（接口速率、CPU 占用等）。
  - *Agent*：
    - 运行在设备上的一个进程；
    - 负责访问本地对象，把数据组织成 MIB，响应 SNMP 请求。
  - *MIB (Management Information Base)*：
    - 层次化组织的变量树；
    - 每个可监控配置的 “点” = 一个 OID。
  - *Managing Entity*：
    - 集中管理系统（NMS），通过 SNMP 协议与各 agent 通信。

6.3 SNMP：Simple Network Management Protocol

- SNMP 提供两种交互模式：
  1. *request/response*（轮询）：
     - 管理端向 agent 发送 GetRequest / GetNext / GetBulk / Set；
     - agent 返回 Response（包含 MIB 数据或设置结果）。
  2. *trap mode*（异步告警）：
     - 当设备上发生异常事件（链路 down、CPU 爆炸等）时；
     - agent 主动向管理端发送 Trap 消息；
     - 不需要管理端先发请求。

6.4 SNMP 消息类型（心里要有张表）

- manager → agent：
  - GetRequest：取某一个 MIB 实例；
  - GetNextRequest：取 “下一个” 变量（遍历）；
  - GetBulkRequest：一口气拿一大块数据；
  - SetRequest：设置某个 MIB 值；
  - InformRequest：某些实现用于 manager↔manager 通信。
- agent → manager：
  - Response：对 Get/Set 的回复；
  - Trap：主动上报告警事件。
- PDU 结构（大概记思路即可）：
  - type（标识是 Get/Set/Trap）；
  - Request ID / Error status / Error index；
  - 若是 Get/Set：后面跟多个 (Name, Value) 对；
  - 若是 Trap：有 enterprise、agent 地址、trap type、time stamp 等。

7. 全讲收束：Network Layer Control Plane 的 “心智总图”

7.1 一张 “从生到死” 的数据包视角

1. 源主机生成 TCP segment → 封成 IP datagram；
2. 在 **数据平面**中：
   - 每一跳路由器用 forwarding table 找下一跳；
   - 转发直至抵达目的。
3. forwarding table 从哪里来?
   - AS 内部：OSPF(link-state)/其他 IGP 决定 intra-AS 路径；
   - AS 之间：BGP 广告前缀 + policy-based 选路；
   - 或由 SDN controller 通过 OpenFlow 直接下发 flow rules。
4. 途中有错、有问题：
   - ICMP 报告 “不可达”、“TTL 过期”、“端口不可达”，辅助 ping/traceroute 等工具；
5. 日常运维：

- SNMP 把各设备状态/MIB 抛给管理系统，或在异常时发 Trap。

7.2 控制平面三种风格对比

维度	Intra-AS (OSPF)	Inter-AS (BGP)	SDN
控制平面位置	每台路由器	每个 AS 边界路由器 + iBGP 全网	逻辑集中于 SDN controller
架构风格	分布式 link-state	分布式 path-vector + policy	集中式 match+action 下发
优先目标	性能（cost, delay）	策略（经济/政治/安全）	易管理、可编程、全网流量工程
与转发表关系	本地算，本地写	本地算，本地写	控制器算，统一下发

7.3 期末复习时的一句话总结

```
Network layer control plane = (OSPF inside AS) + (BGP between ASes)
  -> 在传统模式下，这俩都是“每台路由器自己跑协议”算 forwarding table;
  -> 在 SDN 模式下，算路由的是 controller，路由器/交换机只执行表项。
旁边还有 ICMP 用来抱怨/诊断，SNMP 用来整体运维和监控。
```