网络路由实验

武庆华

wuqinghua@ict.ac.cn

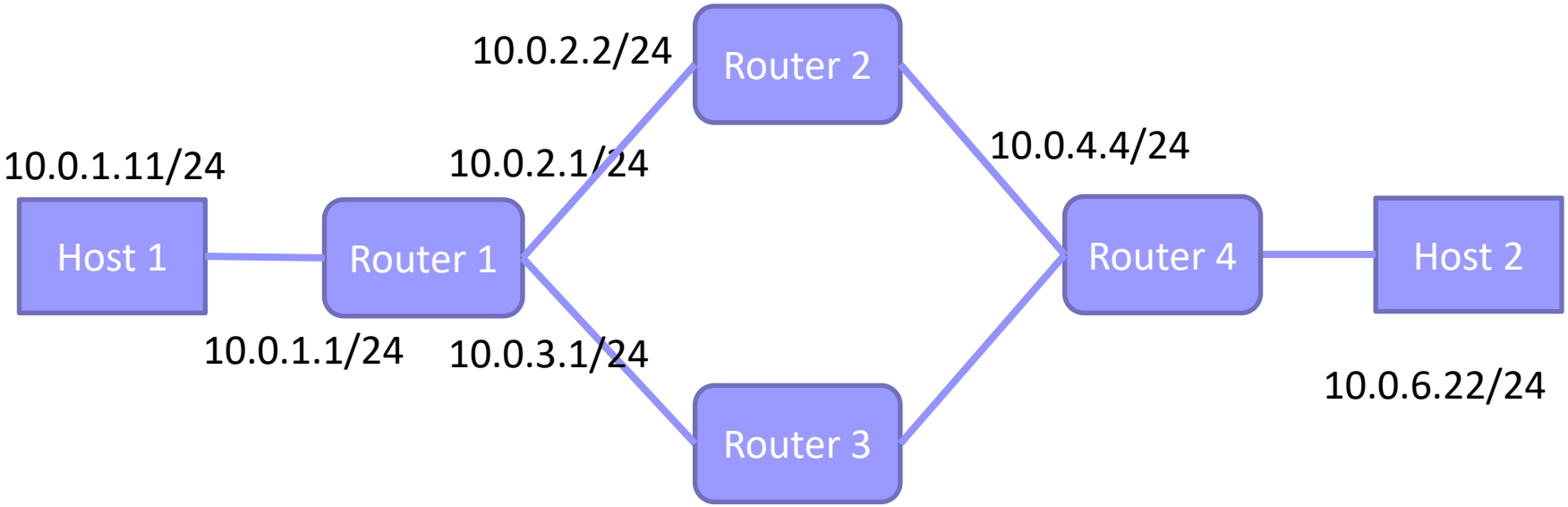
中科院计算所 INSTITUTE OF COMPUTING TECHNOLOGY

中国科学院大学 University of Chinese Academy of Sciences

提纲

- 网络路由
  - 网络转发与网络路由
  - 基于链路状态的路由机制
    - 构建一致性链路状态数据库
      - 邻居发现与管理
      - 链路状态信息洪泛
    - 网络路由计算
      - 最短路径算法
- 实验内容
- 附件文件列表

网络转发与网络路由



Router 1自动生成的转发条目

Network	Gateway	Interface
10.0.1.0/24	0.0.0.0	r1-eth0
10.0.2.0/24	0.0.0.0	r1-eth1
10.0.3.0/24	0.0.0.0	r1-eth2

为了使H1的数据包能够到达H2，还需要如下转发条目

R1: 10.0.6.0/24 -> 10.0.2.2, r1-eth1

R2: 10.0.6.0/24 -> 10.0.4.4, r2-eth1

R4: 10.0.6.0/24 -> 0.0.0.0, r4-eth2

基于链路状态的路由机制

- 链路状态：
  - 路由器端口及其与邻近路由器之间关系的描述

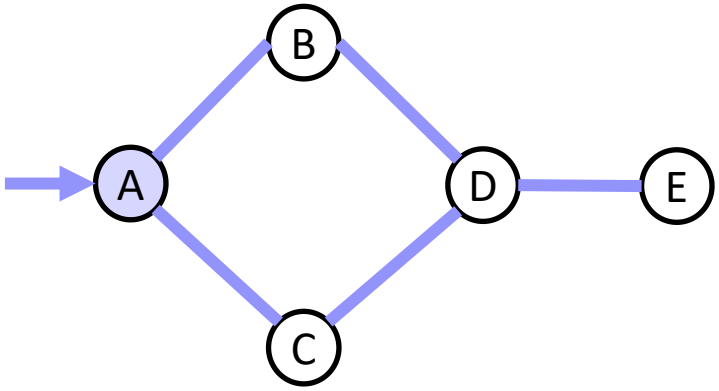
- 基于链路状态的路由机制
  - 每个节点通告自己的链路状态信息，从而构建完整的拓扑信息
    - 通过可靠的洪泛机制，每个节点学习到的拓扑都相同
  - 每个节点单独计算到其它节点的最短路径，生成路由表
    - 使用Dijkstra算法，计算到每个网络的最短路径（下一跳节点）
  - 当网络拓扑发生变动时，重新执行上述两步骤

一致性链路状态数据库

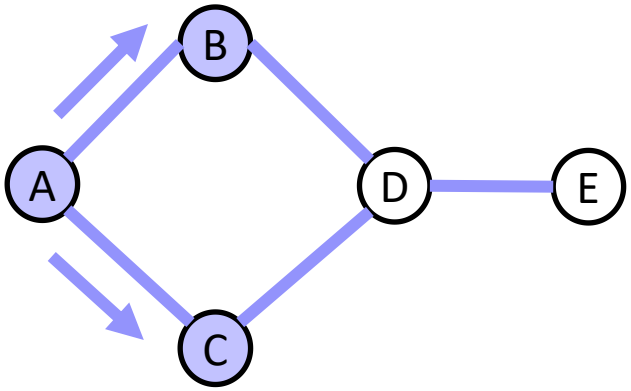
- 每个节点创建链路状态数据包 LSU (Link State Update)
  - 创建该LSU的节点标识（RID, 一般为路由器第1个端口的IP地址）
  - 该节点的相邻节点列表（网络地址和对端节点标识）
  - 序列号，用于区分不同的链路状态更新

- 扩散链路状态
  - 节点B收到来自A的LSU数据包后：
    - 如果之前没有保存对应ID的LSU，则保存
    - 如果之前有保存，新副本的序列号更大，则更新
    - 保存或更新后，向除A以外的所有邻居节点继续扩散

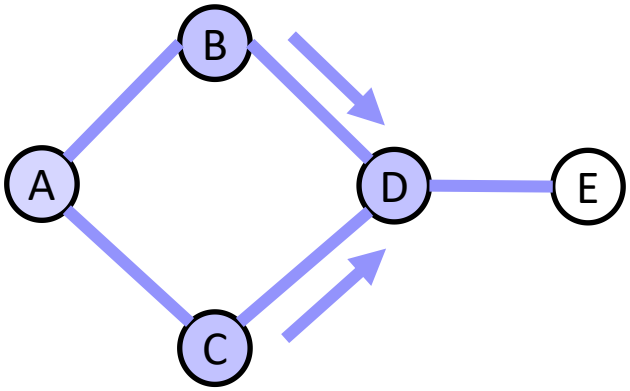
一致性链路状态数据库的例子



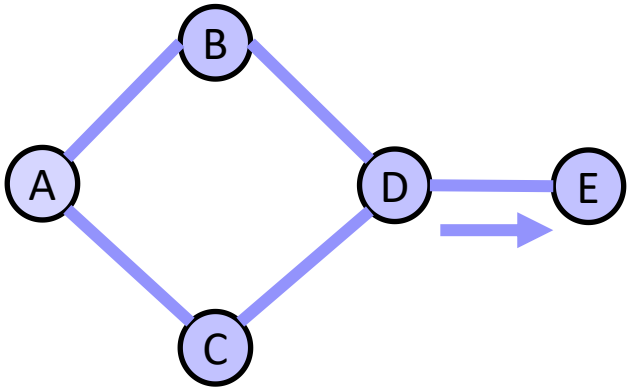
1、某LSU到达A节点



2、A扩散LSU到B和C



3、B和C扩散LSU到D



4、D扩散LSU到E

注意：该图与P3中拓扑无关，只有路由器会参与网络路由

链路状态数据库：邻居发现

- 每个节点周期性（ hello-interval ： 5秒）宣告自己的存在
  - 发送mOSPF Hello消息，包括节点ID, 端口的子网掩码
  - 目的IP地址为224.0.0.5，目的MAC地址为01:00:5E:00:00:05
- 节点收到mOSPF Hello消息后
  - 如果发送该消息的节点不在邻居列表中，添加至邻居列表
  - 如果已存在，更新其达到时间
- 邻居列表老化操作（Timeout）
  - 如果列表中的节点在3*hello-interval时间内未更新，则将其删除

链路状态数据库：链路状态的扩散和更新

- 生成并洪泛链路状态
  - 当节点邻居列表发生变动时，或超过lsu interval (30秒)未发送过链路状态信息时
  - 向每个邻居节点发送链路状态信息
    - 包含该节点ID (mOSPF Header)、邻居节点ID、网络和掩码 (mOSPF LSU)
      - 当端口没有相邻路由器（例如r1-eth0, r4-eth2）时，也要表达该网络，邻居节点ID为0
    - 序列号(sequence number)，每次生成链路状态信息时加1
    - 目的IP地址为邻居节点相应端口的IP地址，目的MAC地址为该端口的MAC地址
- 收到链路状态信息后
  - 如果之前未收到该节点的链路状态信息，或者该信息的序列号更大，则更新链路状态数据库
  - TTL减1，如果TTL值大于0，则向除该端口以外的端口转发该消息
- 处理节点失效问题
  - 当数据库中一个节点的链路状态超过40秒未更新时，表明该节点已失效，将对应条目删除

相关数据结构

```
typedef struct {
    … …

    u32 area_id;        // set to 0.0.0.0
    u32 router_id;      // set to the IP address of 1st interface
    u16 sequence_num;   // sequence number of LSU message
    int lsuint;         // LSU interval, set to 30 seconds
} ustack_t;

extern ustack_t *instance;
```

```
typedef struct {
    … …

    int helloint;               // hello interval, 5 seconds
    int num_nbr;                // number of neighbors
    struct list_head nbr_list; // list of neighbors -> mospf_nbr_t
} iface_info_t;
```

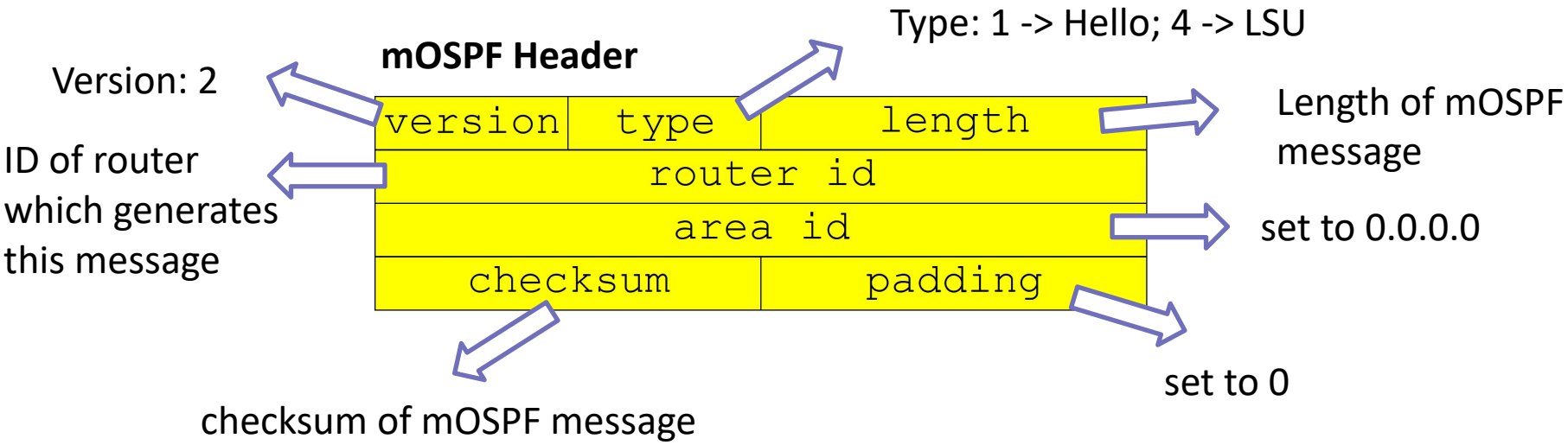
相关数据结构

```
typedef struct {
    struct list_head list;
    u32     nbr_id;          // neighbor ID
    u32     nbr_ip;          // neighbor IP
    u32     nbr_mask;        // neighbor mask
    u8      alive;           // alive for #(seconds)
} mospf_nbr_t;
```

```
typedef struct {
    struct list_head list;
    u32 rid;            // router which sends the LSU message
    u16 seq;            // sequence number of the LSU message
    int nadv;           // number of advertisement
    struct mospf_lsa *array;  // (network, mask, rid)
} mospf_db_entry_t;
```

mOSPF协议格式

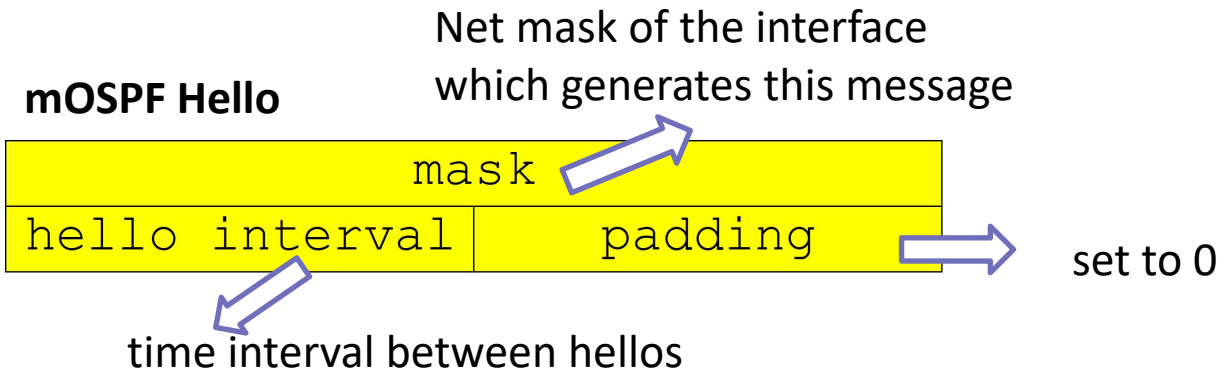
- ip protocol number: 90

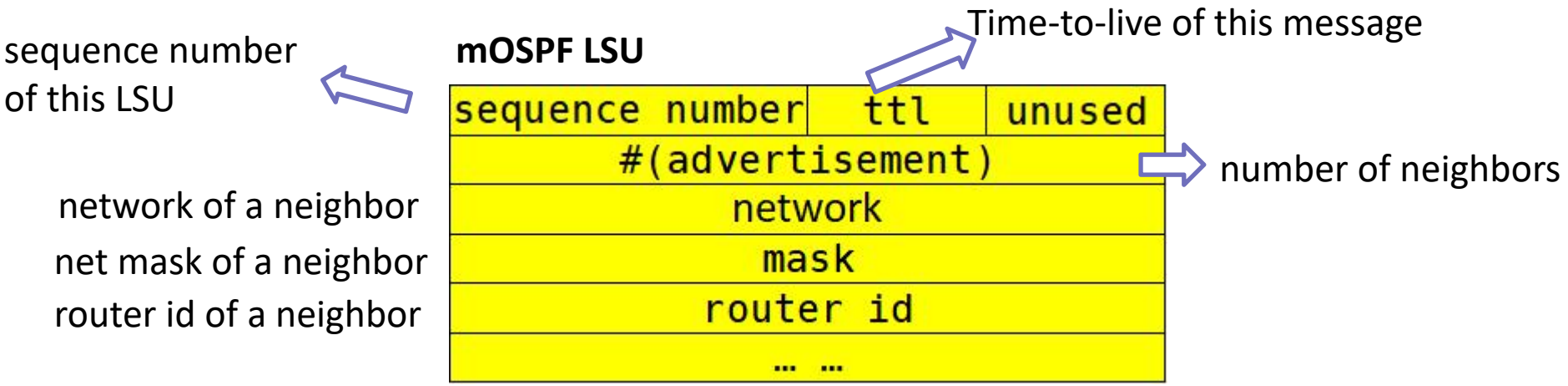


IP (protocol: 90) << mOSPF Header (type: 1) << mOSPF hello

IP (protocol: 90) << mOSPF Header (type: 4) << mOSPF LSU

mOSPF协议格式（续）


Net mask of the interface
which generates this message
mOSPF Hello
mask
hello interval
padding
set to 0
time interval between hellos

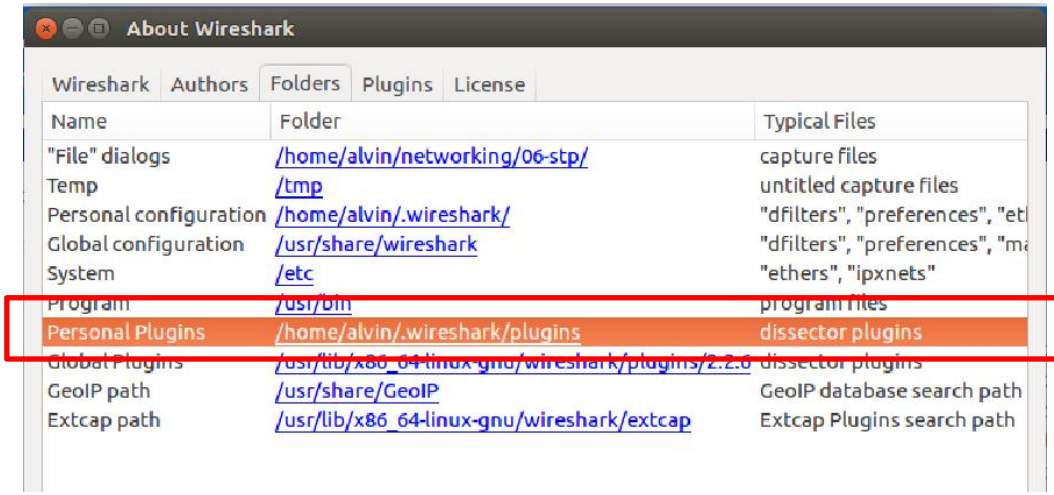

Time-to-live of this message
sequence number
of this LSU
mOSPF LSU
sequence number
ttl
unused
#(advertisement)
number of neighbors
network of a neighbor
network
net mask of a neighbor
mask
router id of a neighbor
router id
… …

mOSPF与OSPFv2的区别

- ip protocol number 不同
  - OSPFv2的protocol number为89
- mOSPF对数据包格式进行了适当简化
- OSPFv2基于可靠洪泛
  - 收到LSU数据包后需要回复ACK
- OSPFv2有更多的消息类型
  - 例如，链路状态数据库Summary
- OSPFv2有安全认证机制（鉴别）

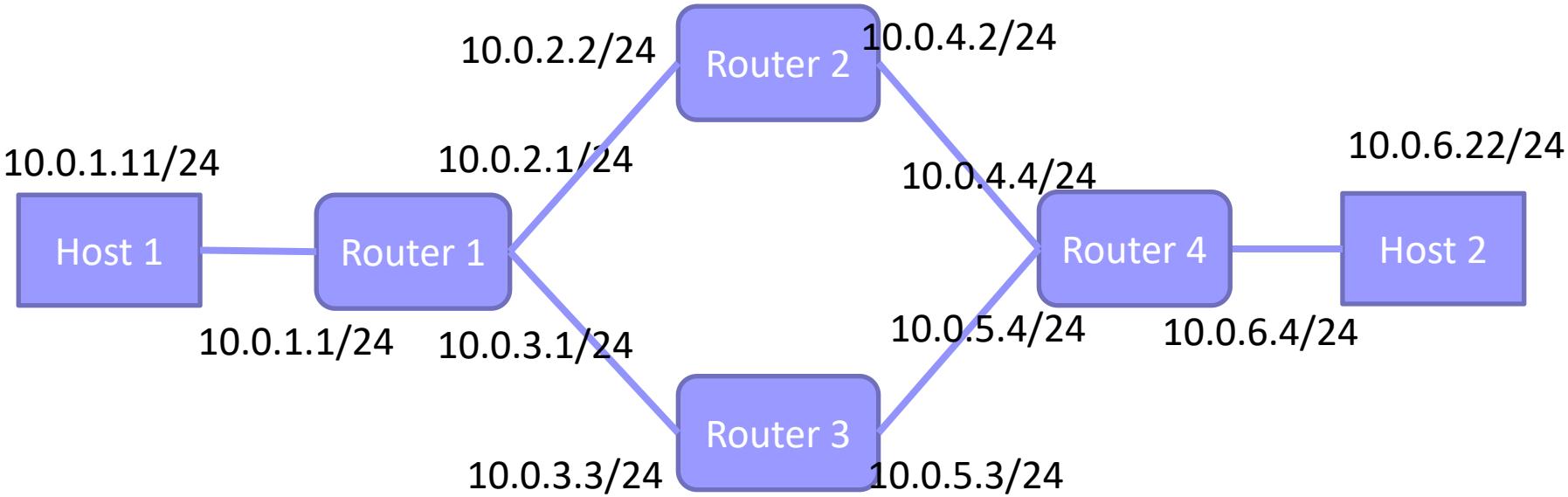
将mOSPF解析脚本加入Wireshark

- 找到wireshark解析插件的存储路径
  - 菜单中打开Help -> About Wireshark



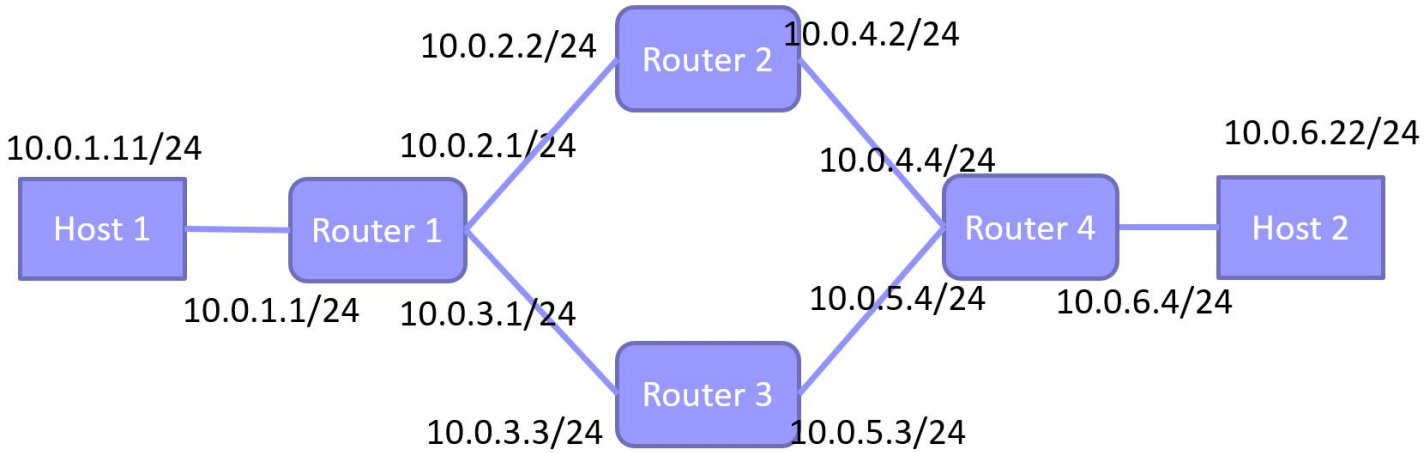
- 将mospf.lua文件放到该目录下

网络路由计算



- 不同节点经过交换链路状态信息，获得一致性链路状态数据库
- 每个节点独立计算路由条目，从而保证网络的可达性

路由条目



- 网络路由条目
  - (destination, mask) -> (gateway, output interface)

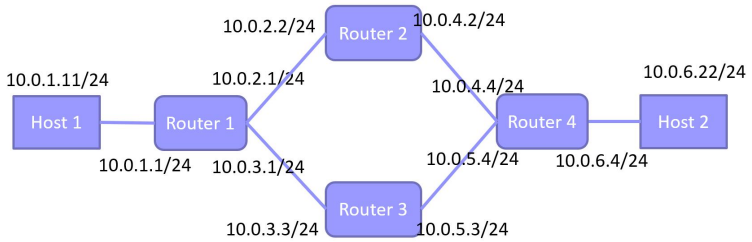
Router 1到本地网络的转发条目

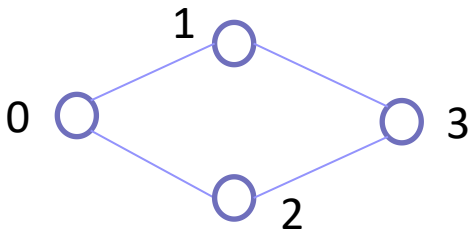
Network	Gateway	Interface
10.0.1.0/24	0.0.0.0	eth0
10.0.2.0/24	0.0.0.0	eth1
10.0.3.0/24	0.0.0.0	eth2

Router 1到其他网络的路由条目

Network	Gateway	Interface
10.0.4.0/24	10.0.2.2	eth1
10.0.5.0/24	10.0.3.3	eth2
10.0.6.0/24	10.0.2.2	eth1

路由计算过程





Prev Node: 0->-1, 1->0, 2->0, 3->1

RT Entry: 10.0.4.0/24 -> (10.0.2.2, eth1)

1. 将链路状态数据库抽象成图拓扑

2. 计算最短路径（前一跳节点）

3. 根据最短路径生成网络路由

计算最短路径

- 使用Dijkstra算法计算源节点到其它节点的最短路径和相应前一跳节点

```
for i in range(num):
    dist[i] = INT_MAX
    visited[i] = false
    prev[i] = -1

dist[0] = 0
```

在未访问的节点中，选取离已访问节点最近的那个

```
for i in range(num):
    u = min_dist(dist, visited, num)
    visited[u] = true

    for v in range(num):
        if visited[v] == false && graph[u][v] > 0 && \
                dist[u] + graph[u][v] < dist[v]:
            dist[v] = dist[u] + graph[u][v]
            prev[v] = u
```

根据最短路径生成路由表

- 路由计算与最短路径算法的不同

	最短路径算法	路由计算
目的	计算到每个节点的路径	计算到每个网络的路由
结果形式	路径长度和前一跳节点	下一跳网关和转发端口

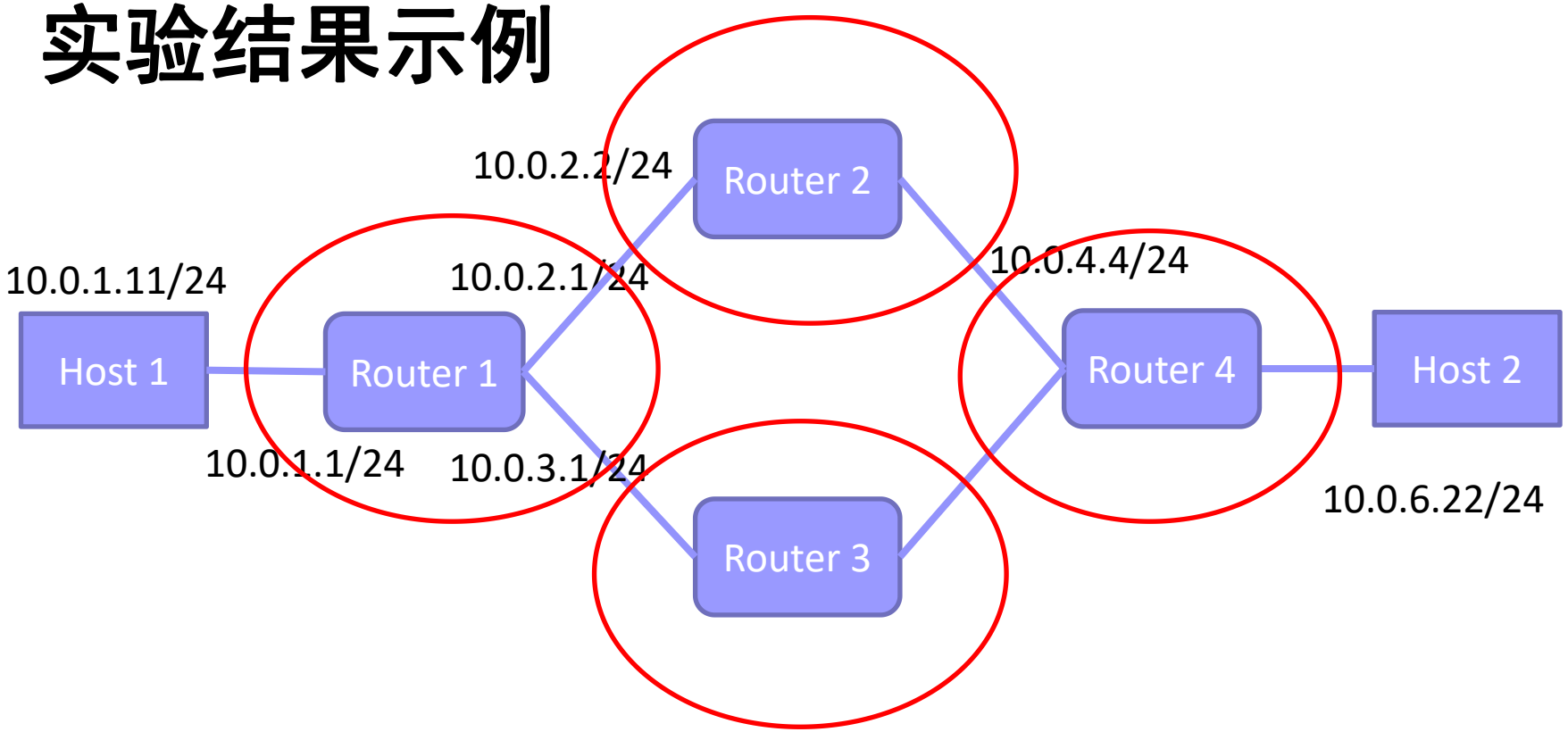
- 由最短路径到路由表项
  - 按照路径长度从小到大依次遍历每个节点
    - 对于节点端口对应的每个网络，如果该网络对应的路由未被计算过
      - 查找从源节点到该节点的下一跳节点
      - 确定下一跳网关地址、源节点的转发端口

实验内容一

- 基于已有代码框架，实现路由器生成和处理mOSPF Hello/LSU消息的相关操作，构建一致性链路状态数据库

- 运行实验
  - 运行网络拓扑(topo.py)
  - 在各个路由器节点上执行disable_arp.sh, disable_icmp.sh, disable_ip_forward.sh)，禁止协议栈的相应功能
  - 运行./mospfd，使得各个节点生成一致的链路状态数据库

实验结果示例



R1 MOSPF Database entries:

10.0.3.3	10.0.3.0	255.255.255.0	10.0.1.1
10.0.3.3	10.0.5.0	255.255.255.0	10.0.4.4
10.0.2.2	10.0.2.0	255.255.255.0	10.0.1.1
10.0.2.2	10.0.4.0	255.255.255.0	10.0.4.4
10.0.4.4	10.0.4.0	255.255.255.0	10.0.2.2
10.0.4.4	10.0.5.0	255.255.255.0	10.0.3.3
10.0.4.4	10.0.6.0	255.255.255.0	0.0.0.0

R2 MOSPF Database entries:

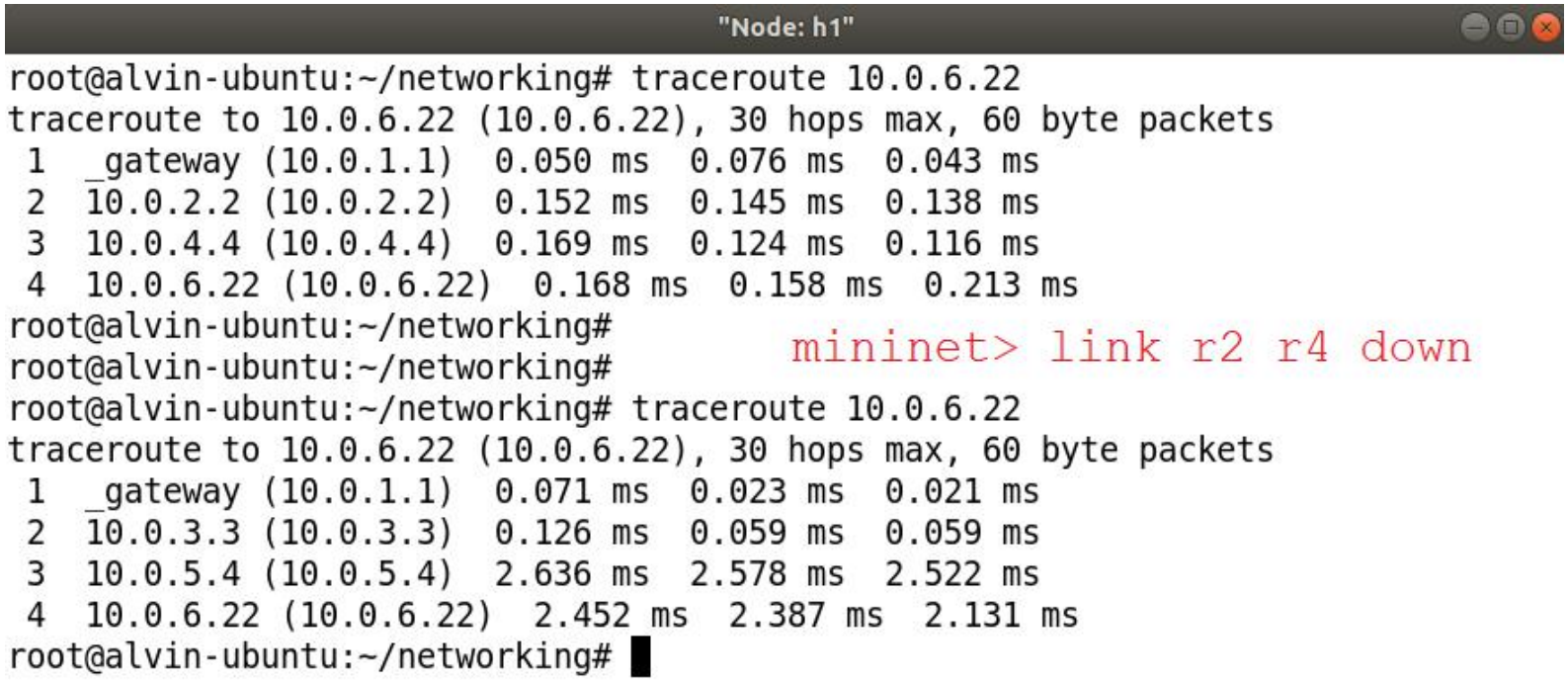
10.0.4.4	10.0.4.0	255.255.255.0	10.0.2.2
10.0.4.4	10.0.5.0	255.255.255.0	10.0.3.3
10.0.4.4	10.0.6.0	255.255.255.0	0.0.0.0
10.0.1.1	10.0.1.0	255.255.255.0	0.0.0.0
10.0.1.1	10.0.2.0	255.255.255.0	10.0.2.2
10.0.1.1	10.0.3.0	255.255.255.0	10.0.3.3
10.0.3.3	10.0.3.0	255.255.255.0	10.0.1.1
10.0.3.3	10.0.5.0	255.255.255.0	10.0.4.4

RID Network Mask Neighbor

实验内容二

- 基于实验一，实现路由器计算路由表项的相关操作
- 运行实验
  - 运行网络拓扑(topo.py)
  - 在各个路由器节点上执行disable_arp.sh, disable_icmp.sh, disable_ip_forward.sh)，禁止协议栈的相应功能
  - 运行./mospfd，使得各个节点生成一致的链路状态数据库
  - 等待一段时间后，每个节点生成完整的路由表项
  - 在节点h1上ping/traceroute节点h2
  - 关掉某节点或链路，等一段时间后，再次用h1去traceroute节点h2

实验结果示例


"Node: h1"
root@alvin-ubuntu:~/networking# traceroute 10.0.6.22
traceroute to 10.0.6.22 (10.0.6.22), 30 hops max, 60 byte packets
 1 _gateway (10.0.1.1) 0.050 ms 0.076 ms 0.043 ms
 2 10.0.2.2 (10.0.2.2) 0.152 ms 0.145 ms 0.138 ms
 3 10.0.4.4 (10.0.4.4) 0.169 ms 0.124 ms 0.116 ms
 4 10.0.6.22 (10.0.6.22) 0.168 ms 0.158 ms 0.213 ms
root@alvin-ubuntu:~/networking#
root@alvin-ubuntu:~/networking#
mininet> link r2 r4 down
root@alvin-ubuntu:~/networking# traceroute 10.0.6.22
traceroute to 10.0.6.22 (10.0.6.22), 30 hops max, 60 byte packets
 1 _gateway (10.0.1.1) 0.071 ms 0.023 ms 0.021 ms
 2 10.0.3.3 (10.0.3.3) 0.126 ms 0.059 ms 0.059 ms
 3 10.0.5.4 (10.0.5.4) 2.636 ms 2.578 ms 2.522 ms
 4 10.0.6.22 (10.0.6.22) 2.452 ms 2.387 ms 2.131 ms
root@alvin-ubuntu:~/networking#

实验注意事项

- 两次traceroute之间尽量间隔在5秒以上，否则会导致traceroute出错

- 实验初始化时，会从内核中读入到本地网络的路由条目
  - 更新路由表时需要注意区分这些条目和计算生成的路由条目

思考题

1. 在构建一致性链路状态数据库中，为什么邻居发现使用组播(Multicast)机制，链路状态扩散用单播(Unicast)机制？

2. 该实验的路由收敛时间大约为20-30秒，网络规模增大时收敛时间会进一步增加，如何改进路由算法的可扩展性？

3. 路由查找的时间尺度为~ns，路由更新的时间尺度为~10s，如何设计路由查找更新数据结构，使得更新对查找的影响尽可能小？

附件文件列表

- include
- ip.c # 处理IP数据包
- libipstack(32).a # 可直接将``路由器转发实验''中自己编译的版本拷贝过来，如果是32位机器，需要将文件名中的32去掉
- main.c
- Makefile
- mospf_daemon.c # 处理Hello、LSU数据包
- mospf_database.c # 链路状态数据库相关函数
- mospf_proto.c # mOSPF协议函数
- mospfd-reference(.32) # 参考实现
- scripts
- topo.py # topo文件
- wireshark # 解析mOSPF协议的wireshark脚本