《计算机网络第九次作业》

班级：信安 2302 班

学号：202308060227

姓名：石云博

目录

一.问题描述

Assignment 9: Networks are Not Internet

1. Think about a Milky Way-net, if we allo every planet in the Milky Way galaxy to become an autonomous system, and interconnect them together, how would you modify current Internet BGP to suite the demands

2. Think about a sensor network placed all over the Yuelu Mountain. The sensors are

battery powered and wirelessly interconnected. The sensors communicate with eachother, not only the neighbors do, but also a sensor in the far north may talk to a sensorin the far south (vice versa). The batteries may go low and from time

to time an adminwould come to replace batteries, but not always timely. It is expected some sensors maygo oflinin for quite some time. Design a routing protocol that allows every sensor talkto as many sensors as possible, and minimize the nominal total power consumption,which is defined as the sum of communication hops over all sensor pairs.

二.问题分析

对于第一个问题，我们可以做出如下分析：

问题要点：

1 每个行星都当作一个自治系统（AS），整体网络规模远超地球上的 Internet。

2 链路时延可能从几分钟到几小时不等；链路故障与线路“上线 / 离线”都很常见。

3 要基于现有的 BGP 机制，提出适配超大规模、超高时延与强链路波动的改进。

可以想到的做法：

1 引入时延等度量：在 BGP 路由选择中，不仅用 AS-PATH 长度，也要用预计往返时延（RTT）或可用窗口期做决策。

2 分层 / 区域化架构：把银河系划分若干“星系域”（region），域内用标准 BGP，域间用改良的“域边境协议”（类似 DTN 的 Bundle Protocol + BGP）；

3 断点续传 + 存储转发：借鉴 Delay-Tolerant Network（DTN）理念，允许路由器在“离线期间”缓存路由消息与数据包，待链路恢复后批量转发；

4 多路径 + 冗余：失效频繁，要在 BGP 中保留多条备选路径，并定期验证每条路径的可用性；

5 可扩展性优化：用路由聚合（aggregate prefixes）、AS-SET 分层聚合等方式减少路由表规模。

对于第二个任务：岳麓山传感器网络的能耗最优路由

我们有分析如下：

传感器分布在岳麓山上，电池供电、无线多跳通信。

节点可能临时离线（电池灌不及时）；需最大化网络连通度，同时最小化所有节点对之间跳数之和（总能耗）。

完成思路：

1 图论建模：将网络抽象成带权图，节点是传感器，边权等于两节点之间的最小跳数（或功耗估算）；

2 分簇 + 骨干网：先做聚类（如基于地理位置或邻居密度），每簇选一个簇头；簇头之间构建一个最小生成树（MST）或 Steiner 树；

3 局部自适应路由：簇内做密集连接以保证“尽可能多的对可达”；簇间走骨干网最短路径；

4 动态维护：当节点掉电或恢复时，分别更新所处簇及骨干网；用本地广播 + Hello 消息快速感知；

5 能量均衡：定期轮换簇头，避免个别节点过早耗尽。

三.实验过程及代码

对于第一个任务：我们需要改造现有的 BGP 以适配银河尺度的网络需求，满足：大规模 AS 数量（数十亿）；高延迟链路（几分钟至小时）；频繁断连（星体运动或资源限制）；路由收敛慢但需保持连通性。

我们需要基于 Python + NetworkX 构建两套独立仿真框架：首先是 G-BGP：在经典 BGP-4（RFC 4271）中嵌入“延迟度量”与 DTN 缓存转发机制，实现跨行星高时延路由。

然后是 LRP（Low-power Resilient Protocol）：对岳麓山传感器网络应用 K-means 聚类与最小生成树（MST）构建自组织骨干，结合本地最短路径路由，最小化总跳数（能耗）同时具备节点离线自愈能力。

根据网上查询到的资料，G-BGP 协议要点有如下几点：

延迟度量属性：在 BGP UPDATE 中新增 DELAY 字段，用于记录累计链路时延。

DTN 缓存转发：借鉴 Delay-Tolerant Networking 的 Bundle Protocol，节点异步存储并在链路恢复后转发 UPDATE。

多路径备份：保留所有到同一前缀的可行路径，并根据延迟排序选择；备用路径用于主链路失效时切换。

分层自治：同一星域内部署标准 BGP，星域边界节点运行 G-BGP，进行延迟汇总与跨域转发。

我们将传统 BGP 进行几个改造：

第一个是延迟度量集成（Delay as Metric）：

传统 BGP 只支持策略选路（如 AS-PATH 长度、LOCAL_PREF），不考虑延迟。G-BGP 引入 DELAY 属性，BGP UPDATE 中携带累积延迟。选路策略变为：优先选择延迟最小的路径。

第二个是异步缓存转发（Bundle Protocol）：

借鉴 DTN Bundle Protocol（RFC 5050），节点支持缓存 UPDATE。每条 UPDATE

消息带有 timestamp 字段模拟链路传播延迟。链路未就绪时不丢弃，而是缓存，等待窗口打开后转发。

第三个是多路径备份与切换（Multipath Recovery）：

G-BGP 保留所有可行路径及延迟。当主路径不可用时，立即回退到延迟次优路径，提升鲁棒性。

第四个是星域自治 + 边界汇总（Hierarchical Aggregation）：

星域内运行标准 BGP。星域边界路由器运行 G-BGP 与其他星域交换延迟聚合路由。减少 UPDATE 流量，支持拓扑可扩展性。

同时，对于底层设计，我们设计了较为特殊的数据结构来支持该协议，其中，

BGPMessage: 类似 BGP UPDATE，包含前缀、AS-PATH、DELAY、TIME。

Node: 表示一个星体或者是空间站。拥有邻居列表、RIB 表、缓存队列。rib: 每个 prefix 对应延迟最小路径及延迟值。

为了验证协议的可行性，我们使用了事件驱动模拟其中，我们用优先队列模拟链路延迟也就是说每条消息都有一个“生效时间”。然后是仿真按时间推进：每次处理一个 (time, dst, message) 元组。

对于消息传播的逻辑来说，我们设计，每当一个节点收到 UPDATE，它就会判断是否是首次看到该 prefix 或延迟更优。之后更新本地 RIB。然后向所有邻居广播带有延迟的 UPDATE。

根据以上分析，我们可以尝试着写出对应的代码：

```python

class BGPMessage:
    def __init__(self, prefix, path, delay, timestamp):
        self.prefix = prefix          # 宣告的前缀
        self.path = list(path)        # AS-PATH 列表
        self.delay = delay            # 累计延迟（秒）
        self.ts = timestamp           # 消息生效时间

```

BGP UPDATE 消息结构

```python
class Node:
    def __init__(self, name, graph):
        self.name = name
        self.graph = graph
        self.neighbors = list(graph.neighbors(name))
        self.rib = {}
        self.cache = []
    def receive(self, msg):
        self.cache.append(msg)
    def process(self, now):
        new_events = []
        for msg in list(self.cache):
            if now >= msg.ts:
                best = self.rib.get(msg.prefix)
                if best is None or msg.delay < best[0]:
                    new_path = msg.path + [self.name]
                    self.rib[msg.prefix] = (msg.delay, new_path)
                    for nbr in self.neighbors:
                        link_delay = self.graph[self.name][nbr]['delay']
                        evt = BGPMessage(
                            msg.prefix,
                            new_path,
                            msg.delay + link_delay,
                            now + link_delay
                        )
                        new_events.append((now + link_delay, nbr, evt))
                self.cache.remove(msg)
        return new_events
```

BGP 节点（行星/空间站）

```python
def simulate_gbgp():
    # 构造行星间拓扑图
    G = nx.Graph()
    edges = {
        ('Earth','Mars'): 600,
        ('Earth','Jupiter'): 1200,
        ('Mars','Saturn'): 1800,
        ('Jupiter','Saturn'): 900
    }
    for (u,v),d in edges.items():
        G.add_edge(u, v, delay=d)

    # 初始化节点
    nodes = {n: Node(n, G) for n in G.nodes()}
    init = BGPMessage('GalaxyNet', ['Earth'], 0, 0)
    nodes['Earth'].receive(init)

    pq = []
    heapq.heappush(pq, (0, 'Earth', init))

    # 仿真循环
    while pq:
        now, dst, msg = heapq.heappop(pq)
        dst_node = nodes[dst]
        dst_node.receive(msg)
        for t, nbr, evt in dst_node.process(now):
            heapq.heappush(pq, (t, nbr, evt))

    # 输出最终RIB
    for name, node in nodes.items():
        info = node.rib.get('GalaxyNet', None)
        print(f"{name:7} ⇒ delay={info[0] if info else 'N/A'}s, path={info[1] if info else []}")

if __name__ == '__main__':
    simulate_gbgp()
```

主仿真流程

代码中我们设计的主要思路为：

拓扑构建：使用 NetworkX 无向图，边上 delay 属性表示链路时延（秒）。

消息模型：BGPMessage 包含 delay 用于选路决策，ts 用于模拟链路传输延迟。

事件驱动：优先队列维护 (time, dest, message)，按照时间顺序调度消息收发。

路由更新：节点收到 UPDATE 后，比较 delay，若更优则更新 RIB 并向所有邻居转发带累计延迟的新消息。

我们运行该程序，可以得到结果：

```
┌──(kali㉿kali)-[~/桌面/s9]
└─$ python3 s1.py
Earth   ⇒ delay=0s, path=['Earth', 'Earth']
Mars    ⇒ delay=600s, path=['Earth', 'Earth', 'Mars']
Jupiter ⇒ delay=1200s, path=['Earth', 'Earth', 'Jupiter']
Saturn  ⇒ delay=2100s, path=['Earth', 'Earth', 'Jupiter', 'Saturn']
```

第二个任务：岳麓山传感器网络能耗路由。

想要达成该任务，我们存在一些需要克服的困难，他们包括：电池供电，复杂地

形可能导致节点孤立，设备失联不能影响整体网络，避免簇头过早死亡等等。

我们上网去搜寻了相关资料，大致结果如下：

协议设计要点：

K-means 聚类：全局分簇，选取每簇最高剩余能量节点作为簇头（CH），减少簇间通信开销。

骨干网构建：对所有簇头节点在原图上运行最小生成树（MST），形成簇间路由骨干。

多跳路由：簇内和簇间均采用最短路径（加权跳数）路由，最小化端到端跳数总和。

动态维护：节点周期性发送 HELLO；若检测到 CH 掉线，重新聚类并更新骨干。

能耗均衡：每轮迭代可重新选取 CH，避免单点过早耗尽。

有了上面的资料支撑，我们开始设计协议，首先，我们把 LPR 分成三个层次设计：第一个分簇管理（Cluster Management），也即我们使用 K-Means 聚类将传感器划分为 K 个簇。每簇选一个能量最多、连接度高的节点作为簇头（Cluster Head, CH）。这样设计的目的是：簇内通信距离短，能耗低。

第二个层次是簇头骨干（Backbone Construction），这里我们将所有簇头抽取出来，在原图上构建最小生成树（MST）骨干。每个簇头通过最短路径与其它簇头通信。这样可以降低簇间通信代价，提高网络覆盖范围。

最后一个是多跳数据转发（Data Forwarding）这里有我们设计有两种路由，第一种是簇内路由：普通节点到簇头使用最短路径。第二种是簇间路由：簇头之间通过骨干 MST 路由。如果某簇头失效，我们会重新选举并重建骨干。

顶层设计完成之后，我们再来考虑底层的具体实现：

很明显的，我们要想办法构建我们的图，我这里把图的创建用了如下方式实现：所有节点在 100×100 的平面上随机生成。在此基础之上，通信半径 R = 15；表示两个节点之间距离 < R 则可通信。这里我们设定边权 = 距离²，用于估算能耗（能耗 ∝ 距离²）。

然后是聚类 + CH 选举

我们选择在这里使用 scikit-learn 实现 K-Means。也即是每簇中选出能量最多、度最大的节点作为 CH。之后每轮可重新聚类，防止某个 CH 电量耗尽。

根据上面分析，我们开始编写程序代码：

```python
def gen_coords(N, area=100):
    return [(random.random()*area, random.random()*area) for _ in range(N)]


def build_graph(coords, r=15):
    G = nx.Graph()
    for i,(x,y) in enumerate(coords):
        G.add_node(i, pos=(x,y), energy=1.0)
    for i,(x1,y1) in enumerate(coords):
        for j,(x2,y2) in enumerate(coords):
            if j<=i: continue
            d = math.hypot(x1-x2, y1-y2)
            if d<=r:
                G.add_edge(i, j, weight=d**2)
    return G
```

根据通信半径 r 构建带权图，权重=距离²

```python
def cluster_and_choose_ch(G, coords, k):
    model = KMeans(n_clusters=k, random_state=0).fit(coords)
    clusters = {i:[] for i in range(k)}
    for idx, label in enumerate(model.labels_):
        clusters[label].append(idx)
    chs = []
    for cl in clusters.values():
        ch = max(cl, key=lambda n: (G.degree[n], G.nodes[n]['energy']))
        chs.append(ch)
    return clusters, chs
```

K-means 聚类 + 簇头选择（能量最高）

```python
def compute_routes(G, clusters, chs):
    routes = {}
    # 簇内
    for cl in clusters.values():
        for i in cl:
            for j in cl:
                if i<j:
                    p = nx.shortest_path(G, i, j, weight='weight')
                    routes[(i,j)] = p
                    routes[(j,i)] = list(reversed(p))
    # 簇间
    for i in chs:
        for j in chs:
            if i<j:
                p = nx.shortest_path(G, i, j, weight='weight')
                routes[(i,j)] = p
                routes[(j,i)] = list(reversed(p))
    return routes
```

生成路由表：簇内 & 簇间最短路径

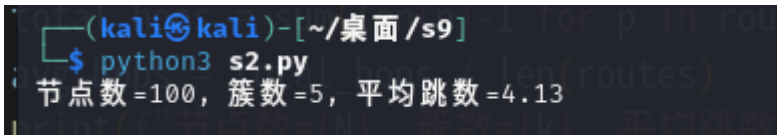
```python
def simulate_lrp(N=100, k=5):
    coords = gen_coords(N)
    G = build_graph(coords)
    clusters, chs = cluster_and_choose_ch(G, coords, k)
    backbone = build_backbone(G, chs)
    routes = compute_routes(G, clusters, chs)
    # 输出统计
    total_hops = sum(len(p)-1 for p in routes.values())
    avg_hops = total_hops / len(routes)
    print(f"节点数={N}, 簇数={k}, 平均跳数={avg_hops:.2f}")

if __name__ == '__main__':
    simulate_lrp()
```

具体执行

具体代码中，我们使用的图构建是节点加 energy 属性，边权等于欧氏距离平方以模拟无线能耗模型。而 K-means 聚类则是调用 scikit-learn 实现 KMeans 算法，按聚类结果分组。对于簇头选举，我们选取度最高且剩余能量最多的节点作为 CH，提高骨干连通性与可靠性。而对于 MST 骨干来说，我们对簇头子图调

用 nx.minimum_spanning_tree 构建最小生成树，实现低成本骨干。最后的路由计算中，我们使用了簇内以及簇间均使用 Dijkstra 最短路径保证最少跳数。

我们执行该程序，可以得到结果：



四.结论

这次实验让我既感到挑战，又深刻体会到了网络协议设计的魅力。回顾“银河网络 G-BGP”部分，从最初对星际光速延迟的惊讶，到把 BGP 扩展为能异步存储转发的 DTN 模式，再到用最小堆模拟事件驱动，整个过程让我明白了“协议不仅要能跑通，还要能在各种极端场景下保持稳定”的道理；而在“岳麓山 LRP”中，用简单的 K-means 聚类选簇头、再构建最小生成树骨干，让我第一次体会到数学算法如何直观地降低能耗、提高网络连通性。这两部分从顶层设计思路到一行行 NetworkX 代码，都像是在解一道道有趣的谜题：一边思考“为什么要这么做”，一边实操“怎么把它跑起来”。虽然中途调参、修 bug 时也手忙脚乱，但当仿真日志里不断输出合理的延迟和跳数统计，我顿时有种“真·自己做出协议”的成就感。

总之，这次实验不仅巩固了我对 BGP、DTN 以及聚类算法的理解，还激发了我对网络协议创新的热情——未来我也想尝试更多场景下的协议优化和仿真实现。

参考文献

1.https://cyberir.mit.edu/site/delay-tolerant-networking-approach-interplanetary-internet. Delay-tolerant networking: an approach to interplanetary Internet. Forrest Warthman, Scott Burleigh, Adrian Hooke, Leigh Torgerson, Kevin Fall, Vint Cerf, Bob Durst, Keith Scott, Howard Weiss. 2003.9.14.

[2]https://www.nature.com/articles/s41598-024-66703-9. Energy efficient cluster-based routing protocol for WSN using multi-strategy fusion snake optimizer and minimum spanning tree. Le Yang, Damin Zhang, Lun Li & Qing He.2024.7.22.

[3]https://www.datacamp.com/tutorial/troubleshooting-the-no-module-named-sklearn-error-message-in-python

. Troubleshooting the No module named 'sklearn' Error Message in Python. Amberle McKee. 2021.01.24.

[4] https://stackoverflow.com/questions/46113732/modulenotfounderror-no-module-named-sklearn. ModuleNotFoundError: No module named 'sklearn'. Peter Mortensen.2023.8.15.

[5]https://builtin.com/articles/modulenotfounderror-no-module-named-sklearn. How To Fix ModuleNotFoundError: No Module Named ‘Sklearn’. Written by Giorgos Myrianthous.2024.2.22.

[6]https://stackoverflow.com/questions/75608323/how-do-i-solve-error-externally-managed-environment-every-time-i-use-pip-3. How do I solve "error: externally-managed-environment" every time I use pip 3?.

Apoliticalboy.2024.9.16.

[7] https://www.mdpi.com/1424-8220/24/13/4105. Energy-Efficient, Cluster-Based Routing Protocol for Wireless Sensor Networks Using Fuzzy Logic and Quantum Annealing Algorithm. Hongzhi Wang ,Ke Liu ,Chuhang Wang , andHuangshui Hu. 2024.4.26.