عنوان:

# شبکه‌های کامپیوتری - پروژه

شماره درس

**۴۰۴۴۳**

تاریخ تحویل

۱۴۰۴/۱۲/۰۴

موضوع

طراحی و تحلیل پروتکل **Gossip**

استاد درس

دکتر امیرمهدی صادق‌زاده

# پروژه

## ۱ عنوان پروژه

طراحی، پیاده‌سازی و تحلیل پروتکل Gossip برای انتشار اطلاعات در شبکه Peer-to-Peer
دقت کنید که این پروژه می‌تواند در قالب تیم‌های یک یا دو نفره انجام شود.

## ۲ مقدمه

در سیستم‌های توزیع‌شده مدرن (مانند بلاکچین‌ها و دیتابیس‌های توزیع‌شده)، یک نیاز پایه‌ای وجود دارد: گره‌ها چگونه یکدیگر را کشف می‌کنند و اطلاعات را به‌صورت مقیاس‌پذیر و قابل اطمینان در شبکه منتشر می‌نمایند؟ پروتکل‌های Gossip یک راه‌حل رایج و قدرتمند برای این مسئله هستند.
پروتکل‌های Gossip (شایعه‌پراکنی) خانواده‌ای از روش‌های انتشار اطلاعات در شبکه‌های بزرگ و غیرمتمرکز هستند که ایده‌ی اصلی آن‌ها از نحوه پخش شدن خبر در اجتماع الهام گرفته شده است. در این روش، هر گره به‌جای ارسال پیام به همه گره‌ها، فقط پیام را برای تعداد محدودی از همسایه‌های خود (معمولا به‌صورت تصادفی) ارسال می‌کند. هر همسایه نیز پس از دریافت پیام، اگر آن را قبلا ندیده باشد، همان کار را تکرار می‌کند و پیام را به چند همسایه دیگر می‌فرستد. به این ترتیب، پیام طی چند دور با هزینه کم و به‌شکل مقیاس‌پذیر در شبکه پخش می‌شود و با احتمال بالا به اکثر گره‌ها می‌رسد.
برای جلوگیری از انتشار بی‌پایان و پردازش تکراری، معمولا دو مکانیزم به‌کار می‌رود: (۱) شناسه یکتا برای هر پیام و نگه‌داری مجموعه پیام‌های دیده‌شده (Seen-Set) و (۲) محدودیت تعداد دورها با استفاده از TTL (کاهش در هر ارسال). در برخی نسخه‌های پیشرفته‌تر، انتشار به‌صورت Push (ارسال فعال) و یا Pull (درخواست پیام‌های جدید از همسایه‌ها) ترکیب می‌شود تا سربار کاهش یابد.

## ۳ خلاصه الزامات کلیدی

- در این پروژه از UDP استفاده کنید.
- پارامترهای قابل تنظیم: fanout، ttl، peer_limit، ping_interval و peer_timeout باید از طریق خط فرمان یا فایل تنظیمات قابل تغییر باشند.
- پرهیز از پردازش تکراری: هر پیام با `msg_id` فقط یک‌بار پردازش شود (با Seen-Set).
- تحلیل آماری: برای هر اندازه شبکه، آزمایش‌ها حداقل ۵ بار با seed متفاوت اجرا شوند و میانگین + انحراف معیار گزارش شود.

## ۴ ساختار کلی پروژه

پروژه از ۵ فاز تشکیل شده است. موفقیت هر فاز، پیش‌نیاز فاز بعدی است.

### ۱.۴ فاز ۱: طراحی پروتکل

هدف این فاز، طراحی دقیق پروتکل قبل از شروع پیاده‌سازی است؛ به‌گونه‌ای که رفتار شبکه، ساختار پیام‌ها و قوانین تصمیم‌گیری هر گره کاملا مشخص و قابل پیاده‌سازی باشد. در این فاز شما باید جزئیاتی مثل اطلاعاتی که هر گره ذخیره می‌کند، قالب و

فیلدهای پیام‌ها، نحوه پیوستن گره جدید به شبکه (Bootstrap)، مدیریت همسایه‌ها و تشخیص گره غیرفعال و منطق انتشار Gossip (مثل بررسی پیام تکراری، انتخاب همسایه‌ها و مقدار TTL/Fanout) را روی کاغذ مشخص کنید تا در فازهای بعدی، پیاده‌سازی و تحلیل بر اساس یک طراحی شفاف و منسجم انجام شود.

### ۱.۱.۴ معماری گره (Node Architecture)

هر گره حداقل باید وضعیت زیر را نگه‌داری کند:

- **NodeID**: یک شناسه یکتا (مثلا UUID) مستقل از IP:Port
- **SelfAddr**: آدرس ip:port
- **Peer List**: لیست همسایه‌ها همراه اطلاعاتی مثل آخرین زمان مشاهده/پاسخ
- **Seen Set**: مجموعه `msg_id` های دیده‌شده برای جلوگیری از پردازش تکراری
- **Config**: پارامترهای قابل تنظیم fanout, ttl, peer_limit, ping_interval, peer_timeout

### ۲.۱.۴ ساختار پیام‌ها (Message Formats)

پروتکل شما حداقل باید از پیام‌های زیر پشتیبانی کند:

- `HELLO` : معرفی گره جدید به یک گره دیگر و آغاز فرایند اتصال (ثبت اولیه در Peer List و هماهنگی‌های اولیه).
- `GET_PEERS` : درخواست لیست همسایه‌های شناخته‌شده یک گره برای گسترش Peer List و کشف بهتر شبکه.
- `PEERS_LIST` : پاسخ به `GET_PEERS` شامل مجموعه‌ای از گره‌های شناخته‌شده (آدرس/شناسه) برای کمک به Bootstrap و به‌روزرسانی همسایه‌ها.
- `GOSSIP` : انتشار یک اطلاعات/رویداد جدید در شبکه. گره‌های دریافت‌کننده در صورت جدید بودن، آن را ذخیره و به تعدادی از همسایه‌ها فوروارد می‌کنند.

همچنین برای مدیریت همسایه‌ها، پیام‌های `PING/PONG` توصیه می‌شوند. برای تشخیص زنده بودن همسایه‌ها و به‌روز نگه‌داشتن Peer List، گره‌ها به‌صورت دوره‌ای پیام `PING` ارسال می‌کنند و در صورت زنده بودن، طرف مقابل با `PONG` پاسخ می‌دهد. اگر یک گره در بازه‌ی زمانی مشخص (Timeout) به چند `PING` پاسخ ندهد، می‌توان آن را غیرفعال در نظر گرفت و از List Peer حذف کرد.
شما می‌توانید پیام‌های بیشتری را در صورت نیاز برای پروتکل خود گسترش دهید.
فرمت کلی پیام (JSON روی UDP).

```
{
  "version": 1,
  "msg_id": "uuid-or-hash",
  "msg_type": "HELLO | GET_PEERS | PEERS_LIST | GOSSIP | PING | PONG",
  "sender_id": "node-uuid",
  "sender_addr": "127.0.0.1:8000",
  "timestamp_ms": 1730,
  "ttl": 8,
  "payload": { ... }
}
```

نمونه Payloadها. دقت کنید که می‌توانید موارد زیر و یا هدر را بر اساس نیازهای خود تغییر دهید. اما باید تغییرات خود را در گزارش توضیح دهید.

- HELLO :

```
"payload": {
  "capabilities": ["udp", "json"]
}
```

- GET_PEERS :

```
"payload": { "max_peers": 20 }
```

- PEERS_LIST :

```
"payload": {
  "peers": [
    {"node_id":"...", "addr":"127.0.0.1:8001"},
    {"node_id":"...", "addr":"127.0.0.1:8002"}
  ]
}
```

- GOSSIP :

```
"payload": {
  "topic": "news",
  "data": "Hello network!",
  "origin_id": "node-uuid",
  "origin_timestamp_ms": 1730
}
```

- PING

```
"payload": {
  "ping_id": "uuid-or-counter",
  "seq": 17
}
```

- PONG

```
"payload": {
  "ping_id": "uuid-or-counter",
  "seq": 17
}
```

### ۳.۱.۴ منطق پروتکل (**Protocol Logic**)

(آ) Bootstrap (پیوستن گره جدید)
در ابتدا نیاز است تا گره بذری را تعریف کنیم: گره بذری یک یا چند گره همیشه در دسترس در شبکه است که یک گره جدید هنگام شروع کار، آدرس آن را از قبل می‌داند تا فرایند ورود به شبکه را آغاز کند. گره تازه‌وارد با ارسال پیام‌هایی مانند HELLO و GET_PEERS به گره بذری، اولین لیست همسایه‌های خود را دریافت می‌کند و سپس می‌تواند سایر گره‌ها را کشف کرده و مستقل از گره بذری به کار ادامه دهد. گره بذری نقش مرکزی دائمی ندارد و صرفا نقطه شروع (Entry Point) برای بوت‌استرپ شبکه محسوب می‌شود.
فرایند پیشنهادی:

۱. ارسال HELLO به گره بذری
۲. ارسال GET_PEERS و دریافت PEERS_LIST
۳. تکمیل List Peer تا سقف peer_limit
۴. شروع چرخه‌های دوره‌ای برای PING/PONG و تازه‌سازی همسایه‌ها

(ب) مدیریت همسایه‌ها و تشخیص گره مرده

- هر ping_interval ثانیه به تعدادی از همسایه‌ها PING ارسال کنید.
- اگر تا peer_timeout پاسخی نیامد، همسایه مشکوک و پس از چند بار عدم پاسخ حذف شود.
- اندازه Peer List محدود باشد و سیاست جایگزینی مشخص داشته باشید (مثلا حذف قدیمی‌ترین یا کم‌فعال‌ترین).

(ج) منطق انتشار Gossip وقتی گره GOSSIP را دریافت می‌کند:

۱. اگر msg_id در Set Seen بود، نادیده بگیریم.
۲. در غیر این صورت ذخیره در Set Seen + ثبت زمان دریافت در لاگ.
۳. TTL را کاهش داده و اگر ttl > ۰ بود پیام را به fanout همسایه تصادفی و بدون تکرار ارسال کند. (fanout تعداد همسایه‌هایی است که هر گره پیام را برای آن‌ها ارسال می‌کند)

## ۲.۴ فاز ۲: پیاده‌سازی

### ۱.۲.۴ الزامات پیاده‌سازی

۱. برنامه اجرایی node بسازید که از CLI اجرا شود و پارامترها را بگیرد:

```
./node --port 8000 --bootstrap 127.0.0.1:9000 \
      --fanout 3 --ttl 8 --peer-limit 20 \
      --ping-interval 2 --peer-timeout 6 \
      --seed 42
```

توضیح پارامترهای port ، bootstrap و seed :

- port– : پورتی است که گره روی آن روی همان ماشین به پیام‌های ورودی گوش می‌دهد. در شبیه‌سازی روی localhost هر گره باید پورت متفاوت داشته باشد تا تداخل ایجاد نشود (مثلاً ۸۰۰۰، ۸۰۰۱، ۸۰۰۲، ...).
- –bootstrap <ip:port> : آدرس گره بذری به فرم IP:Port است. گره جدید در شروع، از طریق این آدرس به شبکه معرفی می‌شود و با پیام‌هایی مثل HELLO و GET_PEERS اولین لیست همسایه‌های خود را دریافت می‌کند. در شبیه‌سازی محلی معمولاً 127.0.0.1:<port> استفاده می‌شود.
- seed– : مقدار اولیه برای تولید اعداد شبه‌تصادفی است (برای مثال در انتخاب تصادفی همسایه‌ها هنگام فوروارد کردن پیام یا انتخاب همسایه برای PING ). داشتن seed مشخص باعث می‌شود اجرای آزمایش‌ها قابل تکرار باشد و بتوان نتایج را منصفانه مقایسه کرد. در فاز تحلیل، برای هر $N$ باید چندین بار آزمایش با seedهای متفاوت اجرا شود.

۲. هر گره باید به‌صورت همزمان این کارها را انجام دهد (Thread/Async/Event loop همگی قابل قبول‌اند):

- گوش دادن به پیام‌های ورودی
- ارسال پیام‌های دوره‌ای (PING و مدیریت Peer List)
- دریافت ورودی کاربر برای تولید یک GOSSIP جدید

دقت کنید که تمامی عملکردهای یک گره باید به صورت خودکار انجام شود. کاربر باید بتواند در ترمینال مربوط به هر گره پیامی را بنویسد و آن گره باید فرایند Gossip را آغاز کند.

۳. لاگ‌های مهم باید در خروجی چاپ شوند (اتصال، دریافت پیام جدید، فوروارد به همسایه‌ها، حذف همسایه مرده و ...).

#### ۲.۲.۴ الزامات صحت (**Correctness**)

- هر msg_id حداکثر یک بار پردازش شود.
- پیام خراب/JSON نامعتبر باعث کرش نشود.
- TTL از انتشار بی‌نهایت جلوگیری کند.
- Peer List بی‌رویه رشد نکند (سقف و سیاست مدیریت داشته باشد).

در انتها، شبکه‌ای با ۱۰ گره روی localhost اجرا کنید و نشان دهید با تولید یک پیام، حداقل ۹ گره آن را دریافت کرده‌اند.

### ۳.۴ فاز ۳: شبیه‌سازی و تحلیل عملکرد

#### ۱.۳.۴ سناریوی شبیه‌سازی

یک اسکریپت بنویسید که شبکه‌هایی با اندازه‌های زیر را به‌صورت خودکار اجرا کند:

$$N \in \{10,\ 20,\ 50\}$$

برای هر $N$، آزمایش را حداقل ۵ بار با seed متفاوت اجرا کنید.

#### ۲.۳.۴ جمع‌آوری داده‌ها

هر گره باید زمان دریافت هر GOSSIP msg_id را در لاگ ثبت کند. اسکریپت تحلیل باید از روی لاگ‌ها معیارها را محاسبه کند.

#### ۳.۳.۴ معیارهای ارزیابی

۱) زمان همگرایی (Convergence Time)
اگر پیام در زمان $t_0$ تولید شود، زمان همگرایی کمترین زمانی است که تا آن لحظه ۹۵% گره‌ها پیام را دریافت کرده باشند.
۲) سربار پیام (Message Overhead)
مجموع تعداد تمام پیام‌های ارسال‌شده (شامل GOSSIP و پیام‌های کنترلی مانند HELLO/GET_PEERS/PEERS_LIST/PING/PONG) در کل شبکه، از لحظه تولید پیام تا رسیدن به ۹۵% پوشش.

#### ۴.۳.۴ ارائه نتایج

نتایج را با نمودارهای مقایسه‌ای ارائه کنید:

- محور افقی: $N$
- محور عمودی: زمان همگرایی و سربار پیام

همچنین اثر تنظیمات `fanout` و `ttl` و سیاست همسایه‌ها را تفسیر کنید و نتایج را به ازای پارامترهای مختلف تحلیل کنید.

### ۴.۴ فاز ۴: بخش‌های تکمیلی (الزامی)

در این پروژه علاوه بر Gossip ساده (Push-based)، دو قابلیت تکمیلی زیر نیز الزامی هستند. هدف از این بخش‌ها، افزایش واقع‌گرایی پروژه و مقایسه‌ی عملی بین طراحی‌های مختلف از نظر سربار پیام و سرعت همگرایی است.

#### ۱.۴.۴ بخش تکمیلی ۱: **Hybrid Push-Pull**

در روش Push ساده، هر گره پس از دریافت پیام جدید آن را برای تعدادی از همسایه‌ها ارسال می‌کند. این روش معمولا سریع است اما می‌تواند سربار پیام را افزایش دهد (به‌خصوص وقتی اکثر گره‌ها قبلا پیام را دیده‌اند). در روش Hybrid (ترکیب Push و Pull)، علاوه بر Push، یک مکانیزم سبک برای درخواست پیام‌های ندیده اضافه می‌شود تا از ارسال تکراری و بی‌فایده جلوگیری گردد.

پارامترهای جدید (الزامی و قابل تنظیم).

- `pull_interval` : فاصله زمانی ارسال پیام‌های Pull/Hybrid (مثلا هر ۲ ثانیه)
- `ihave_max_ids` : حداکثر تعداد شناسه‌هایی که در یک پیام `IHAVE` قرار می‌گیرد (مثلا ۳۲)

هر گره به‌صورت دوره‌ای به همسایه‌ها اعلام می‌کند چه پیام‌هایی را دارد (فقط `msg_id` ها، نه محتوای کامل). همسایه اگر پیام را نداشته باشد، آن را درخواست می‌کند و پیام کامل فقط در صورت نیاز ارسال می‌شود.

پیام‌های **Hybrid**:

- `IHAVE` : اعلان خلاصه‌ی پیام‌های جدید (لیست کوتاهی از `msg_id` )
- `IWANT` : درخواست پیام‌های مشخص (لیست `msg_id` های موردنیاز)

نمونه Payload برای `IHAVE` :

```
"payload": {
  "ids": ["id1", "id2", "id3"],
  "max_ids": 32
}
```

نمونه Payload برای `IWANT` :

```
"payload": {
  "ids": ["id2", "id3"]
}
```

قوانین اجرای Hybrid

- هر گره هر `pull_interval` ثانیه (قابل تنظیم) به تعدادی از همسایه‌ها `IHAVE` می‌فرستد.
- `IHAVE` فقط شامل شناسه پیام‌ها باشد و طول آن با `ihave_max_ids` محدود شود تا پیام کنترلی بزرگ نشود.
- گره دریافت‌کننده، لیست `ids` را با Set Seen مقایسه می‌کند. هر `msg_id` که موجود نبود، در `IWANT` درخواست می‌شود.
- گره ارسال‌کننده در پاسخ به `IWANT` ، پیام کامل `GOSSIP` مربوط به هر `msg_id` را ارسال می‌کند.

در نهایت در فاز ۳ باید دو حالت را اجرا و مقایسه کنید:

- Push-only: فقط انتشار معمولی `GOSSIP` با Fanout
- Hybrid Push-Pull: Push + پیام‌های `IHAVE/IWANT`

و اثر Hybrid را بر Overhead و Convergence گزارش کنید (میانگین و انحراف معیار).

### ۲.۴.۴ بخش تکمیلی ۲: مقاومت در برابر **Sybil** با **Proof-of-Work**

در شبکه‌های P2P یک مهاجم می‌تواند با ساخت تعداد زیادی گره جعلی (Sybil)، Peer List دیگران را آلوده کند و انتشار پیام را مختل نماید. در این پروژه یک مکانیزم دفاعی سبک بر پایه‌ی Proof-of-Work (PoW) برای پذیرش `HELLO` اضافه می‌شود تا پیوستن به شبکه هزینه‌بر شود.

- `pow_k` : تعداد صفرهای ابتدایی موردنیاز در هش (Difficulty). نمونه‌ی پیشنهادی: $\texttt{pow\_k} = 4$ (قابل تنظیم)

تعریف PoW: هر گره جدید باید عددی به نام nonce پیدا کند به‌گونه‌ای که:

$$H(\texttt{node\_id} \,\|\, \texttt{nonce}) \text{ با } \texttt{k} \text{ صفر ابتدایی شروع شود}$$

که در آن $H$ یک تابع هش (مثلاً SHA-256) است و $k$ همان `pow_k` است.

نمونه Payload برای `HELLO` همراه PoW:

```
"payload": {
  "capabilities": ["udp", "json"],
  "pow": {
    "hash_alg": "sha256",
    "difficulty_k": 4,
    "nonce": 9138472,
    "digest_hex": "0000ab12..."
  }
}
```

قوانین پذیرش همسایه

- هر گره هنگام دریافت HELLO باید PoW را اعتبارسنجی کند: (۱) محاسبه کند $H(node\_id||nonce)$ واقعا با $k$ صفر شروع می‌شود، (۲) مقدار difficulty_k با pow_k تنظیم‌شده برابر باشد.
- اگر PoW معتبر نبود، درخواست باید رد شود و فرستنده به Peer List اضافه نشود.
- مقدار $k$ باید طوری انتخاب شود که روی لپ‌تاپ معمولی چندصدم ثانیه تا چند ثانیه زمان ببرد (نه آن‌قدر سخت که غیرعملی شود).

در نهایت در گزارش نهایی، تاثیر مکانیزم PoW را به‌صورت شفاف تحلیل کنید و به موارد زیر پاسخ دهید:

- مزیت امنیتی: توضیح دهید PoW چگونه پیوستن انبوه گره‌های جعلی (Sybil) را هزینه‌بر می‌کند و احتمال آلوده شدن Peer List را کاهش می‌دهد (حتی اگر به‌طور کامل حمله را حذف نکند).
- هزینه و اثر جانبی: زمان متوسط لازم برای پیدا کردن nonce را برای مقدار انتخابی pow_k گزارش کنید (مثلا میانگین و بازه‌ی تقریبی روی سیستم خودتان) و توضیح دهید این هزینه چه اثری روی سرعت پیوستن گره‌های سالم به شبکه دارد.
- انتخاب سختی: مقدار pow_k انتخاب‌شده را ذکر کنید و استدلال کنید چرا این مقدار متعادل است (نه آن‌قدر آسان که بی‌اثر شود، نه آن‌قدر سخت که برای استفاده عادی غیرعملی گردد).
- یک جدول کوچک یا نمودار ساده از زمان تولید PoW برحسب pow_k (برای ۲ یا ۳ مقدار مختلف) ارائه کنید تا روند هزینه به‌طور ملموس مشخص شود.

### ۵.۴ فاز ۵: مستندسازی و ارائه

۱. کد منبع

۲. گزارش PDF شامل طراحی، نکات پیاده‌سازی، نمودارها و تحلیل

۳. ویدئوی کوتاه (حداکثر ۱۰ دقیقه):

- اجرای شبکه (مثلاً ۱۰ گره) و انتشار یک پیام
- توضیح تصمیمات مهم طراحی
- جمع‌بندی نتایج تحلیل

## ۵ نحوه تحویل

تمام فایل‌ها (کد، گزارش و لینک ویدیو) را در یک فایل ZIP تحویل دهید.

یادداشت مهم: استفاده از کتابخانه‌های آماده که خود پروتکل Gossip را پیاده‌سازی کرده‌اند مجاز نیست اما استفاده از کتابخانه‌های استاندارد شبکه، JSON و ... مجاز است. همچنین برای پیاده‌سازی می‌توانید از همه زبان‌های برنامه‌نویسی استفاده کنید.