بسمه تعالی

دانشکده مهندسی کامپیوتر

درس: شبکه‌های کامپیوتری
نیم‌سال اول ۱۴۰۵-۱۴۰۴

گزارش پروژه درس شبکه‌های کامپیوتری

طراحی، پیاده‌سازی و تحلیل پروتکل Gossip روی UDP

اعضای گروه	
ابوالفضل شیخها	۴۰۱۱۰۶۱۴۷
امیرمحمد گمار	۴۰۱۱۰۶۴۲۵

۷ اسفند ۱۴۰۴

فهرست مطالب

۱ چکیده

در این پروژه ما یک پروتکل Gossip مبتنی بر UDP را برای انتشار پیام در شبکه همتا به همتا طراحی و پیاده‌سازی کردیم. تمرکز ما در طراحی روی سه هدف اصلی بود: انتشار سریع پیام، کنترل سربار، و پایداری شبکه در برابر خرابی یا قطع همسایه‌ها. برای رسیدن به این اهداف، در سطح پروتکل از Seen Set برای جلوگیری از پردازش تکراری، از ترکیب TTL و Fanout برای کنترل دامنه انتشار، و از مکانیزم‌های HELLO، GET_PEERS، PEERS_LIST، PING و PONG برای عضویت و نگه‌داری همسایه‌ها استفاده کردیم. علاوه بر حالت پایه Push، نسخه Hybrid Push-Pull با پیام‌های IHAVE و IWANT را نیز اضافه کردیم تا همگرایی و سربار در سناریوهای مختلف قابل مقایسه باشد. همچنین برای مقاوم‌سازی اولیه در برابر Sybil، اعتبارسنجی PoW در مرحله پذیرش HELLO پیاده‌سازی شد. در ادامه گزارش، ابتدا فاز طراحی پروتکل (فاز ۱) را به‌صورت دقیق ارائه می‌کنیم، سپس پیاده‌سازی، اسکریپت‌های آزمایش، و نتایج تحلیلی را گزارش می‌دهیم.

۲ فاز ۱: طراحی پروتکل

۱.۲ هدف طراحی و تصمیم‌های کلیدی

در فاز ۱ باید قبل از ورود به پیاده‌سازی، رفتار پروتکل را کامل و قابل‌اجرا مشخص می‌کردیم. ما طراحی را طوری انجام دادیم که هم با الزامات رسمی پروژه هم‌خوان باشد و هم با پیاده‌سازی واقعی گره در کد یک‌به‌یک منطبق بماند. تصمیم‌های کلیدی ما در این فاز عبارت‌اند از:

- استفاده از UDP + JSON با یک هدر مشترک برای همه پیام‌ها.
- جداسازی پیام‌های عضویت، زنده‌مانی و انتشار (HELLO، GET_PEERS، PEERS_LIST، PING، PONG و GOSSIP).
- با استفاده از seen_msgs، هر msg_id فقط یک‌بار پردازش می‌شود.
- کنترل انتشار با fanout و ttl برای جلوگیری از طوفان پیام.
- محدود نگه‌داشتن لیست همسایه‌ها با peer_limit و سیاست جایگزینی مشخص.

۲.۲ ۱.۱.۴ معماری گره — Node Architecture

هر گره مطابق طراحی ما این وضعیت داخلی را نگه می‌دارد:

- NodeID: شناسه یکتا با UUID مستقل از IP:Port.
- SelfAddr: آدرس شنود گره روی 127.0.0.1:port.
- Peer List: شامل node_id, addr, last_seen, last_ping, missed_pongs.
- Seen Set: مجموعه msg_idهای دیده‌شده برای جلوگیری از پردازش تکراری.
- Gossip Cache: کش پیام کامل برای پاسخ به IWANT در حالت هیبرید.
- Config: پارامترهای fanout, ttl, peer_limit, ping_interval, peer_timeout و پارامترهای تکمیلی (pull_interval, ihave_max_ids, pow_k).

۳.۲ ۲.۱.۴ قالب پیام‌ها — Message Formats

همه پیام‌ها از یک قالب سربرگ مشترک استفاده می‌کنند:

```
{
  "version": 1,
  "msg_id": "uuid-or-hash",
  "msg_type": "...",
  "sender_id": "node-uuid",
  "sender_addr": "127.0.0.1:8000",
  "timestamp_ms": 1730,
  "ttl": 8,
  "payload": { ... }
}
```

پیام‌های اصلی طراحی ما:

- HELLO: معرفی گره جدید؛ در صورت فعال بودن PoW، شامل payload.pow.
- GET_PEERS: درخواست همسایه‌ها با max_peers.
- PEERS_LIST: پاسخ همسایه‌ها با لیست (node_id, addr).
- PING/PONG: پایش زنده‌مانی همسایه‌ها.
- GOSSIP: انتشار داده کاربردی با topic/data/origin_id/origin_timestamp_ms.

در payload پیام‌های پایه (GET_PEERS, PEERS_LIST, GOSSIP, PING, PONG) تغییر ساختاری نداده‌ایم و همان الگوی استاندارد را پیاده‌سازی کرده‌ایم. تنها افزونه payload در کد ما، اضافه شدن فیلد payload.pow در پیام HELLO هنگام فعال بودن PoW است.

۴.۲ ۳.۱.۴ منطق پروتکل — Protocol Logic

۱.۴.۲ آ) Bootstrap (پیوستن گره جدید)

جریان ورود گره جدید در طراحی ما:

۱. گره جدید آدرس یک seed را می‌داند.

۲. پیام HELLO به seed ارسال می‌شود (در صورت نیاز همراه PoW).

۳. پیام GET_PEERS با سقف peer_limit ارسال می‌شود.

۴. پاسخ PEERS_LIST دریافت و Peer List اولیه ساخته می‌شود.

۵. برای همسایه‌های تازه کشف‌شده، HELLO تکمیلی ارسال می‌کنیم تا اتصال دوطرفه سریع‌تر شکل بگیرد.

۲.۴.۲ ب) مدیریت همسایه و تشخیص گره مرده

در طراحی ما، نگه‌داری همسایه‌ها دوره‌ای و محدود است:

- هر ping_interval ثانیه، PING برای حداکثر fanout همسایه ارسال می‌شود.
- اگر همسایه بعد از peer_timeout پاسخ ندهد، شمارنده missed_pongs افزایش می‌یابد.
- بعد از چند پنجره عدم پاسخ (در پیاده‌سازی: ۳ بار)، همسایه حذف می‌شود.
- اگر ظرفیت peer_limit پر باشد، سیاست جایگزینی ما حذف «قدیمی‌ترین last_seen» است.

۳.۴.۲ ج) منطق انتشار Gossip (Push)

در دریافت GOSSIP:

۱. اگر msg_id قبلاً در seen_msgs باشد، پیام نادیده گرفته می‌شود.

۲. در غیر این صورت، msg_id ثبت، زمان دریافت لاگ، و پیام در کش ذخیره می‌شود.

۳. اگر ttl <= 0 باشد انتشار متوقف می‌شود.

۴. اگر ttl > 0 باشد پیام با ttl-1 به حداکثر fanout همسایه تصادفی (بدون تکرار) ارسال می‌شود.

۴.۴.۲ د) تصمیم‌های تکمیلی در طراحی

- یک حلقه کشف دوره‌ای همسایه با GET_PEERS داریم تا overlay در طول زمان متصل بماند.
- حالت Hybrid Push-Pull با IHAVE و IWANT به‌صورت افزونه طراحی شد تا در فازهای بعدی مقایسه عملکردی ممکن باشد.
- در پذیرش HELLO می‌توانیم PoW را اجباری کنیم تا هزینه ساخت هویت‌های جعلی بالا برود.

۳ فاز ۲: پیاده‌سازی

۱.۳ ۱.۲.۴ الزامات پیاده‌سازی

۱.۱.۳ برنامه اجرایی **node** و پارامترهای CLI

گره به‌صورت یک برنامه اجرایی از خط فرمان اجرا می‌شود و پارامترهای اصلی پروژه را دریافت می‌کند.

```
def parse_args(argv: Optional[List[str]] = None) -> NodeConfig:
    parser = argparse.ArgumentParser(description="UDP Gossip node")
    parser.add_argument("--port", type=int, required=True)
    parser.add_argument("--bootstrap", type=str, default=None, help="bootstrap ip:port")
    parser.add_argument("--fanout", type=int, default=3)
    parser.add_argument("--ttl", type=int, default=8)
    parser.add_argument("--peer-limit", type=int, default=50)
    parser.add_argument("--ping-interval", type=float, default=2.0)
    parser.add_argument("--peer-timeout", type=float, default=6.0)
    parser.add_argument("--seed", type=int, default=42)
```

```
    parser.add_argument("--pull-interval", type=float, default=2.0, help="hybrid IHAVE
        interval seconds")
    parser.add_argument("--ihave-max-ids", type=int, default=32, help="max msg_ids per IHAVE")
    parser.add_argument("--pow-k", type=int, default=0, help="PoW difficulty (leading zero hex
         digits)")
    parser.add_argument("--stdin", type=str, default="true", help="enable stdin gossip loop (
        true/false)")
    parser.add_argument("--discovery-interval", type=float, default=4.0, help="periodic
        GET_PEERS interval seconds")

    args = parser.parse_args(argv)
    bootstrap_tuple: Optional[Tuple[str, int]] = None
    if args.bootstrap:
        try:
            host, port_str = args.bootstrap.split(":")
            bootstrap_tuple = (host, int(port_str))
        except Exception:
            raise SystemExit("--bootstrap must be in form ip:port")

    return NodeConfig(
        port=args.port,
        bootstrap=bootstrap_tuple,
        fanout=args.fanout,
        ttl=args.ttl,
        peer_limit=args.peer_limit,
        ping_interval=args.ping_interval,
        peer_timeout=args.peer_timeout,
        seed=args.seed,
        pull_interval=args.pull_interval,
        ihave_max_ids=args.ihave_max_ids,
        pow_k=max(0, int(args.pow_k)),
        stdin_enabled=_parse_bool(args.stdin),
        discovery_interval=max(0.0, float(args.discovery_interval)),
    )
```

کد 1: ساخت و آرگومان‌ها پارس NodeConfig فایل در gossip_node.py

در این قسمت چند نکته اجرایی مهم وجود دارد:

- پارامترهای اجباری و پایه دقیقا همان چیزهایی هستند که برای اجرای گره لازم داریم: port, bootstrap, fanout, ttl, peer-limit, ping-interval, peer-timeout, seed.
- پارامترهای تکمیلی هم برای رفتارهای پیشرفته اضافه شده‌اند: pull-interval, ihave-max-ids, pow-k, discovery-interval, stdin.
- خروجی نهایی parse به‌جای دیکشنری پراکنده، وارد یک ساختار مشخص NodeConfig می‌شود؛ این کار باعث می‌شود تمام بخش‌های گره با یک قرارداد ثابت کار کنند.

۲.۱.۳ اجرای هم‌زمان وظایف گره

گره باید هم‌زمان پیام دریافت کند، پیام دوره‌ای بفرستد، و در حالت تعاملی پیام جدید تولید کند.

```
    async def start(self) -> None:
        loop = asyncio.get_running_loop()
        transport, _ = await loop.create_datagram_endpoint(
            lambda: GossipNodeProtocol(self),
            local_addr=("0.0.0.0", self.cfg.port),
        )
        self.transport = transport  # type: ignore[assignment]

        if self.cfg.bootstrap is not None:
            bootstrap_addr = self.cfg.bootstrap
            logger.info("Bootstrapping via %s:%d", *bootstrap_addr)
            hello = self._make_hello()
            self._send_raw(hello, bootstrap_addr)
```

```
            get_peers = self._make_get_peers(self.cfg.peer_limit)
            self._send_raw(get_peers, bootstrap_addr)

        tasks = [
            asyncio.create_task(self._ping_loop()),
            asyncio.create_task(self._hybrid_pull_loop()),
            asyncio.create_task(self._peer_discovery_loop()),
        ]
        if self.cfg.stdin_enabled:
            tasks.append(asyncio.create_task(self._stdin_gossip_loop()))

        try:
            await asyncio.gather(*tasks)
        except asyncio.CancelledError:
```

کد 2: راه‌اندازی endpoint و taskهای موازی در فایل gossip_node.py

پیاده‌سازی ما این هم‌زمانی را با asyncio انجام می‌دهد:

- endpoint روی UDP ساخته می‌شود و در ادامه event-loop فعال می‌ماند.
- هم‌زمان ۳ حلقه‌ی دوره‌ای اصلی اجرا می‌شوند: ping_loop، hybrid_pull_loop و peer_discovery_loop.
- اگر stdin فعال باشد، حلقه‌ی stdin_gossip_loop هم اضافه می‌شود تا کاربر مستقیما پیام جدید تزریق کند.
- با این ساختار، requirement «گوش‌دادن + ارسال دوره‌ای + تولید پیام» بدون thread جداگانه برآورده می‌شود.

۳.۱.۳ پردازش پیام‌های ورودی و dispatch

هر دیتاگرام دریافتی پس از decode و parse شدن، بر اساس msg_type به handler مربوطه فرستاده می‌شود.

```
    async def handle_datagram(self, data: bytes, addr: Tuple[str, int]) -> None:
        try:
            text = data.decode("utf-8")
            msg = json.loads(text)
        except Exception:
            logger.warning("Invalid JSON from %s", addr)
            return

        msg_type = msg.get("msg_type")
        sender_id = msg.get("sender_id")
        ttl = msg.get("ttl", 0)
        payload = msg.get("payload", {}) or {}

        if not msg_type or not sender_id:
            logger.warning("Missing fields in message from %s", addr)
            return

        logger.info(
            "RECV type=%s msg_id=%s from=%s:%d",
            msg_type,
            msg.get("msg_id"),
            addr[0],
            addr[1],
        )

        if sender_id != self.node_id and msg_type != "HELLO":
            self._update_peer(sender_id, addr)

        if msg_type == "HELLO":
            await self._on_hello(msg, addr)
        elif msg_type == "GET_PEERS":
            await self._on_get_peers(msg, addr)
```

```
        elif msg_type == "PEERS_LIST":
            await self._on_peers_list(msg)
        elif msg_type == "PING":
            await self._on_ping(msg, addr)
        elif msg_type == "PONG":
            await self._on_pong(msg)
        elif msg_type == "GOSSIP":
            await self._on_gossip(msg, ttl, sender_id)
        elif msg_type == "IHAVE":
            await self._on_ihave(msg, sender_id)
        elif msg_type == "IWANT":
            await self._on_iwant(msg, sender_id)
        else:
            logger.warning("Unknown msg_type=%s from %s", msg_type, addr)
```

کد 3: فایل gossip_node.py در پیام dispatch و دریافت مسیر

جریان اجرا در این بخش مرحله‌ای است:

۱. JSON. parse و decode

۲. بررسی فیلدهای اجباری مثل msg_type و sender_id.

۳. به‌روزرسانی peer در پیام‌های غیر HELLO.

۴. dispatch به تابع اختصاصی همان پیام (_on_hello, _on_get_peers, ...).

این جداسازی باعث شده منطق هر پیام مستقل بماند و توسعه/دیباگ ساده‌تر شود.

۴.۱.۳ لاگ‌گیری رویدادهای مهم

برای مشاهده رفتار شبکه، ارسال و دریافت پیام‌ها به‌صورت ساختاریافته لاگ می‌شوند.

```
    def _send_raw(self, msg: Dict, addr: Tuple[str, int]) -> None:
        if not self.transport:
            return
        try:
            data = json.dumps(msg).encode("utf-8")
        except Exception as e:
            logger.error("JSON encode error: %s", e)
            return
        self.transport.sendto(data, addr)
        logger.info(
            "SEND type=%s msg_id=%s dest=%s:%d at_ms=%d",
            msg.get("msg_type"),
            msg.get("msg_id"),
            addr[0],
            addr[1],
            int(time.time() * 1000),
        )
```

کد 4: فایل gossip_node.py در پیام ارسال لاگ ثبت

```
        logger.info(
            "GOSSIP_RECEIVED node_id=%s port=%d msg_id=%s topic=%s data=%s from=%s at_ms=%d
                origin_ts=%s",
            self.node_id,
            self.self_addr[1],
            msg_id,
            topic,
            data,
            msg.get("sender_addr"),
            now_ms,
            payload.get("origin_timestamp_ms"),
```

```
        )
```

کد 5: دریافت لاگ ثبت GOSSIP در فایل gossip_node.py

فرمت لاگ‌ها طوری انتخاب شده که در تحلیل خودکار قابل parse باشند:

- در ارسال: نوع پیام، msg_id، مقصد و timestamp.
- در دریافت GOSSIP: پورت گیرنده، msg_id، زمان دریافت و origin_ts.

به همین دلیل فاز تحلیل می‌تواند بدون دست‌کاری دستی، معیارها را مستقیما از لاگ استخراج کند.

۲.۳ ۲.۲.۴ الزامات درستی

۱.۲.۳ پردازش هر **id_msg** حداکثر یک‌بار

برای جلوگیری از پردازش تکراری، قبل از هر عمل روی GOSSIP، مجموعه seen_msgs بررسی می‌شود.

```
    async def _on_gossip(self, msg: Dict, ttl: int, sender_id: str) -> None:
        msg_id = msg.get("msg_id")
        if not msg_id:
            return
        if msg_id in self.seen_msgs:
            return
        self.seen_msgs.add(msg_id)
        now_ms = int(time.time() * 1000)
        payload = msg.get("payload") or {}
        topic = payload.get("topic")
        data = payload.get("data")
        logger.info(
            "GOSSIP_RECEIVED node_id=%s port=%d msg_id=%s topic=%s data=%s from=%s at_ms=%d
                origin_ts=%s",
            self.node_id,
            self.self_addr[1],
            msg_id,
            topic,
            data,
            msg.get("sender_addr"),
            now_ms,
            payload.get("origin_timestamp_ms"),
        )
        self.gossip_cache[msg_id] = msg
        if len(self.gossip_cache) > 1024:
            first_key = next(iter(self.gossip_cache.keys()))
            self.gossip_cache.pop(first_key, None)
        if ttl <= 0:
            return
        fwd = dict(msg)
        fwd["ttl"] = ttl - 1
        self._broadcast(fwd, exclude=sender_id)
```

کد 6: Deduplication و بازپخش کنترل‌شده در فایل gossip_node.py

در این منطق:

- اگر msg_id قبلا در seen_msgs باشد، پیام همان‌جا drop می‌شود.
- فقط پیام جدید وارد cache شده و امکان forward پیدا می‌کند.
- با این کار، بازپخش‌های حلقه‌ای یا انفجاری ناشی از تکرار دریافت کنترل می‌شود.

۲.۲.۳ ایمنی در برابر پیام خراب یا JSON نامعتبر

خرابی در parse پیام نباید باعث crash شود.

```
    async def handle_datagram(self, data: bytes, addr: Tuple[str, int]) -> None:
        try:
            text = data.decode("utf-8")
            msg = json.loads(text)
        except Exception:
            logger.warning("Invalid JSON from %s", addr)
            return
```

کد 7: gossip_node.py فایل در نامعتبر پیام مدیریت

اگر fail parse شود، فقط warning ثبت می‌شود و گره ادامه می‌دهد؛ بنابراین یک بسته خراب کل نود را متوقف نمی‌کند.

۳.۲.۳ جلوگیری از انتشار بی‌نهایت با **TTL**

در منطق GOSSIP، شرط ttl <= 0 صریحاً بررسی می‌شود و در آن نقطه انتشار قطع می‌شود. بازپخش فقط با ttl-1 انجام می‌گیرد؛ بنابراین زنجیره انتشار کران‌دار است.

۴.۲.۳ کنترل رشد **Peer List**

لیست همسایه‌ها سقف دارد و هنگام پر بودن، سیاست جایگزینی فعال می‌شود.

```
    def _update_peer(self, node_id: str, addr: Tuple[str, int]) -> None:
        now = time.time()
        if node_id in self.peers:
            p = self.peers[node_id]
            p.addr = addr
            p.last_seen = now
        else:
            if len(self.peers) >= self.cfg.peer_limit:
                oldest_id = min(self.peers.items(), key=lambda kv: kv[1].last_seen)[0]
                logger.info("Dropping peer %s due to peer_limit", oldest_id)
                self.peers.pop(oldest_id, None)
            self.peers[node_id] = PeerInfo(node_id=node_id, addr=addr, last_seen=now)
```

کد 8: gossip_node.py فایل در جایگزینی سیاست و peer_limit اعمال

در این سیاست، همسایه‌ای که قدیمی‌ترین last_seen را دارد حذف می‌شود.

۵.۲.۳ تشخیص همسایه مرده

پایش زنده‌بودن همسایه‌ها با PING/PONG دوره‌ای انجام می‌شود.

```
    async def _ping_loop(self) -> None:
        while True:
            await asyncio.sleep(self.cfg.ping_interval)
            now = time.time()
            dead: List[str] = []
            for pid, p in self.peers.items():
                if p.last_ping > 0 and (now - p.last_ping) > self.cfg.peer_timeout and (now -
                    p.last_seen) > self.cfg.peer_timeout:
                    p.missed_pongs += 1
                    p.last_ping = now
                    logger.debug("Peer %s missed pong (%d/%d)", pid, p.missed_pongs, self.
                        _max_missed_pongs)
                    if p.missed_pongs >= self._max_missed_pongs:
                        dead.append(pid)
```

```
            for pid in dead:
                info = self.peers.pop(pid, None)
                logger.info("Peer %s removed (timeout) addr=%s", pid, info.addr if info else
                    None)
```

کد 9: تشخیص timeout در همسایه فایل gossip_node.py

اگر همسایه چند پنجره متوالی پاسخ ندهد، از لیست همسایه‌ها حذف می‌شود.

۶.۲.۳ آزمون ۱۰ نود (حداقل ۹ گیرنده)

برای پوشش requirement نهایی، یک تست مستقل آماده شده که ۱۰ نود را بالا می‌آورد، یک GOSSIP تزریق می‌کند، و تعداد گیرنده‌ها را چک می‌کند.

```
def main() -> int:
    run_id = int(time.time() * 1000)
    logs_dir = PROJECT_ROOT / "logs" / "tests_10_nodes_min9" / f"run_{run_id}"
    logs_dir.mkdir(parents=True, exist_ok=True)

    base_port = _reserve_free_udp_port_block(NODES_COUNT)
    ports = [base_port + i for i in range(NODES_COUNT)]
    bootstrap = ("127.0.0.1", ports[0])

    procs: List[subprocess.Popen] = []
    try:
        for i, p in enumerate(ports):
            boot = None if i == 0 else bootstrap
            procs.append(_start_node(log_file=logs_dir / f"node_{p}.log", port=p, bootstrap=
                boot, seed=500 + i))

        time.sleep(WARMUP_SECONDS)
        msg_id = str(uuid.uuid4())
        _inject_gossip(target_port=ports[1], msg_id=msg_id)
        time.sleep(PROPAGATION_SECONDS)
    finally:
        _stop_processes(procs)

    receivers = _count_receivers(logs_dir, msg_id)
    passed = len(receivers) >= MIN_RECEIVERS
    summary = {
        "test": "10_nodes_min9_receivers",
        "msg_id": msg_id,
        "base_port": base_port,
        "receivers_count": len(receivers),
        "required_min_receivers": MIN_RECEIVERS,
        "receivers_ports": sorted(receivers),
        "logs_dir": str(logs_dir),
        "result": "PASS" if passed else "FAIL",
    }
    with (logs_dir / "summary.json").open("w", encoding="utf-8") as f:
        json.dump(summary, f, indent=2)

    print(json.dumps(summary, indent=2))
    return 0 if passed else 1
```

کد 10: تست ارزیابی منطق فایل در نود ۱۰ test_gossip_10_nodes_min9_receivers.py

توضیح کد بالا:

۱. ابتدا یک پوشه run_<timestamp> در مسیر logs/tests_10_nodes_min9 ساخته می‌شود.

۲. سپس ۱۰ نود روی یک بازه پورت آزاد اجرا می‌شوند (نود اول bootstrap است).

۳. بعد از warmup، یک پیام GOSSIP با msg_id یکتا تزریق می‌شود و زمان انتشار منتظر می‌مانیم.

۴. در پایان، از روی لاگ‌ها تعداد گیرنده‌های پیام شمرده می‌شود و نتیجه PASS/FAIL همراه جزئیات در summary.json ذخیره می‌شود.

برای مشاهده نتایج حاصل شده می‌توانید به مسیر زیر در پروژه مراجعه کنید:

```
logs/tests_10_nodes_min9/run_1772114409341
```

این مسیر در پروژه موجود است و شامل summary.json و لاگ ۱۰ نود است.

جدول ۱: خلاصه اجرای تست ۱۰ نود از summary.json

msg_id	required_min_receivers	receivers_count	result
35d668bd-0e8f-48fb-9fb4-5c7aacf5127a	۹	۱۰	PASS

همچنین بررسی خطوط GOSSIP_RECEIVED در همان run نشان می‌دهد:

- هر ۱۰ پورت 30892..30901 پیام را دریافت کرده‌اند.
- بازه زمانی اولین تا آخرین دریافت فقط 9 ms بوده است.
- زمان‌های ثبت‌شده at_ms بین 1772114414361 تا 1772114414370 هستند.

۴ فاز ۳: شبیه‌سازی و تحلیل عملکرد

۱.۴ ۱.۳.۴ اسکریپت اجرای سناریوها (**simulation.py**)

```
def main() -> None:
    ap = argparse.ArgumentParser(description="Run required Gossip experiments with separated outputs")
    ap.add_argument("--part", choices=["all", "part2", "part3"], default="all")
    ap.add_argument("--mode", choices=["push", "hybrid", "both"], default="both", help="Used for part2")
    ap.add_argument("--out-root", default="results")
    ap.add_argument("--N", type=int, default=None, help="legacy single-N override")
    ap.add_argument("--Ns", type=int, nargs="+", default=[10, 20, 50], help="N values for part2")
    ap.add_argument("--runs-per-n", type=int, default=5)
    ap.add_argument("--base-port", type=int, default=None, help="fixed base port; default allocates per run")
    ap.add_argument("--seed", type=int, default=DEFAULT_START_SEED)
    ap.add_argument("--warmup-s", type=float, default=5.0)
    ap.add_argument("--runtime-s", type=float, default=35.0)
    ap.add_argument("--default-ttl", type=int, default=8)
    ap.add_argument("--default-fanout", type=int, default=3)
    ap.add_argument("--ttl-values", type=int, nargs="+", default=[4, 8, 12])
    ap.add_argument("--fanout-values", type=int, nargs="+", default=[2, 3, 5, 8])
    ap.add_argument("--n-fixed", type=int, default=50, help="N used in part3 sweeps")
    ap.add_argument("--peer-limit", type=int, default=None)
    ap.add_argument("--pull-interval", type=float, default=2.0)
    ap.add_argument("--ping-interval", type=float, default=2.0)
    ap.add_argument("--peer-timeout", type=float, default=8.0)
    ap.add_argument("--discovery-interval", type=float, default=2.0)
    ap.add_argument("--ihave-max-ids", type=int, default=32)
    ap.add_argument("--pow-k", type=int, default=0)
    ap.add_argument("--python", default=sys.executable)
```

```
    ap.add_argument("--node-script", default="gossip_node.py")
    ap.add_argument("--inject-message", default="hello")
    args = ap.parse_args()
```

کد 11: اجرای پارامترهای batch در فایل simulation.py

در این بخش، سناریوهای قابل اجرا و پارامترهای اصلی تعریف می‌شوند:

- برای مقیاس شبکه، Ns = {10,20,50} و runs-per-n=5 پشتیبانی شده است.
- برای فاز ۳، sweepهای ttl-values و fanout-values به‌صورت مستقیم از CLI قابل تنظیم‌اند.
- n-fixed مشخص می‌کند که در سویپ TTL/Fanout همه runها روی یک اندازه ثابت شبکه انجام شوند (در خروجی فعلی: N=50).

```
def _run_single(
    *,
    node_script: Path,
    out_dir: Path,
    tag: str,
    mode: str,
    N: int,
    ttl: int,
    fanout: int,
    seed: int,
    fixed_base_port: Optional[int],
    warmup_s: float,
    runtime_s: float,
    inject_message: str,
    peer_limit: Optional[int],
    pull_interval: float,
    ping_interval: float,
    peer_timeout: float,
    discovery_interval: float,
    ihave_max_ids: int,
    pow_k: int,
    python_exe: str,
) -> Tuple[Path, int, str]:
    suffix = int(time.time() * 1000)
    run_dir = out_dir / _make_run_dir_name(tag, N, fanout, ttl, seed, mode, suffix)
    run_dir.mkdir(parents=True, exist_ok=True)
    base_port = fixed_base_port if fixed_base_port is not None else _allocate_free_port_block(
        N)

    plimit = peer_limit if peer_limit is not None else N
    procs: List[subprocess.Popen] = []
    mid = ""
    try:
        bootstrap = ("127.0.0.1", base_port)
        procs.append(
            _start_node_process(
                python_exe, node_script, base_port, None, fanout, ttl, plimit, seed, mode,
                pull_interval, ping_interval, peer_timeout, discovery_interval,
                ihave_max_ids, pow_k, run_dir / f"node_{base_port}.log"
            )
        )
        for i in range(1, N):
            port = base_port + i
            procs.append(
                _start_node_process(
                    python_exe, node_script, port, bootstrap, fanout, ttl, plimit, seed + i,
                        mode,
                    pull_interval, ping_interval, peer_timeout, discovery_interval,
                    ihave_max_ids, pow_k, run_dir / f"node_{port}.log"
                )
            )
        time.sleep(max(0.0, warmup_s))
        mid = _inject_udp_gossip(base_port, ttl, inject_message)
```

```
        print(f"[INJECT] msg_id={mid} mode={mode} tag={tag} N={N} F={fanout} T={ttl} port={
            base_port}")
        time.sleep(max(0.0, runtime_s))
    finally:
        _terminate_processes(procs, grace_seconds=2.0)
    return run_dir, base_port, mid
```

کد 12: فایل simulation.py در پیام تزریق تا راه‌اندازی از کامل run یک اجرای

منطق اجرای هر run مرحله‌به‌مرحله است:

۱. یک بلوک پورت آزاد رزرو می‌شود تا تداخل بین runها رخ ندهد.

۲. نود پایه اجرا می‌شود و بقیه نودها با bootstrap به آن متصل می‌شوند.

۳. بعد از warmup یک GOSSIP با msg_id یکتا تزریق می‌شود.

۴. پس از runtime کل فرآیندها جمع می‌شوند و لاگ هر نود داخل پوشه run ذخیره می‌شود.

```
    if args.part in ("all", "part3"):
        mode = "push"
        for ttl in sorted(set(args.ttl_values)):
            for i in range(args.runs_per_n):
                seed = args.seed + i
                run_dir, base_port, msg_id = _run_single(
                    node_script=node_script,
                    out_dir=part3_logs / "ttl_sweep",
                    tag="part3ttl",
                    mode=mode,
                    N=args.n_fixed,
                    ttl=ttl,
                    fanout=args.default_fanout,
                    seed=seed,
                    fixed_base_port=args.base_port,
                    warmup_s=args.warmup_s,
                    runtime_s=args.runtime_s,
                    inject_message=args.inject_message,
                    peer_limit=args.peer_limit,
                    pull_interval=args.pull_interval,
                    ping_interval=args.ping_interval,
                    peer_timeout=args.peer_timeout,
                    discovery_interval=args.discovery_interval,
                    ihave_max_ids=args.ihave_max_ids,
                    pow_k=args.pow_k,
                    python_exe=args.python,
                )
                manifest_rows.append(
                    {
                        "part": "part3_ttl",
                        "tag": "part3ttl",
                        "mode": mode,
                        "N": args.n_fixed,
                        "fanout": args.default_fanout,
                        "ttl": ttl,
                        "seed": seed,
                        "base_port": base_port,
                        "msg_id": msg_id,
                        "run_dir": str(run_dir),
                    }
                )
                print(f"[OK] part3 ttl_sweep N={args.n_fixed} F={args.default_fanout} T={ttl}
                    seed={seed} -> {run_dir}")

        for fanout in sorted(set(args.fanout_values)):
            for i in range(args.runs_per_n):
                seed = args.seed + i
                run_dir, base_port, msg_id = _run_single(
                    node_script=node_script,
                    out_dir=part3_logs / "fanout_sweep",
```

```
                    tag="part3fan",
                    mode=mode,
                    N=args.n_fixed,
                    ttl=args.default_ttl,
                    fanout=fanout,
                    seed=seed,
                    fixed_base_port=args.base_port,
                    warmup_s=args.warmup_s,
                    runtime_s=args.runtime_s,
                    inject_message=args.inject_message,
                    peer_limit=args.peer_limit,
                    pull_interval=args.pull_interval,
                    ping_interval=args.ping_interval,
                    peer_timeout=args.peer_timeout,
                    discovery_interval=args.discovery_interval,
                    ihave_max_ids=args.ihave_max_ids,
                    pow_k=args.pow_k,
                    python_exe=args.python,
                )
                manifest_rows.append(
                    {
                        "part": "part3_fanout",
                        "tag": "part3fan",
                        "mode": mode,
                        "N": args.n_fixed,
                        "fanout": fanout,
                        "ttl": args.default_ttl,
                        "seed": seed,
                        "base_port": base_port,
                        "msg_id": msg_id,
                        "run_dir": str(run_dir),
                    }
                )
                print(f"[OK] part3 fanout_sweep N={args.n_fixed} F={fanout} T={args.
                    default_ttl} seed={seed} -> {run_dir}")
```

کد 13: سویپ حلقه‌های TTL و Fanout در فاز ۳ فایل simulation.py

فاز ۳ در این بخش کاملا جدا از مقایسه مودها اجرا می‌شود و مود روی push ثابت است:

- سویپ TTL: برای هر مقدار ttl و هر seed یک run ثبت می‌شود.
- سویپ Fanout: برای هر مقدار fanout و هر seed یک run ثبت می‌شود.
- متادیتای هر run (مثل N, ttl, fanout, seed, msg_id) داخل run_manifest.csv ذخیره می‌شود.

۲.۴ ۲.۳.۴ اسکریپت تحلیل لاگ‌ها (**analyze_logs.py**)

```
def _parse_run(run_dir: Path, converge_ratio: float) -> Tuple[Optional[Dict], Optional[str]]:
    m = RUN_RE.search(run_dir.name)
    if not m:
        return None, "run name does not match expected format"

    node_logs = sorted(run_dir.glob("node_*.log"))
    if not node_logs:
        return None, "no node logs"

    injected_msg_id = None
    origin_ts = None
    injected_recv_ms = None
    for lf in node_logs:
        with lf.open("r", errors="ignore") as f:
            for line in f:
                rm = RECV_RE.search(line)
                if not rm:
                    continue
                if rm.group(2) != "127.0.0.1:0":
```

```
                    continue
                injected_msg_id = rm.group(1)
                injected_recv_ms = int(rm.group(3))
                try:
                    origin_ts = int(rm.group(4))
                except Exception:
                    return None, "invalid origin_ts"
                break
        if injected_msg_id is not None:
            break

    if injected_msg_id is None or origin_ts is None:
        return None, "could not find injected message in logs"

    first_recv_at = {}
    send_events: List[Tuple[Optional[int], str, Optional[str]]] = []
    for lf in node_logs:
        first = None
        with lf.open("r", errors="ignore") as f:
            for line in f:
                sm = SEND_RE.search(line)
                if sm:
                    msg_type = sm.group(1)
                    at_ms = int(sm.group(2)) if sm.group(2) else None
                    msg_id = None
                    msg_id_m = re.search(r"\bmsg_id=([^\s]+)\b", line)
                    if msg_id_m:
                        msg_id = msg_id_m.group(1)
                    send_events.append((at_ms, msg_type, msg_id))
                rm = RECV_RE.search(line)
                if rm and rm.group(1) == injected_msg_id:
                    at_ms = int(rm.group(3))
                    if first is None or at_ms < first:
                        first = at_ms
        if first is not None:
            first_recv_at[lf.name] = first

    received_count = len(first_recv_at)
    total_nodes = len(node_logs)
    if received_count == 0:
        return None, "injected message was not received by any node"

    recv_times = sorted(first_recv_at.values())
    k_needed = max(1, math.ceil(converge_ratio * total_nodes))
    converged = 1 if received_count >= k_needed else 0
    t_conv = recv_times[k_needed - 1] if converged else recv_times[-1]
    convergence_ms = t_conv - origin_ts

    if any(ts is not None for ts, _, _ in send_events):
        overhead_until_conv = sum(1 for ts, _, _ in send_events if ts is not None and ts <=
            t_conv)
    else:
        overhead_until_conv = len(send_events)
    gossip_target_until_conv = sum(
        1
        for ts, msg_type, msg_id in send_events
        if msg_type == "GOSSIP" and msg_id == injected_msg_id and (ts is None or ts <= t_conv)
    )
    gossip_target_all_time = sum(
        1 for _, msg_type, msg_id in send_events if msg_type == "GOSSIP" and msg_id ==
            injected_msg_id
    )

    return {
        "tag": m.group("tag"),
        "N": int(m.group("N")),
        "fanout": int(m.group("F")),
        "ttl": int(m.group("T")),
        "seed": int(m.group("seed")),
        "mode": m.group("mode"),
        "run_dir": str(run_dir),
        "msg_id": injected_msg_id,
        "origin_ts": origin_ts,
```

```
        "injected_recv_ms": injected_recv_ms,
        "convergence_ms": convergence_ms,
        "overhead_until_convergence": overhead_until_conv,
        "message_overhead_target_gossip_until_convergence": gossip_target_until_conv,
        "message_overhead_target_gossip_all_time": gossip_target_all_time,
        "coverage_fraction": received_count / float(total_nodes),
        "converged_to_ratio": converged,
        "required_receivers": k_needed,
        "received_receivers": received_count,
        "node_count": total_nodes,
        "total_sends_all_time": len(send_events),
    }, None
```

کد 14: استخراج داده از لاگ و محاسبه معیارهای هر run در فایل analyze_logs.py

این بخش لاگ‌های SEND و GOSSIP_RECEIVED را parse می‌کند و برای هر run خروجی عددی می‌سازد:

- زمان همگرایی: با معیار T95 (پوشش 95%) از روی اولین زمان دریافت هر نود.
- سربار: تعداد ارسال‌های GOSSIP مربوط به پیام تزریق‌شده (message_overhead_target_gossip_all_time).
- پوشش: نسبت نودهای دریافت‌کننده پیام به کل نودها.

```
def _analyze_part3(results_root: Path, converge_ratio: float) -> int:
    ttl_logs = results_root / "part3_push_ttl_fanout" / "logs" / "ttl_sweep"
    fan_logs = results_root / "part3_push_ttl_fanout" / "logs" / "fanout_sweep"
    out_dir = results_root / "part3_push_ttl_fanout" / "analysis"
    ttl_rows, ttl_skipped = _collect_runs(ttl_logs, converge_ratio=converge_ratio)
    fan_rows, fan_skipped = _collect_runs(fan_logs, converge_ratio=converge_ratio)
    if not ttl_rows and not fan_rows:
        print("[WARN] part3: no valid runs found")
        return 0

    plots = 0
    trend_lines: List[str] = []

    if ttl_rows:
        ttl_df = pd.DataFrame(ttl_rows)
        _save_df(ttl_df, out_dir / "part3_ttl_per_run.csv")
        ttl_summary = (
            ttl_df.groupby(["ttl"], as_index=False)
            .agg(
                convergence_mean=("convergence_ms", "mean"),
                convergence_std=("convergence_ms", "std"),
                overhead_mean=("message_overhead_target_gossip_all_time", "mean"),
                overhead_std=("message_overhead_target_gossip_all_time", "std"),
                coverage_mean=("coverage_fraction", "mean"),
                runs=("run_dir", "count"),
            )
            .sort_values("ttl")
        )
        _save_df(ttl_summary, out_dir / "part3_ttl_summary.csv")
        _plot_single_series(
            ttl_df, "ttl", "convergence_ms",
            "Part3 (push): Convergence vs TTL (constant fanout)",
            f"Convergence time to T{int(converge_ratio * 100)} (ms)",
            out_dir / "part3_convergence_vs_ttl.png",
        )
        plots += 1
        _plot_single_series(
            ttl_df, "ttl", "message_overhead_target_gossip_all_time",
            "Part3 (push): Overhead vs TTL (constant fanout)",
            "Target injected message GOSSIP forwards",
            out_dir / "part3_overhead_vs_ttl.png",
        )
        plots += 1
        _plot_bar_with_error(
            ttl_df, "ttl", "convergence_ms",
            "Part3 (push): Convergence by TTL",
```

```
            "Convergence time (ms)",
            out_dir / "part3_convergence_bar_ttl.png",
        )
        plots += 1
        _plot_bar_with_error(
            ttl_df, "ttl", "message_overhead_target_gossip_all_time",
            "Part3 (push): Overhead by TTL",
            "Target injected message GOSSIP forwards",
            out_dir / "part3_overhead_bar_ttl.png",
        )
        plots += 1
        _plot_scatter_with_mean(
            ttl_df, "ttl", "convergence_ms",
            "Part3 (push): Convergence per run by TTL",
            "Convergence time (ms)",
            out_dir / "part3_convergence_runs_ttl.png",
        )
        plots += 1
        _plot_scatter_with_mean(
            ttl_df, "ttl", "message_overhead_target_gossip_all_time",
            "Part3 (push): Overhead per run by TTL",
            "Target injected message GOSSIP forwards",
            out_dir / "part3_overhead_runs_ttl.png",
        )
        plots += 1

        ov = ttl_summary["overhead_mean"].tolist()
        monotonic_ov = all(ov[i] <= ov[i + 1] for i in range(len(ov) - 1))
        trend_lines.append(f"TTL sweep overhead monotonic non-decreasing: {monotonic_ov}")

    if fan_rows:
        fan_df = pd.DataFrame(fan_rows)
        _save_df(fan_df, out_dir / "part3_fanout_per_run.csv")
        fan_summary = (
            fan_df.groupby(["fanout"], as_index=False)
            .agg(
                convergence_mean=("convergence_ms", "mean"),
                convergence_std=("convergence_ms", "std"),
                overhead_mean=("message_overhead_target_gossip_all_time", "mean"),
                overhead_std=("message_overhead_target_gossip_all_time", "std"),
                coverage_mean=("coverage_fraction", "mean"),
                runs=("run_dir", "count"),
            )
            .sort_values("fanout")
        )
        _save_df(fan_summary, out_dir / "part3_fanout_summary.csv")
        _plot_single_series(
            fan_df, "fanout", "convergence_ms",
            "Part3 (push): Convergence vs Fanout (constant TTL)",
            f"Convergence time to T{int(converge_ratio * 100)} (ms)",
            out_dir / "part3_convergence_vs_fanout.png",
        )
        plots += 1
        _plot_single_series(
            fan_df, "fanout", "message_overhead_target_gossip_all_time",
            "Part3 (push): Overhead vs Fanout (constant TTL)",
            "Target injected message GOSSIP forwards",
            out_dir / "part3_overhead_vs_fanout.png",
        )
        plots += 1
        _plot_bar_with_error(
            fan_df, "fanout", "convergence_ms",
            "Part3 (push): Convergence by fanout",
            "Convergence time (ms)",
            out_dir / "part3_convergence_bar_fanout.png",
        )
        plots += 1
        _plot_bar_with_error(
            fan_df, "fanout", "message_overhead_target_gossip_all_time",
            "Part3 (push): Overhead by fanout",
            "Target injected message GOSSIP forwards",
            out_dir / "part3_overhead_bar_fanout.png",
        )
```

```
        plots += 1
        _plot_scatter_with_mean(
            fan_df, "fanout", "convergence_ms",
            "Part3 (push): Convergence per run by fanout",
            "Convergence time (ms)",
            out_dir / "part3_convergence_runs_fanout.png",
        )
        plots += 1
        _plot_scatter_with_mean(
            fan_df, "fanout", "message_overhead_target_gossip_all_time",
            "Part3 (push): Overhead per run by fanout",
            "Target injected message GOSSIP forwards",
            out_dir / "part3_overhead_runs_fanout.png",
        )
        plots += 1

        conv = fan_summary["convergence_mean"].tolist()
        ov = fan_summary["overhead_mean"].tolist()
        monotonic_conv = all(conv[i] >= conv[i + 1] for i in range(len(conv) - 1))
        monotonic_ov = all(ov[i] <= ov[i + 1] for i in range(len(ov) - 1))
        trend_lines.append(f"Fanout sweep convergence monotonic non-increasing: {
            monotonic_conv}")
        trend_lines.append(f"Fanout sweep overhead monotonic non-decreasing: {monotonic_ov}")

    with (out_dir / "part3_trend_checks.txt").open("w", encoding="utf-8") as f:
        for line in trend_lines:
            f.write(line + "\n")
    with (out_dir / "part3_skipped_runs.txt").open("w", encoding="utf-8") as f:
        for name, reason in ttl_skipped + fan_skipped:
            f.write(f"{name} :: {reason}\n")
    return plots
```

کد 15: خروجی‌های تولید summary و trend فایل در ۳ فاز برای analyze_logs.py

در این بخش، خروجی‌های اصلی فاز ۳ ساخته می‌شوند:

- part3_ttl_summary.csv و part3_fanout_summary.csv
- نمودارهای convergence و overhead برای هر سویپ
- فایل part3_trend_checks.txt برای صحت روندهای مورد انتظار

۳.۴ ۳.۳.۴ نحوه اجرای واقعی آزمایش‌ها

```
# A) N-scaling in push mode only (N in {10,20,50}, 5 seeds each)
python3 simulation.py --part part2 --mode push --out-root results_production --runs-per-n 5

# B) Phase-3 sweeps in push mode (N=50)
python3 simulation.py --part part3 --out-root results_production --runs-per-n 5 \
  --n-fixed 50 --ttl-values 4 8 12 --fanout-values 2 3 5 8

# C) Analyze generated logs
python3 analyze_logs.py --results-root results_production --part all --converge-ratio 0.95
```

خروجی runها در results_production ذخیره شده و فایل‌های تحلیل استفاده‌شده در این گزارش از همین مسیر خوانده شده‌اند.

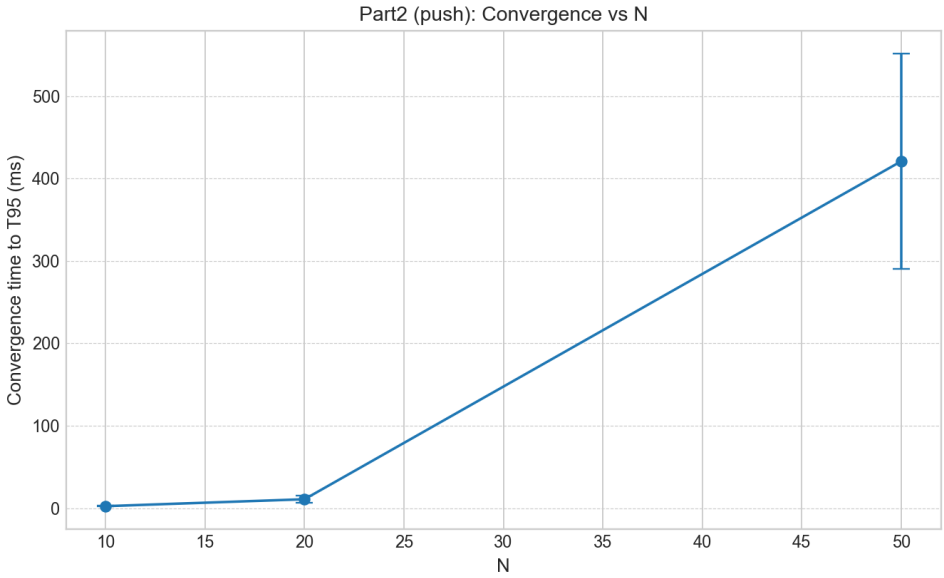
۴.۴ ۴.۳.۴ نتایج و تحلیل دقیق (Push)

۱.۴.۴ الف) مقایسه اندازه شبکه (**N in {10,20,50}**)

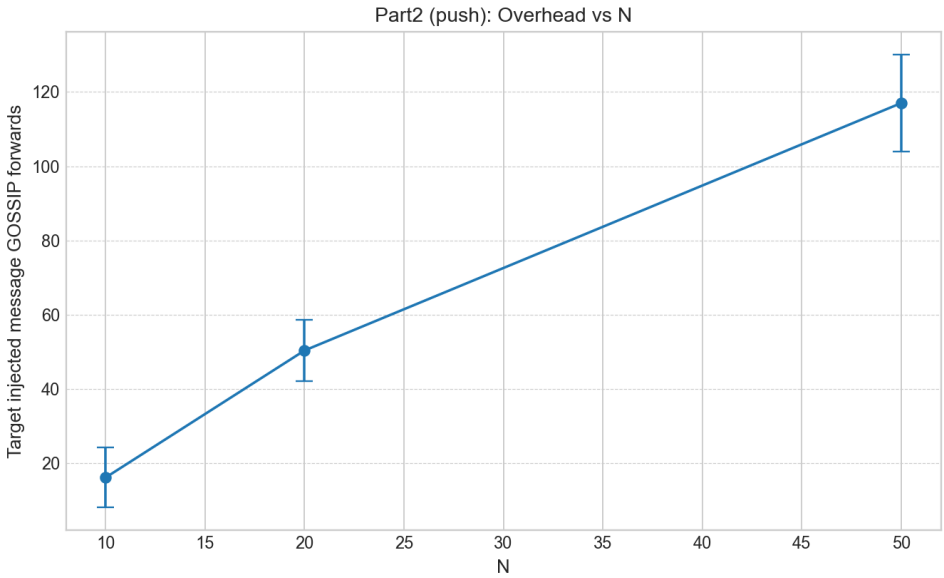
منبع داده: results_production/part2_n_scaling/analysis/part2_summary.csv (فقط ردیف‌های push)

جدول ۲: خلاصه push برای مقایسه N

N	conv_mean(ms)	conv_std	overhead_mean	overhead_std	coverage_mean
10	2.4	0.55	16.2	8.11	0.56
20	10.8	4.21	50.4	8.32	0.84
50	421.0	130.88	117.0	13.08	0.84



شکل ۱: مقایسه زمان همگرایی برای Nهای مختلف در حالت push



شکل ۲: مقایسه سربار پیام برای Nهای مختلف در حالت push

تحلیل: با بزرگ‌تر شدن شبکه، زمان همگرایی به‌صورت غیرخطی رشد کرده است. از N=10 به N=20 زمان همگرایی حدود 4.5x و سربار حدود 3.1x شده است؛ از N=20 به N=50 زمان همگرایی حدود 39x و سربار حدود 2.3x

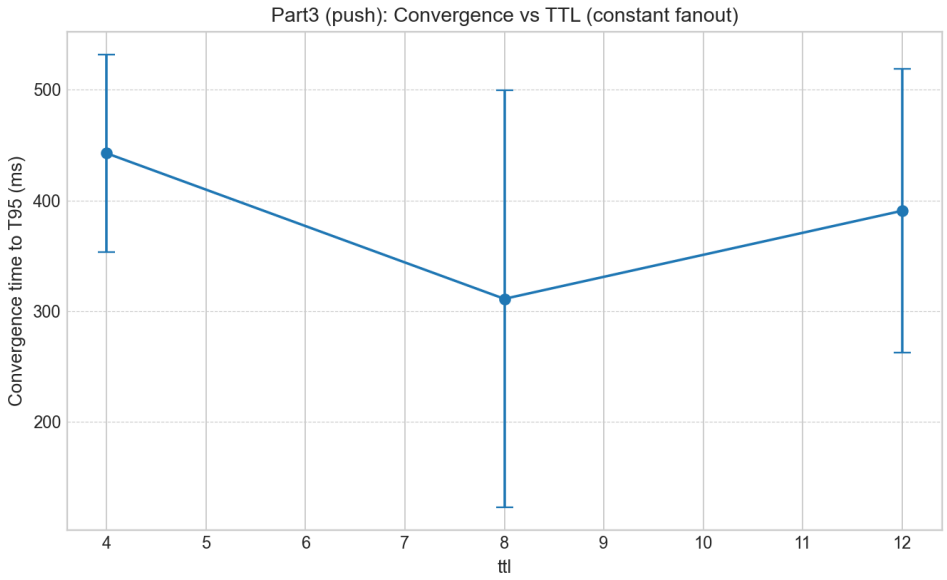
شده است. پوشش میانگین از 0.56 به 0.84 رسیده و برای N=20 و N=50 برابر مانده است. از نظر فنی، با بزرگ شدن گراف و ثابت ماندن fanout/ttl، طول مسیرهای انتشار بیشتر می‌شود و پیام برای رسیدن به نواحی دورتر به hopهای بیشتری نیاز دارد؛ در نتیجه هم دیرتر به آستانه T95 می‌رسیم و هم تعداد فورواردهای هدف افزایش پیدا می‌کند.

۲.۴.۴ ب) سویپ **TTL** (با **N=50** و **fanout=3**)

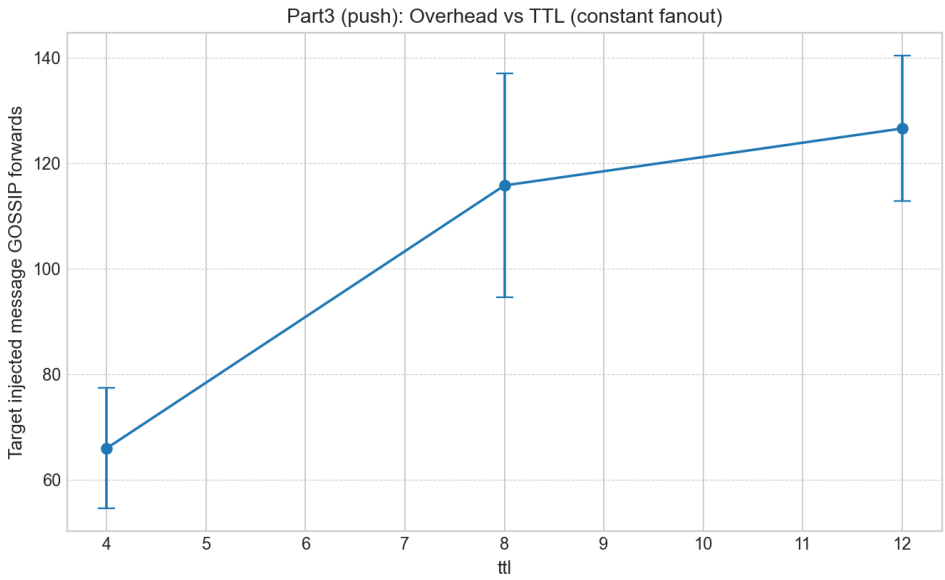
منبع داده: results_production/part3_push_ttl_fanout/analysis/part3_ttl_summary.csv

جدول ۳: خلاصه اثر TTL در حالت push

ttl	conv_mean(ms)	conv_std	overhead_mean	overhead_std	coverage_mean
4	442.6	89.24	66.0	11.42	0.704
8	311.0	188.73	115.8	21.17	0.844
12	390.6	128.34	126.6	13.81	0.844



شکل ۳: مقایسه زمان همگرایی برای TTLهای مختلف در حالت push



شکل ۴: مقایسه سربار پیام برای TTLهای مختلف در حالت push

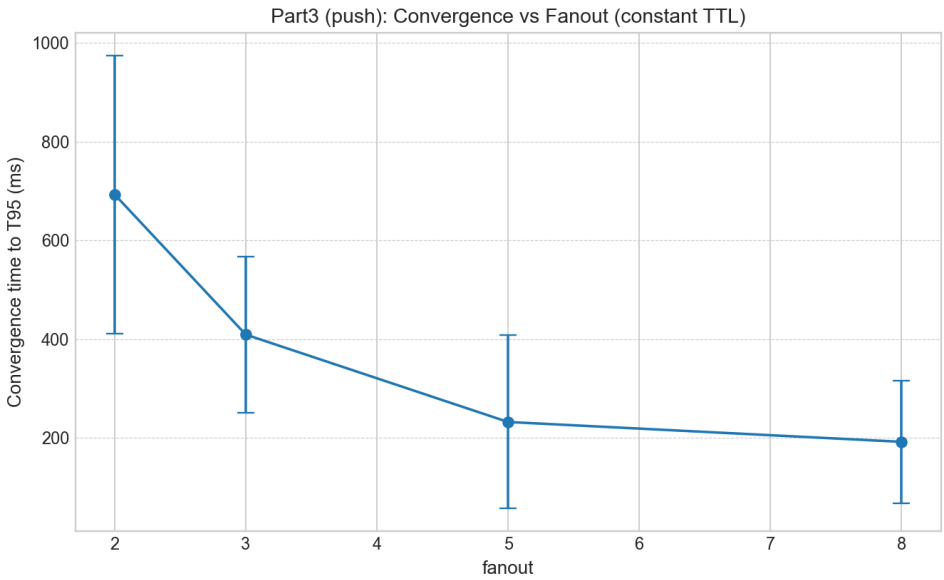
تحلیل: افزایش TTL از 4 به 8 زمان همگرایی را حدود 29.7% کاهش داده، اما سربار 75.5% افزایش یافته است. افزایش بیشتر تا TTL=12 پوشش را بهتر نکرده (0.844) و زمان همگرایی را حدود 25.6% بدتر کرده، در حالی‌که سربار همچنان 9.3% بالا رفته است. پس در این بازه، TTL=8 نقطه متعادل‌تری بین سرعت و هزینه است. توضیح علمی این رفتار این است که TTL بیشتر، عمق جست‌وجوی پیام را افزایش می‌دهد؛ اما بعد از اشباع بخش بزرگی از شبکه، افزایش عمق عمدتاً فورواردهای تکراری تولید می‌کند و سود همگرایی کاهش پیدا می‌کند.

۳.۴.۴ ج) سویپ **Fanout** (با **N=50** و **ttl=8**)

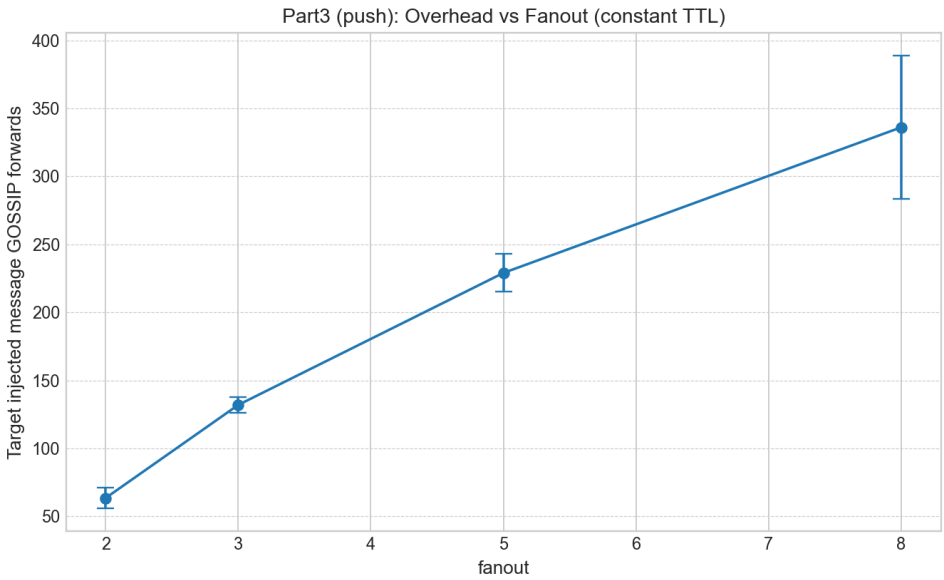
منبع داده: results_production/part3_push_ttl_fanout/analysis/part3_fanout_summary.csv

جدول ۴: خلاصه اثر Fanout در حالت push

fanout	conv_mean(ms)	conv_std	overhead_mean	overhead_std	coverage_mean
2	692.8	281.38	63.6	7.80	0.688
3	409.0	157.74	132.0	5.61	0.900
5	232.0	175.48	229.0	13.87	0.920
8	191.8	124.48	336.0	52.76	0.840



شکل ۵: مقایسه زمان همگرایی برای Fanoutهای مختلف در حالت push



شکل ۶: مقایسه سربار پیام برای Fanoutهای مختلف در حالت push

تحلیل: با افزایش fanout، زمان همگرایی به‌صورت یکنواخت کاهش یافته (692.8 -> 191.8 ms) اما سربار به‌صورت یکنواخت افزایش یافته (63.6 -> 336.0). بیشترین پوشش روی fanout=5 با مقدار 0.92 دیده شده است؛ fanout=8 سریع‌تر است اما با رشد شدید سربار و افت پوشش به 0.84. برای این تنظیمات، fanout=5 تعادل بهتری بین سرعت، هزینه و پوشش نشان می‌دهد. دلیل این trade-off این است که fanout بزرگ‌تر ضریب انشعاب انتشار را بالا می‌برد و رسیدن به نودهای دور را سریع‌تر می‌کند؛ اما هم‌زمان احتمال ارسال به نودهای از قبل مطلع و نیز burst شدن ترافیک UDP را افزایش می‌دهد که می‌تواند نوسان پوشش را زیاد کند.

۵ فاز ۴: بخش تکمیلی اجباری (Hybrid Push-Pull)

۱.۵ ۱.۴.۴ پارامترها و پیام‌های جدید

```python
@dataclass
class NodeConfig:
    port: int
    bootstrap: Optional[Tuple[str, int]]
    fanout: int
    ttl: int
    peer_limit: int
    ping_interval: float
    peer_timeout: float
    seed: int
    # hybrid push-pull
    pull_interval: float = 2.0
    ihave_max_ids: int = 32
    # proof-of-work difficulty (leading zero hex digits)
    pow_k: int = 0
    # stdin gossip loop toggle
    stdin_enabled: bool = True
    # periodic peer discovery via GET_PEERS
    discovery_interval: float = 4.0
```

کد 16: پارامترهای Hybrid در NodeConfig فایل gossip_node.py

```python
    def _make_ihave(self, ids: List[str]) -> Dict:
        msg = self._base_msg("IHAVE")
        msg["payload"] = {"ids": ids[: self.cfg.ihave_max_ids], "max_ids": self.cfg.
            ihave_max_ids}
        return msg

    def _make_iwant(self, ids: List[str]) -> Dict:
        msg = self._base_msg("IWANT")
        msg["payload"] = {"ids": ids}
        return msg
```

کد 17: ساخت payload پیام‌های IHAVE و IWANT فایل gossip_node.py

در این فاز دو پارامتر قابل تنظیم اضافه شده‌اند:

- pull_interval: فاصله زمانی ارسال دوره‌ای IHAVE.
- ihave_max_ids: سقف تعداد شناسه در یک IHAVE، برای کنترل اندازه پیام کنترلی.

در پیام IHAVE فقط شناسه‌ها ارسال می‌شوند و با ids[:ihave_max_ids] محدود می‌گردند؛ در IWANT فقط شناسه‌های لازم درخواست می‌شوند.

۲.۵ ۲.۴.۴ منطق اجرایی Hybrid در خود گره

```python
    async def _on_ihave(self, msg: Dict, sender_id: str) -> None:
        payload = msg.get("payload") or {}
        ids = payload.get("ids") or []
        unknown_ids = [mid for mid in ids if mid not in self.seen_msgs]
        if not unknown_ids:
            return
        logger.debug("IHAVE from %s, requesting %d messages", sender_id, len(unknown_ids))
        iwant = self._make_iwant(unknown_ids)
        peer = self.peers.get(sender_id)
        if peer:
            self._send_raw(iwant, peer.addr)

```

```
    async def _on_iwant(self, msg: Dict, sender_id: str) -> None:
        payload = msg.get("payload") or {}
        ids = payload.get("ids") or []
        peer = self.peers.get(sender_id)
        if not peer:
            return
        for mid in ids:
            msg_obj = self.gossip_cache.get(mid)
            if msg_obj is None:
                continue
            self._send_raw(msg_obj, peer.addr)
        logger.debug("IWANT from %s, sent %d messages", sender_id, len(ids))
```

کد 18: هندلرهای منطق IWANT و IHAVE در فایل gossip_node.py

```
    async def _hybrid_pull_loop(self) -> None:
        while True:
            await asyncio.sleep(self.cfg.pull_interval)
            if self.cfg.pull_interval <= 0:
                continue
            if not self.peers or not self.seen_msgs:
                continue
            peer_items = list(self.peers.items())
            self.random.shuffle(peer_items)
            count = min(self.cfg.fanout, len(peer_items))
            ids_list = list(self.seen_msgs)
            self.random.shuffle(ids_list)
            ihave_msg = self._make_ihave(ids_list)
            for peer_id, p in peer_items[:count]:
                self._send_raw(ihave_msg, p.addr)
            logger.debug("IHAVE broadcast to %d peers", count)
```

کد 19: ارسال دوره‌ای حلقه IHAVE در فایل gossip_node.py

رفتار Hybrid در پیاده‌سازی ما این‌گونه است:

۱. هر pull_interval ثانیه، نود به حداکثر fanout همسایه پیام IHAVE می‌فرستد.

۲. گیرنده، ids دریافتی را با seen_msgs مقایسه می‌کند و فقط idهای ناشناخته را در IWANT می‌خواهد.

۳. فرستنده در پاسخ IWANT، پیام کامل GOSSIP هر id را از gossip_cache برمی‌گرداند.

بنابراین پیام کامل فقط زمانی ارسال می‌شود که واقعاً در مقصد وجود نداشته باشد.

۳.۵ ۳.۴.۴ ادغام Hybrid در اسکریپت آزمایش و تحلیل

```
    cmd = [
        python_exe, str(node_script),
        "--port", str(port),
        "--fanout", str(fanout),
        "--ttl", str(ttl),
        "--peer-limit", str(peer_limit),
        "--seed", str(seed),
        "--ping-interval", str(ping_interval),
        "--peer-timeout", str(peer_timeout),
        "--pull-interval", "0" if mode == "push" else str(pull_interval),
        "--discovery-interval", str(discovery_interval),
        "--ihave-max-ids", str(ihave_max_ids),
        "--pow-k", str(pow_k),
        "--stdin", "false",
    ]
    if bootstrap is not None:
        cmd += ["--bootstrap", f"{bootstrap[0]}:{bootstrap[1]}"]
```

کد 20: رفتار انتخاب push/hybrid هنگام اجرای نودها در فایل simulation.py

```
    if args.part in ("all", "part2"):
        for mode in modes:
            for N in n_values:
                for i in range(args.runs_per_n):
                    seed = args.seed + i
                    tag = "part2"
                    run_dir, base_port, msg_id = _run_single(
                        node_script=node_script,
                        out_dir=part2_logs / mode,
                        tag=tag,
                        mode=mode,
                        N=N,
                        ttl=args.default_ttl,
                        fanout=args.default_fanout,
                        seed=seed,
                        fixed_base_port=args.base_port,
                        warmup_s=args.warmup_s,
                        runtime_s=args.runtime_s,
                        inject_message=args.inject_message,
                        peer_limit=args.peer_limit,
                        pull_interval=args.pull_interval,
                        ping_interval=args.ping_interval,
                        peer_timeout=args.peer_timeout,
                        discovery_interval=args.discovery_interval,
                        ihave_max_ids=args.ihave_max_ids,
                        pow_k=args.pow_k,
                        python_exe=args.python,
                    )
                    manifest_rows.append(
                        {
                            "part": "part2",
                            "tag": tag,
                            "mode": mode,
                            "N": N,
                            "fanout": args.default_fanout,
                            "ttl": args.default_ttl,
                            "seed": seed,
                            "base_port": base_port,
                            "msg_id": msg_id,
                            "run_dir": str(run_dir),
                        }
                    )
                    print(f"[OK] part2 mode={mode} N={N} seed={seed} -> {run_dir}")
```

کد 21: اجرای batch مقایسه‌ای push/hybrid روی Nهای مختلف در فایل simulation.py

```
    summary = (
        df.groupby(["mode", "N"], as_index=False)
        .agg(
            convergence_mean=("convergence_ms", "mean"),
            convergence_std=("convergence_ms", "std"),
            overhead_mean=("message_overhead_target_gossip_all_time", "mean"),
            overhead_std=("message_overhead_target_gossip_all_time", "std"),
            coverage_mean=("coverage_fraction", "mean"),
            converged_ratio_mean=("converged_to_ratio", "mean"),
            runs=("run_dir", "count"),
        )
    )
    _save_df(summary, out_dir / "part2_summary.csv")

    plots = 0
    _plot_compare_modes(
        df, "N", "convergence_ms",
        "Part2: Convergence vs N (Push vs Hybrid)",
        f"Convergence time to T{int(converge_ratio * 100)} (ms)",
        out_dir / "part2_convergence_vs_n_push_vs_hybrid.png",
    )
    plots += 1
    _plot_compare_modes(
        df, "N", "message_overhead_target_gossip_all_time",
        "Part2: Message Overhead vs N (Push vs Hybrid)",
        "Target injected message GOSSIP forwards",
```

```
        out_dir / "part2_overhead_vs_n_push_vs_hybrid.png",
    )
    plots += 1
```

کد 22: تولید summary و مودها مقایسه نمودار فایل در analyze_logs.py

در simulation.py اگر mode برابر push باشد، pull_interval عملاً صفر می‌شود و مسیر Hybrid غیرفعال می‌گردد؛ و اگر mode برابر hybrid باشد، حلقه Pull فعال است. در analyze_logs.py نیز خروجی‌ها برحسب mode تجمیع و نمودارهای مقایسه‌ای تولید می‌شوند.

۴.۵ ۴.۴.۴ مقایسه Push و Hybrid بر اساس داده‌های واقعی

منابع داده:

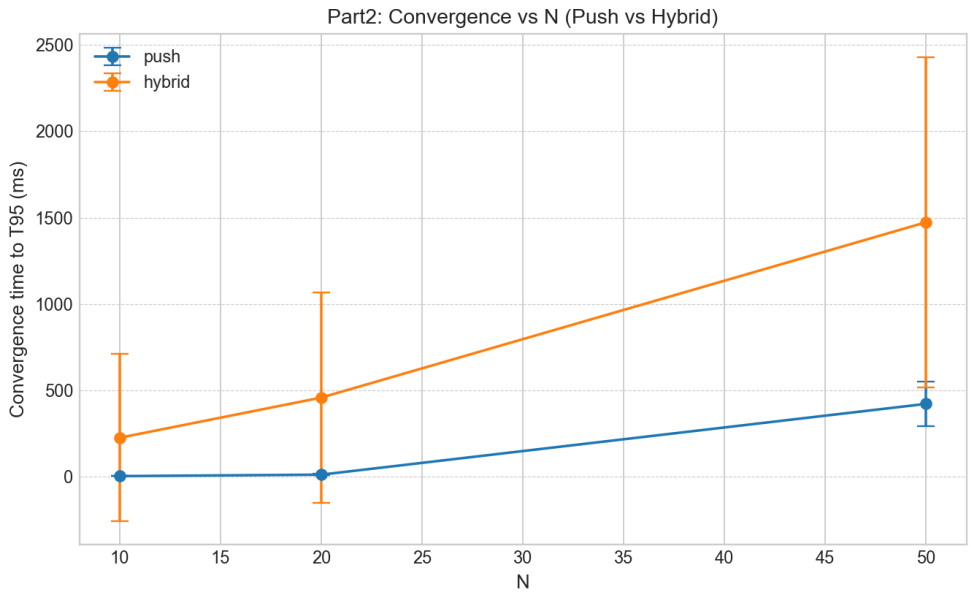
- results_production/part2_n_scaling/analysis/part2_summary.csv
- results_production/part2_n_scaling/analysis/part2_per_run.csv

جدول ۵: مقایسه push/hybrid برحسب معیارهای اصلی

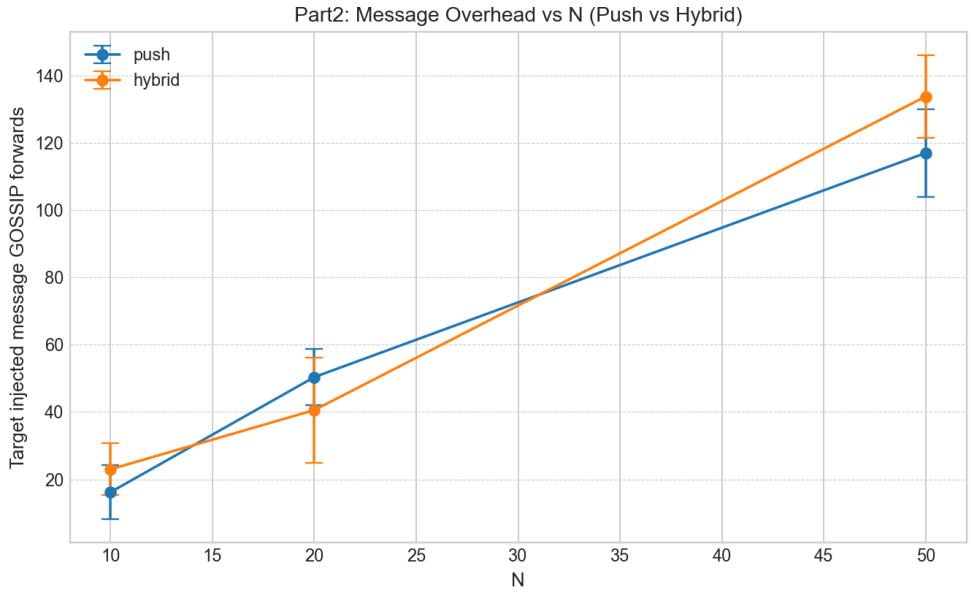
mode	N	conv_mean(ms)	conv_std	target_overhead_mean	coverage_mean	runs
push	10	2.4	0.55	16.2	0.560	5
hybrid	10	224.6	484.90	23.0	0.760	5
push	20	10.8	4.21	50.4	0.840	5
hybrid	20	457.4	610.36	40.6	0.660	5
push	50	421.0	130.88	117.0	0.840	5
hybrid	50	1473.2	956.50	133.8	0.872	5

جدول ۶: مقایسه کل تعداد ارسال‌ها (total_sends_all_time)

mode	N	total_sends_mean	total_sends_std	runs
push	10	1247.2	614.59	5
hybrid	10	1989.6	665.87	5
push	20	3840.8	667.95	5
hybrid	20	3552.0	1411.75	5
push	50	11262.0	862.95	5
hybrid	50	13010.0	1391.04	5



شکل ۷: مقایسه زمان همگرایی push و hybrid برحسب N



شکل ۸: مقایسه سربار هدف GOSSIP در push و hybrid برحسب N

تحلیل مقایسه‌ای: در داده‌های فعلی، Hybrid از نظر زمان همگرایی در هر سه اندازه شبکه کندتر است (93.6x برای N=10، 42.3x برای N=20 و 3.5x برای N=50). علت اصلی این است که مسیر بازیابی پیام در Hybrid سه‌مرحله‌ای است (GOSSIP -> IWANT -> IHAVE) و به تایمر pull_interval وابسته است؛ پس اگر Push اولیه پیام را به بخشی از شبکه نرساند، جبران آن تا سیکل Pull بعدی عقب می‌افتد.

از نظر سربار، دو مشاهده باید جدا تحلیل شوند:

- سربار هدف در نمودار رسمی (target_overhead_mean): در N=20 Hybrid بهتر است (40.6 در برابر 50.4) چون مکانیزم درخواست انتخابی، فورواردهای غیرضروری پیام هدف را کم کرده است؛ اما در N=10 و N=50 بیشتر شده است.

- کل ارسال‌ها (total_sends_mean از part2_per_run.csv): در N=10 و Hybrid N=50 بالاتر است (1989.6 در برابر 1247.2 و 13010.0 در برابر 11262.0) که نشان می‌دهد هزینه پیام‌های کنترلی IHAVE و IWANT و زمان اجرای طولانی‌تر، می‌تواند مزیت کاهش فورووارد مستقیم را خنثی کند.

بنابراین در تنظیمات فعلی، Hybrid بیشتر نقش «ترمیم پوشش» داشته تا «کاهش قطعی سربار و زمان». برای بهینه‌شدن رفتار Hybrid لازم است pull_interval، fanout و حتی سیاست همسایه‌گزینی هم‌زمان تیون شوند تا تأخیر سیکل Pull کمتر و ترافیک کنترلی کنترل شود.

۵.۵ ۵.۴.۴ بخش دوم: مقاومت در برابر Sybil با **PoW**

۱.۵.۵ الف) نحوه پیاده‌سازی در کد

```
    def _make_hello(self) -> Dict:
        msg = self._base_msg("HELLO")
        payload: Dict = {"capabilities": ["udp", "json"]}
        if self.cfg.pow_k > 0:
            if self._hello_pow_payload is None:
                self._hello_pow_payload = self._compute_pow(self.node_id, self.cfg.pow_k)
            payload["pow"] = self._hello_pow_payload
        msg["payload"] = payload
        return msg
```

کد 23: اضافه‌کردن PoW به HELLO در فایل gossip_node.py

این بخش نشان می‌دهد که HELLO همیشه capabilities دارد و اگر pow_k > 0 باشد:

- فقط یک‌بار PoW محاسبه می‌شود (self._hello_pow_payload) تا هزینه join تکراری نشود.
- خروجی PoW داخل payload.pow قرار می‌گیرد و در همه HELLOهای بعدی استفاده می‌شود.

```
    # ----------------- PoW helpers -----------------
    @staticmethod
    def _compute_pow(node_id: str, k: int) -> Dict:
        assert k >= 0
        if k == 0:
            return {
                "hash_alg": "sha256",
                "difficulty_k": 0,
                "nonce": 0,
                "digest_hex": "",
            }
        prefix = "0" * k
        nonce = 0
        while True:
            h = hashlib.sha256(f"{node_id}{nonce}".encode("utf-8")).hexdigest()
            if h.startswith(prefix):
                return {
                    "hash_alg": "sha256",
                    "difficulty_k": k,
                    "nonce": nonce,
                    "digest_hex": h,
                }
            nonce += 1

    @staticmethod
    def _verify_pow(node_id: str, pow_payload: Dict, k: int) -> bool:
        try:
            hash_alg = str(pow_payload.get("hash_alg", "")).lower()
            difficulty_k = int(pow_payload.get("difficulty_k", -1))
            nonce = int(pow_payload.get("nonce", -1))
            digest_hex = str(pow_payload.get("digest_hex", ""))
        except Exception:
```

```
            return False
        if hash_alg != "sha256":
            return False
        if difficulty_k != k:
            return False
        if nonce < 0:
            return False
        computed_digest = hashlib.sha256(f"{node_id}{nonce}".encode("utf-8")).hexdigest()
        return computed_digest == digest_hex and computed_digest.startswith("0" * k)
```

کد 24: اعتبارسنجی و تولید توابع PoW در فایل gossip_node.py

جزئیات مهم این پیاده‌سازی:

- تابع _compute_pow با brute-force روی nonce جلو می‌رود تا SHA256(node_id || nonce) با k صفر هگز شروع شود.
- تابع _verify_pow چهار چیز را چک می‌کند: hash_alg، برابری difficulty_k با k تنظیمی، معتبر بودن nonce، و تطابق کامل digest بازمحاسبه‌شده.
- بنابراین جعل payload بدون حل واقعی معما با احتمال بالا رد می‌شود.

```
    async def _on_hello(self, msg: Dict, addr: Tuple[str, int]) -> None:
        logger.info("HELLO from %s", addr)
        sender_id = msg.get("sender_id")
        payload = msg.get("payload") or {}
        pow_payload = payload.get("pow")
        if self.cfg.pow_k > 0:
            if not isinstance(sender_id, str) or not isinstance(pow_payload, dict):
                logger.warning("HELLO without valid PoW from %s rejected", addr)
                return
            if not self._verify_pow(sender_id, pow_payload, self.cfg.pow_k):
                logger.warning("HELLO with invalid PoW from %s rejected", addr)
                return
        if isinstance(sender_id, str):
            self._update_peer(sender_id, addr)
        reply = self._make_peers_list()
        self._send_raw(reply, addr)
```

کد 25: پذیرش قانون همسایه در HELLO فایل gossip_node.py

در _on_hello اگر pow_k فعال باشد:

- HELLO بدون pow معتبر رد می‌شود.
- HELLO با pow نامعتبر رد می‌شود.
- فقط در صورت موفقیت اعتبارسنجی، گره به Peer List اضافه می‌شود.

```
    parser = argparse.ArgumentParser(description="Benchmark PoW difficulty vs time")
    parser.add_argument("--node-id", type=str, default="benchmark-node")
    parser.add_argument("--trials", type=int, default=5)
    parser.add_argument("--k-values", type=int, nargs="+", default=[0, 2, 3, 4])
    parser.add_argument("--out-csv", type=str, default="k_pow_bench/pow_bench.csv")
    parser.add_argument("--out-png", type=str, default="k_pow_bench/pow_bench.png")
    args = parser.parse_args()
```

کد 26: بنچمارک پارامترهای PoW فایل pow_bench.py

```python
def measure_pow_times(node_id: str, k_values: List[int], trials: int) -> List[dict]:
    results = []
    dummy_cfg = NodeConfig(
        port=0,
        bootstrap=None,
        fanout=1,
        ttl=1,
        peer_limit=1,
        ping_interval=1.0,
        peer_timeout=1.0,
        seed=0,
    )
    dummy_node = GossipNode(dummy_cfg)

    for k in k_values:
        times = []
        for _ in range(trials):
            t0 = time.time()
            _ = dummy_node._compute_pow(node_id, k)
            t1 = time.time()
            times.append(t1 - t0)
        arr = np.array(times)
        results.append(
            {
                "pow_k": k,
                "time_avg_s": float(arr.mean()),
                "time_std_s": float(arr.std(ddof=0)),
            }
        )
    return results
```

کد 27: تولید زمان اندازه‌گیری PoW در فایل pow_bench.py

```python

    csv_path = Path(args.out_csv)
    csv_path.parent.mkdir(parents=True, exist_ok=True)
    with csv_path.open("w", newline="") as f:
        writer = csv.DictWriter(f, fieldnames=["pow_k", "time_avg_s", "time_std_s"])
        writer.writeheader()
        for r in results:
            writer.writerow(r)

    #              PoW      pow_k
    ks = [r["pow_k"] for r in results]
    avgs = [r["time_avg_s"] for r in results]

    plt.figure()
    plt.plot(ks, avgs, marker="o")
    plt.xlabel("pow_k (leading zero hex digits)")
    plt.ylabel("Average PoW generation time (s)")
    plt.title("PoW cost vs difficulty")
    plt.grid(True)
    png_path = Path(args.out_png)
    png_path.parent.mkdir(parents=True, exist_ok=True)
    plt.savefig(png_path, dpi=150)
```

کد 28: ذخیره CSV و نمودار رسم PoW در فایل pow_bench.py

اسکریپت pow_bench.py زمان تولید PoW را برای kهای مختلف اندازه می‌گیرد و خروجی قابل استناد برای گزارش می‌سازد.

۲.۵.۵ ب) شواهد لاگ از رد و پذیرش **HELLO**

برای نمایش رفتار واقعی، یک اجرای دمو در مسیر زیر انجام شد:

`logs/pow_phase4_demo/run_1772133629138`

در لاگ نود seed (با pow_k=4) این دو رخداد ثبت شده است:

```
[2026-02-26 22:50:30,202] WARNING HELLO without valid PoW from ('127.0.0.1', 9301) rejected
[2026-02-26 22:50:30,381] INFO HELLO from ('127.0.0.1', 9302)
```

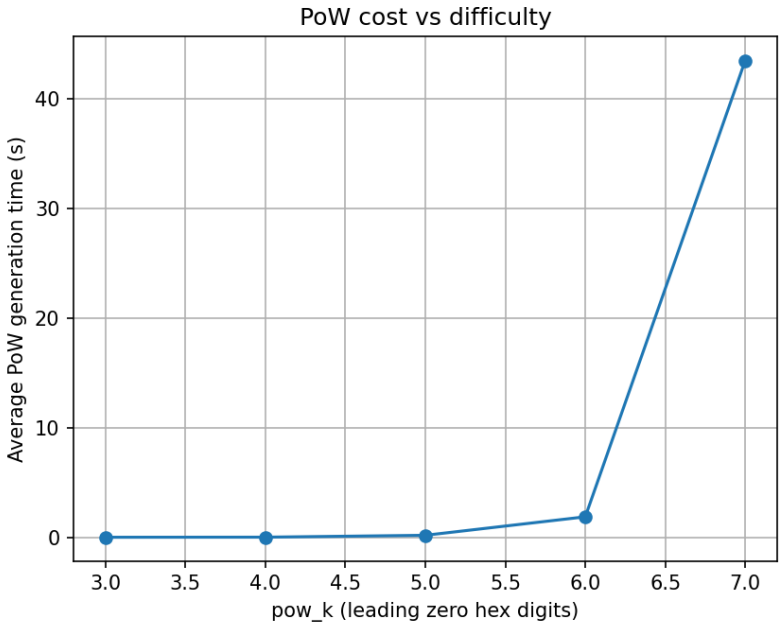
پس مسیر reject برای نود بدون PoW و مسیر accept برای نود دارای PoW معتبر در عمل دیده شده است.

۳.۵.۵ ج) هزینه محاسباتی **PoW** (بر اساس داده واقعی)

منبع داده: k_pow_bench/pow_bench.csv

جدول ۷: زمان تولید PoW برحسب k (میانگین ۵ تکرار)

pow_k	avg(s)	std(s)	approx range(s)	scale vs previous k
3	0.00223	0.00002	[0.00222, 0.00225]	-
4	0.01018	0.00031	[0.00987, 0.01049]	4.56x
5	0.18222	0.00247	[0.17976, 0.18469]	17.89x
6	1.86618	0.03887	[1.82731, 1.90505]	10.24x
7	43.46220	0.21622	[43.24599, 43.67842]	23.29x



شکل ۹: روند افزایش هزینه تولید PoW با افزایش k_pow

۴.۵.۵ د) پاسخ مستقیم به سوال‌های فاز **PoW**

۱. مزیت امنیتی: PoW هزینه ساخت هر هویت جدید را از «تقریبا صفر» به «هزینه CPU واقعی» تبدیل می‌کند. چون احتمال اینکه یک هش تصادفی k صفر هگز ابتدایی داشته باشد برابر ۱۶^{-k} است، تعداد تلاش مورد انتظار $۱۶^{k} \approx$ می‌شود. بنابراین مهاجم Sybil برای ساخت انبوه هویت‌ها هزینه‌ای تقریباً خطی در تعداد هویت و نمایی در k می‌پردازد.

۲. هزینه و اثر جانبی: هزینه تولید nonce روی سیستم ما از 0.01s در k=4 به 1.87s در k=6 و 43.46s در k=7 می‌رسد. پس با افزایش k امنیت بیشتر می‌شود، اما زمان join نود سالم هم مستقیماً زیاد می‌شود و bootstrap شبکه کندتر می‌گردد.

۳. انتخاب **k_pow**: مقدار انتخابی ما برای اجرای شبکه k_pow=4 بوده است. دلیل: در این مقدار هزینه join برای نود سالم بسیار کم (حدود صدم ثانیه) است، ولی برای مهاجم، تولید تعداد زیاد هویت دیگر رایگان نیست. اگر هدف امنیتی شدیدتر باشد، k=5 هم قابل قبول است (حدود 0.18s) اما k>=6 برای سناریوی تعاملی سبک، کند محسوب می‌شود.

۴. جدول/نمودار روند هزینه: جدول بالا و شکل pow_bench.png روند را به‌صورت ملموس نشان می‌دهند: افزایش k باعث رشد تند (نزدیک به نمایی) در زمان تولید PoW می‌شود.

۶ جمع‌بندی

در این گزارش کل چرخه پروژه از طراحی تا ارزیابی نهایی پوشش داده شد: در فاز ۱ معماری گره، قالب پیام‌ها و منطق انتشار/مدیریت همسایه‌ها مشخص شد؛ در فاز ۲ پیاده‌سازی مبتنی بر UDP+JSON با dedup، TTL/Fanout، لاگ‌گیری ساختاریافته و تست ۱۰ نود ارائه شد؛ در فاز ۳ تحلیل عملکرد push روی مقیاس شبکه و سویپ پارامترها انجام شد و trade-off بین سرعت همگرایی و سربار به‌صورت داده‌محور نشان داده شد؛ و در فاز ۴ هر دو بخش تکمیلی Push-Pull (Hybrid) و (PoW با کد، آزمایش و مقایسه کمی بررسی شدند.

برآیند نتایج این است که رفتار شبکه به‌شدت به تنظیم پارامترها وابسته است: fanout/ttl اگر زیاد شوند همگرایی سریع‌تر می‌شود اما هزینه پیام بالا می‌رود؛ Hybrid بدون تنظیم دقیق pull_interval لزوماً سریع‌تر نیست؛ و PoW با سختی مناسب می‌تواند هزینه حمله Sybil را بالا ببرد در حالی که join نود سالم هنوز عملی باقی بماند.