避難支援 MANET の実機実証実験 —— 実験設計と実装

—— 案 B: セル密度に対する情報拡散の遅延と到達率 ——

多賀 2021 第 4 章の 4 エージェントを実 Embedded Xinu / 実 IBSS 無線へ載せる

概要

多賀祥平 博士論文(2021)が提案する避難支援 MANET を,**実 Embedded Xinu 機(Raspberry Pi)と実 ad-hoc 無線**の上で評価するための実験系を設計・実装した報告である。

出発点は否定的な発見であった。既存の AIPL 実装 `phone_node_fire.abcl` は「そのまま実機へ載せても動く」と自称していたが,**実機には載らない**。AIPL → C コンパイラ `aipl2c` が `web_expose` を無言で消去し,隣接ノードへの MANET ホップ `now remote(...)` を**ローカルのアクタ 0 への呼び出しに黙って誤訳する**こと,Xinu 側に TCP クライアントが存在しない(状態遷移に `SYN_SENT` が無い)こと,アクタランタイムが単一スレッドで同期リモート呼び出しを再入できないこと —— の 3 点が独立に致命的である。

そこで本研究は**同期 RPC を捨て,プロトコルを非同期 UDP へ転換**した。第 4 章の push が返信を必要としない(呼び出し側が戻り値を捨てている)ことを根拠とし,実無線で実績のある `wifi_udp_tx` の上に MANET アプリ層(M14, UDP/5000)を実装した。

測定手法の中核は**時刻同期の除去**である。実機には RTC が無く,`cntpct_el0` はボードごとに自走し,ad-hoc モードでは NTP も機能しない。本研究は受信側の内部所要時間を**差分のまま ACK のペイロードに載せて返す**ことで,発火ノードが自分の時計だけで遅延の内訳を回収できるようにした。また,IBSS のブロードキャストと ACK がともに無確認送信である以上**到達率を ACK で測ってはならない**ことを指摘し,試行後に有線制御面から受信側の共有テーブルを直接問い合わせる厳密判定を採用した。

実装(実機 458 行,ハーネス,ダッシュボード)はビルドを通しているが,**実機による測定は未実施**である。本報告は実験設計と実装の記録であり,結果報告ではない。

1 実験の目的 —— 何を測るべきか

10 機程度の実機で,多賀論文 図 6.2(避難者数 $N = 50$–200,システム利用時の接触回数が非利用時の約半分)を再現することはできない。$10 \ll 80$ だからである。そもそもこの命題は既にシミュレータが示しており,より少ないノードで示し直すことに意味はない。

実機実験の値打ちは,シミュレータが「仮定」した部分を壊しにいくことにある。現行の `manet_fire.abcl` は,接触した通行不能点の情報が**その tick のうちに hop 1 の隣人へ確実に届く**ことを前提としている。無線には遅延も衝突も損失もない,という仮定である。実機はここを壊せる。したがって本実験の主命題は次の一文となる。

> 実無線・実 Xinu の上で,拡散した通行不能点情報は**避難者がその危険点に到達する前に**届くのか。届かなくなるのはセル密度がいくつからか。

評価軸を「接触回数・死亡者数」から**情報が間に合う確率** P_{useful} へ移すのが要点である。間に合わな

ければ use=1 は use=0 へ劣化するので，これは図 6.2 の**前提条件**を検証していることになる。密度は仮想トポロジで作れるため，この問いは手持ち 3 台でも曲線の形が確定する。10 台は曲線を $N = 80$ まで伸ばすためだけに必要である。

2　前提の破綻 —— 既存 AIPL コードは実機に載らない

phone_node_fire.abcl の冒頭コメントは「実機 Embedded-Xinu(rpi3/4/5) と同一の配線なので，この .abcl をそのまま実機へ載せても動く」と述べる。**これは事実ではない。**コンパイラを実際に走らせて確認した。

```
$ aipl2c.exe fire.abcl --xinu-jit --no-typecheck
parse error: Parser.Syntax_error(_, "syntax error in program")
```

配列リテラル [] がパースエラーであり，コード生成に到達すらしない。より深刻なのは，通る部分が**黙って壊れる**ことである（表 1）。

表 1: aipl2c --xinu-jit が AIPL の主要プリミティブに対して行うこと

プリミティブ	Xinu バックエンドでの扱い	出典
class / new / send	実装あり（g_spawn / enqueue）	—
array_len / array_get	実装あり（v_list_*）	—
array_push / reply	**バックエンドに存在しない**	—
web_expose / web_listen	**/* unsupported call */ を吐いて消滅**	c_translator.ml:2418
now remote(host,actor)	**ローカルのアクタ 0 への呼び出しに誤訳**	c_translator.ml:2356
[] リテラル	**パースエラー**	—

最後の行が本研究で最も危険な発見である。c_translator.ml:2356 の

```
| RemoteTarget _ -> "0"
```

により，**隣接ノードへの MANET ホップがローカルのアクタ 0 への呼び出しにコンパイルされる。**警告も例外も出ず，終了コードは 0 で，それらしく動作して誤った結果を出す。プレーン C バックエンドも同様に "-1" を返す（同 :286, :2044）。この沈黙のために，前述の「実機で動く」という記述が何ヶ月も生き延びた。

さらに，仮に now remote を実装しようとしても次の 2 点が独立に阻む。

TCP クライアントの不在 system/tcp_server.c の状態遷移は CLOSED，LISTEN，SYN_RCVD，ESTABLISHED，FIN_ であり **SYN_SENT が無い**（同 :66--73）。受動オープン専用で，構造的に自分から接続を開けない。tcp_connect / SYN_SENT は全リポジトリで 0 ヒットである。

再入不能 アクタランタイムの cc_pump は単一スレッドの協調ループである。同期 now remote は dispatch() の途中でブロックしつつ返信のためにネットワーク I/O を回す必要があるが，継続もブロッキングプリミティブも存在しない。2 ノードが相互に同期呼び出しするとボードが停止する。

直ちに行うべき対策

c_translator.ml:2356 と :2418 を failwith に変更すること（作業量 1 時間）。これが黙っている限り同種の誤りが再生産される。なお :2408 付近のコメントは，この沈黙が過去に一度実害を出した

（GC アクタが自らを保護対象として登録できず自分の sweep で消えた）ことを既に記録している。

3　設計転換 —— 同期 RPC から非同期 UDP へ

プロトコルを UDP へ移せば，実装量は「数ヶ月」から「数週間」になる。そしてこれは妥協ではなく，**より忠実**である。根拠は `phone_node_fire.abcl` 自身にある。

```
// DiffusionAgent.dispatch() --- 140 行目
var _r = now remote(nbr, "diffusion").receive(id);   // 戻り値 _r は捨てられている
...
// DiffusionAgent.receive()
method receive(id) { send info.learn(id); reply(1); }   // この reply(1) は誰も読まない
```

push は返信を必要としていない。同期であるのは `now remote` を使った副作用にすぎない。ブロードキャストへ置き換えれば意味論は同一で，デッドロックも $O(d \times \mathrm{RTT})$ の逐次ブロックループも消える。実際の MANET は隣人に同期 RPC などしない。ブロードキャストする。

送信路は既に存在する。`wifi_udp_tx`（`wifi.c:2374`）は AODV（同 `:2417`）と TFTP が実無線で動作実績を持つ経路である。受信側だけが欠けていた ——`wifi_handle_frame` は ARP / AODV(654) / ICMP / TCP の 4 つしか捌かず，それ以外の UDP は `if (dport != AODV_PORT) return 0;`（同 `:2436`）の後に黙って捨てられていた。

■AODV は使わない。 重複検出キャッシュが 8 エントリしかなく（`wifi.c:2405`），10 機のフラッディングで溢れて再ブロードキャスト storm を起こす。経路表も 16 スロットで，溢れると `g_rt[0]` を無言で踏み潰す（同 `:2401`）。経路の期限切れも RERR も無いため，仮想移動でトポロジを変えても古い経路が残る。ただし**第 4 章・第 6 章のプロトコルは hop 1 の push とゲート pull だけで足り，多段ルーティングを必要としない**。AODV を回避すればこれらの地雷は全て無効化される。

4　実装 —— M14 MANET アプリ層（UDP/5000）

`xinu-rpi5/device/wifi/wifi.c` に M14 節として実装した（API は `include/wifi.h`，シェルは `shell/shell.c`。計 458 行）。受信フックは `wifi_handle_frame` から `aodv_ip_in` の直後に `manet_ip_in` を呼ぶ形で挿入した（`wifi.c:1928`）。

表 2: メッセージ（16 バイト固定ヘッダ，UDP/5000）

type	名称	宛先	役割
1	PUSH	ブロードキャスト	③ 拡散エージェント（hop 1）。返信不要
2	ACK	ユニキャスト	測定用。受信側の内部差分を運ぶ
3	PULLREQ	ブロードキャスト	④ 収集エージェント
4	PULLRSP	ユニキャスト	公開テーブル（② ノード管理）の返答
5	BG	ブロードキャスト	背景負荷。取り込むが ACK しない

■返信は ARP を挟まない。 ACK / PULLRSP は受信フレームの送信元 MAC と IP へ直接ユニキャストする。ARP 解決を経由しないので，**ARP の遅延が測定値に混入しない**。

```
/* manet_ip_in() --- PUSH 受信時 */
if (!mn_nbr_ok(src)) { g_mn_filtered++; return 1; }   /* whitelist = 仮想トポロジ */
mn_learn(fid);                        /* gknown へ取り込み */
```

```
d_ri = SYSTIMER_CLO - t_rx;          /* rx -> 取り込み */
d_ir = mn_reroute();                 /* 取り込み -> 再経路完了 */
mn_mk(a, MN_ACK, seq, 0);
a[6]=(u8)d_ri;  a[7]=(u8)(d_ri>>8);  a[8]=(u8)(d_ri>>16);  a[9]=(u8)(d_ri>>24);
a[10]=(u8)d_ir; a[11]=(u8)(d_ir>>8); a[12]=(u8)(d_ir>>16); a[13]=(u8)(d_ir>>24);
wifi_udp_tx(e + 6, ip + 12, MN_PORT, MN_PORT, a, MN_HDR);   /* src MAC/IP へ直接 */
```

無線層は既存資産をそのまま使う。`wifi_adhoc()`(`wifi.c:2966`)が `wifi_cmd_int(WLC_SET_INFRA, 0)`（同 `:2979`）で IBSS へ切り替え，BSSID を `02:4d:41:4e:45:54`（”MANET”）に固定する（同 `:2987`）ので，全機が決定的に同一セルへ入る。IP は静的に `10.0.0.<n>/24`。

5　測定手法

5.1　時刻同期を要らなくする —— ACK が差分を運ぶ

実機には RTC が無い。時刻は `cntpct_el0` という自走カウンタのみで，原点はボードごとに無関係である。NTP クライアントは実装済みだが実ゲートウェイへの ARP を要し，ad-hoc では `wifi_gw` が架空の `10.0.0.1` に設定されるため**機能しない**。素直に片道遅延を測ることはできない。

本研究は**測定を単一クロックへ畳む**ことでこれを回避した（図 1）。発火ノードは自分の時計だけで RTT を測る。受信側の内部所要時間は**そのボード内で完結する差分**なので同期なしに有効であり，ACK のペイロードに載せて持ち帰らせる。

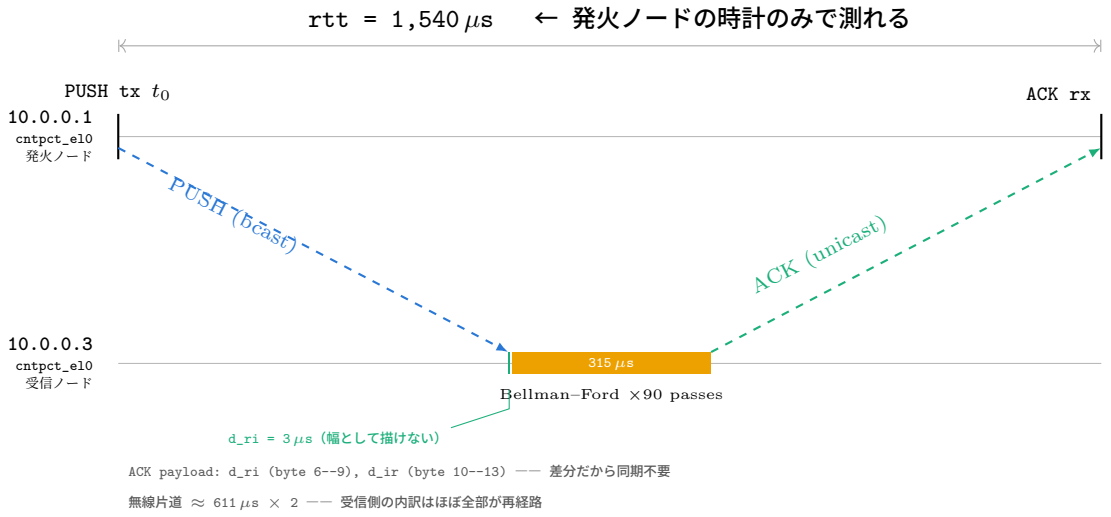

図 1: 2 本の時計。上下の `cntpct_el0` は原点が無関係な自走カウンタであり，RTC も NTP も無い。それでも内訳が取れるのは，受信側の 2 つの差分がそのボード内で完結しているからである。数値は無線モデルによる推定値（測定値ではない）。

5.2　到達率を ACK で測ってはならない

IBSS のブロードキャストは無確認・最低基本レート送信であり，ACK ユニキャストも無確認である。したがって ACK が返らないことは**「PUSH が落ちた」のか「ACK が落ちた」のかを区別しない**。ACK による到達率は下界にすぎない。

本研究は ACK を**遅延の測定にのみ**用い，到達率は試行後に**有線制御面から** `manet has <fid>` を発行して受信側の共有テーブルを直接読むことで厳密に判定する。ダッシュボードは「厳密」と「ACK

観測」の両者を併記するので，**その差がそのまま ACK 自身の消失量**として読める。

5.3 変数と対照群

表 3: 実験変数

独立変数	offered load $L \in \{0, 4, 16, 64, 128, 256\}$ [pps]（密度 N のセルは各ノード毎 tick 1 発なので $L \approx N$）
主従属変数	**到達率** $r(L)$（厳密判定）
従属変数	$t_{\mathrm{all}}(L)$ の分位数（p50 / p95 / max），内訳 d_{ri} / d_{ir}
導出量	$P_{\mathrm{useful}}(L) = r(L) \cdot \Pr[t_j < T_{\mathrm{avail}}]$
試行	各 L につき 30 回以上。発火ノードをラウンドロビン，L **の順序はランダム化**（熱ドリフト交絡の回避）

表 4: 対照群 —— これが無いと何を測ったか言えない

対照	何を分離するか
`--ackonly`	受信処理を省き RX 直後に ACK ⇒ 無線・輻輳をエージェント処理から分離
`--rrp 0`	再経路を切る ⇒ プロトコル素の遅延
`--load 0`	静穏ベースライン
ACK 有無の比較	ACK 自身（d パケット）が媒体へ与える摂動の上界

5.4 猶予時間と P_{useful}

猶予 T_{avail} は，隣人 j がその危険点 v へ到達するまでの徒歩時間 $\mathrm{dist}(j, v)/1.4\,[\mathrm{m/s}]$ である。通信半径 50 m を上限とすれば $T_{\mathrm{avail}} \approx 36\,\mathrm{s}$。有用性の判定は $P_{\mathrm{useful}}(L) = \Pr[t_j < T_{\mathrm{avail}}]$ であり，これが本実験の主図となる。

6 実験手順

制御面は**有線 Ethernet** 上の HTTP（`/run?cmd=<shell>`）とし，無線を汚さない。データ面のみが実 IBSS を通る。1 試行は次の通り。

1. 各ボードへ `manet node <n>` / `manet cfg <proxy> <rrp> <rrv> <ackonly>` / `manet reset`
2. 背景負荷を peers 上で非同期起動：`manet load <pps> <ms>`（負荷 L を peers で山分け）
3. 発火：`manet fire <fid> <timeout_ms>` ⇒ `@B ack` / `@B done` 行を回収
4. 厳密到達率：各 peer へ `manet has <fid>` ⇒ `@B has node=N fid=F v=0|1`

制約として，HTTP `/run` の応答本文は **640 バイト**（`tcp_server.c:533`），コマンドは 128 バイトが上限である。共有テーブル全体の取得は切り捨てられるため，到達判定に `manet has`（1 行）を用いる設計とした。

7　結果 —— 未測定である

実機による測定は行っていない。実機 3 台（Pi5 = `10.0.0.3` / Pi4 = `10.0.0.1` / Pi3 = `10.0.0.2`）は本報告の執筆時点で電源が落ちている。実装とハーネスはビルドを通しており，シンボルはカーネルへリンク済みであることを確認したが，無線上での疎通は未検証である。

本節に測定値は無い。代わりに，実装の計算量から導かれる**予測**を記す。

遅延では有意差が出ない見込み PUSH は 16 バイトヘッダであり，$L = 64$ でも媒体負荷は $64{\times}200\,\mathrm{B} \approx 13\,\mathrm{kB/s}$ にすぎない。802.11 はこれをミリ秒で捌く。猶予 36 s に対して遅延はミリ秒桁であり，$P_{\mathrm{useful}} \approx r$ となる。すなわち**律速は遅延ではなく損失**である。主従属変数を $r(L)$ に置いた理由はここにある。

損失が効く IBSS のブロードキャストは無確認かつ最低基本レート送信なので再送が無い。素の損失が L とともに増える。

受信側の内訳は再経路がほぼ全部 取り込み（`mn_learn` の線形走査）は数 μs，再経路（Bellman–Ford, 90 passes × 327 辺 ≈ 29k 緩和）は約 300 μs と見積もられる。ただし両者を合わせても無線往復（≈ 611 μs × 2）には及ばない。

■現行ノード実装の忠実性について。 `phone_node_fire.abcl` の `InfoAgent` はコメント上「避難経路の再構成」を担うと述べるが，**コードにその実装は無い**。再経路（距離場の再計算）は世界側の `manet_fire.abcl` にある。論文の情報エージェントは端末上で再経路するので，**再経路をノードへ移すことが忠実であり，かつ密度効果が現れる場所**である。M14 はこれを `manet cfg <proxy> <rrp> <rrv>` で切り替え可能な合成負荷として持つ。

8　限界 —— 報告時に必ず明記すべきこと

1. **1 部屋に置くと物理的には単一衝突ドメインである。**アプリ側で whitelist を掛けても CSMA は全機が全送信を聞く。荒川区で 500 m 離れた端末は競合しないのに，部屋の中では競合する。したがって本実験は**輻輳を過大評価する悲観側の上界**を測っている。欠陥ではなく，そう報告すれば正当な最悪ケース評価となる。
2. **背景負荷はパケットレートの近似であって N 台のセルではない。**1 つの無線が N ノード分を出しても，CSMA の backoff は無線ごとなので**独立に競合する送信機の数**は再現できない。
3. **無線伝搬モデルは検証できない。**仮想トポロジを被せた時点で，50 m ユニットディスク仮定はシミュレータから実機へ移動しただけである。ここを主張してはならない。実機が真に足しているのは「実 Xinu 上での実行」と「実パケットの衝突・損失・遅延」の 2 点のみである。
4. **再経路は Bellman–Ford のコストモデルである。**passes × edges の二重ループという計算構造を合わせた合成負荷であり，実在の荒川 266 頂点地図ではない。測っているのは経路の正しさではなく計算コストである。

9　論文の主張への接続

本実験の成果は，それ単体では「実機のリンク特性」にとどまる。論文の主張へ接続する手順は次の通りである。

1. 実測した $P_{\mathrm{useful}}(L)$ と $t_j(L)$ を `manet_fire.abcl` へ戻す。
2. 現行の「push は必ず即時に届く」を「確率 $P_{\mathrm{useful}}(L)$ で遅延 $t_j(L)$ の後に届く」へ置き換える。
3. 図 6.2（$N = 50$–200，各 50 回）を再実行する。

use $<$ not-use が生き残れば**実無線条件下で論文の主張が検証された**ことになり，高密度で崩れれば**実機でしか見つからない限界を発見した**ことになる。どちらでも結果である。これが「有用性の評価」に対する本研究の答えである。

10　今後 —— 着手順

1. `c_translator.ml:2356` / `:2418` を `failwith` へ（1 時間）
2. 実機 3 台を起動し，**M14 の疎通確認**（A がブロードキャスト → B が受信 → ACK が返る）—— ここが全ての土台
3. 3 台で $L = 0, 4, 16, 64$ の $r(L)$ と $t_{\mathrm{all}}(L)$ を取得 —— ここで本実験の図が初めて出る
4. 7 台調達 → L を伸ばす → $P_{\mathrm{useful}}(L)$ → シミュレータへ差し戻して図 6.2 を再実行
5.（案 A）ハイブリッド HIL：実機 10 + 仮想 70。実機が仮想の群衆の中に埋め込まれた観測点となる

成果物：実機実装 `xinu-rpi5/device/wifi/wifi.c`（M14 節），`include/wifi.h`，`shell/shell.c`。ハーネス `drone-taga/expB/bench_b.py`（ドライバ・配信），`dash.html`（可視化），`README.md`。処理の可視化（Web）：https://claude.ai/code/artifact/eb0853a5-416a-4cc6-b2aa-123ffbba0580