情報指向ネットワークのプログラマビリティを活用したアプリケーション開発

電子情報通信学会 ICN 研究会　2019.5

大阪市立大学　大学院工学研究科

阿多　信吾
ata@eng.osaka-cu.ac.jp

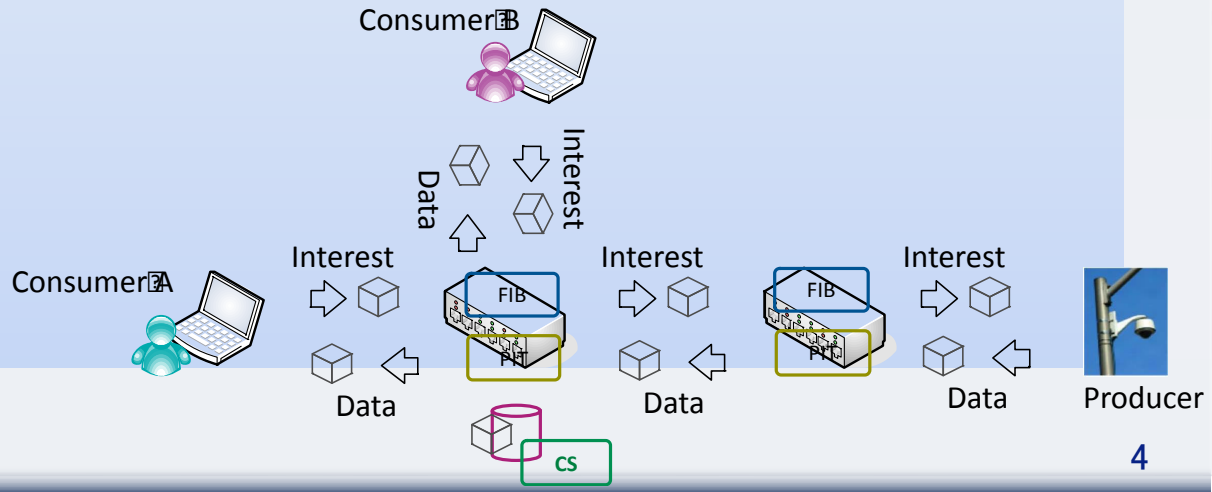
本研究は NICT 委託研究 (#184, ICN2020) の成果の一部です

情報指向ネットワーク概要

- 名前によるアドレッシング
  - IP アドレスではなく「名前」をパケットの宛先に
  - 階層型名前構造
    - /building/floor/room/temperature
- 名前によるルーティング・フォワーディング
  - 順方向：FIB (Forwarding Information Base)
  - 逆方向：PIT (Pending Interest Table)
- コンテンツストアによるネットワーク内キャッシュ
  - CS (Content Store) によるパケット単位のキャッシュ

ICN の基本的な通信プロトコル

- Interest/Data 交換
  - Consumer が Interest を送信
  - Forwarder が FIB を参照してルーティング
  - Producer が Data を返送（PIT を用いてルーティング）
- CS によるコンテンツキャッシュ
  - 別の Consumer が Interest を送信した場合は CS から Data を返送

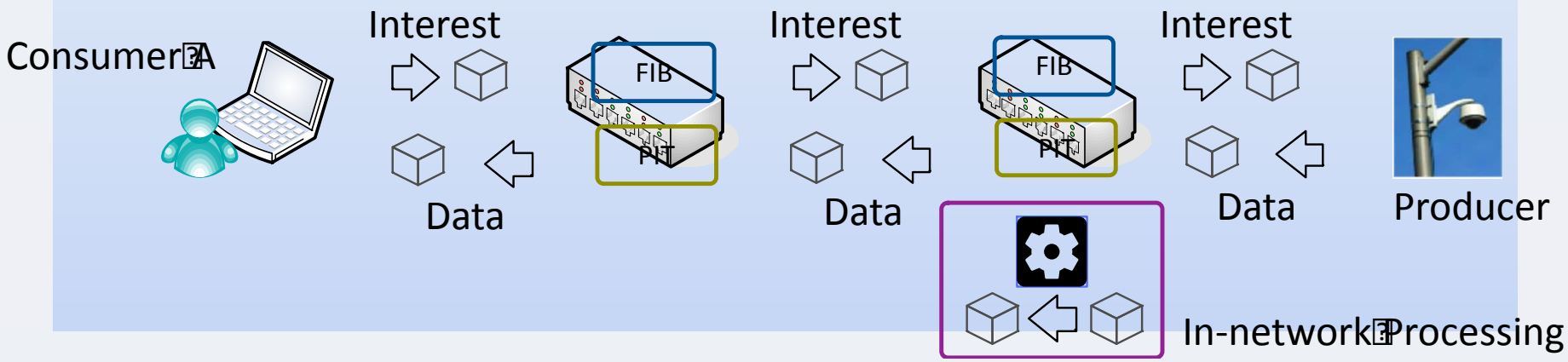


情報指向ネットワークのメリット

- 名前解決ノードとの不整合性の解消
  - ネットワークレイヤでコンテンツルーティングを行うことで上位層でのコンテンツ管理が不要に
- ステートレス
  - コンテンツごとに（セグメント位置も含めた）独立したパケットの送受信
- セキュリティ
  - コンテンツ自体への署名、暗号化
- コンテンツキャッシュによる効果
  - コンテンツ取得性能の向上
  - トラヒックエンジニアリングの柔軟化
  - 耐障害性の向上
  - コンテンツモビリティのサポート

ネットワーク内 (in-network) XXX

- 当初は静的コンテンツを対象
  - ネットワーク内キャッシュで資源有効利用と性能向上
- ネットワーク内で処理できるのはキャッシュだけではないはず
  - CPU 資源を使ってもっと拡張性のあることができないか
  - CPU だけではなく他の処理(制御)にも使えないか？
  - ノードに拡張性を持たせて一括した名前で呼べないか？



ICN によるネットワーク層プログラマビリティ

- 「名前」だけでどれだけプログラマビリティが実現できるのか？
  - プログラマビリティは今後重要
  - 特に IoT, M2M デバイス
    - デバイスのプログラマビリティは期待できない
    - センシングだけでなくアクチュエーションをどう実現するか
- ICN はプログラマビリティの可能性を秘めている（はず！）
  - 可読性を有する「名前」
  - 名前構造の柔軟性
  - 名前ネイティブなルーティング
  - In-network processing によるエッジコンピューティング

ファンクションモビリティとエッジコンピューティング

- コンテンツキャッシュができればファンクションキャッシュもできるはず
  - 必要な場所に必要な量だけ「処理関数」を設置
  - 人気のある処理、過負荷な処理は他ノードに分散
  - オンデマンド、動的に処理関数を配置して実行
- ネットワーク内処理とコンテンツ（関数）キャッシュの組み合わせ
  - ICN によるエッジコンピューティングの実現

研究の目的と位置づけ

- プログラマブルネットワークとしての情報指向ネットワーク
  - ICN as a Programming Language
  - Interest-Data 交換 ⇔ API 呼出 による完全分散処理
  - 「名前」を使った柔軟なパケット処理
    - 直感的なコンテンツ取得
    - 動的なコンテンツ取得、デバイスコントロール
  - In-network processing による情報処理
    - 取得したデータの加工・集計
    - 動的な処理の追加・配置
  - ネットワーク層による最適化
    - コンテンツだけでなくファンクションも
- プログラマブルフレームワークの有用性？
  - アプリケーション（ユースケース）を考える

課題

- ネットワーク層でいろいろできる
  - 直感的な Interest-Data のやりとり
  - 確かに…
- プログラマビリティ？
  - 依然として「ICN を使う」ことのお作法が必要
  - アプリケーション開発者にとってのメリットは？
    - 学習コストが低い
    - コード行が少ない
  - 他のフレームワークに勝てる？？
    - 例えば WebAPI、REST

ICN 研究者（主としてネットワーク分野）とプログラム開発者の
ギャップをどうやって埋めるか

今回の目標

- アプリケーションの ICN 対応
  - 可能な限り小さい開発・学習コストで実現
  - ICN のメリットをより簡単に享受
- Python による簡便な ICN サポートの実現手法の提示
  - よりシンプルに ICN を利用するためのサポートフレームワーク（ほど大げさではない）の設計と実装
  - 実現例
  - 考察（メリットと今後の課題を中心に）

今回の目標（より具体的に）

◻ サンプルプログラム

```
from time import sleep
import random

import add

while True:
    parm1 = random.randrange(10)
    parm2 = random.randrange(10)
    ret = add.add(parm1, parm2)
    print(ret)
    sleep(1)

```

```
def add(parm1, parm2):
    return int(parm1) + int(parm2)

```

add モジュールを ICN で遠隔実行

```
from time import sleep
import random
import icnsupport

import add

while True:
    parm1 = random.randrange(10)
    parm2 = random.randrange(10)
    ret = add.add(parm1, parm2)
    print(ret)
    sleep(1)

```

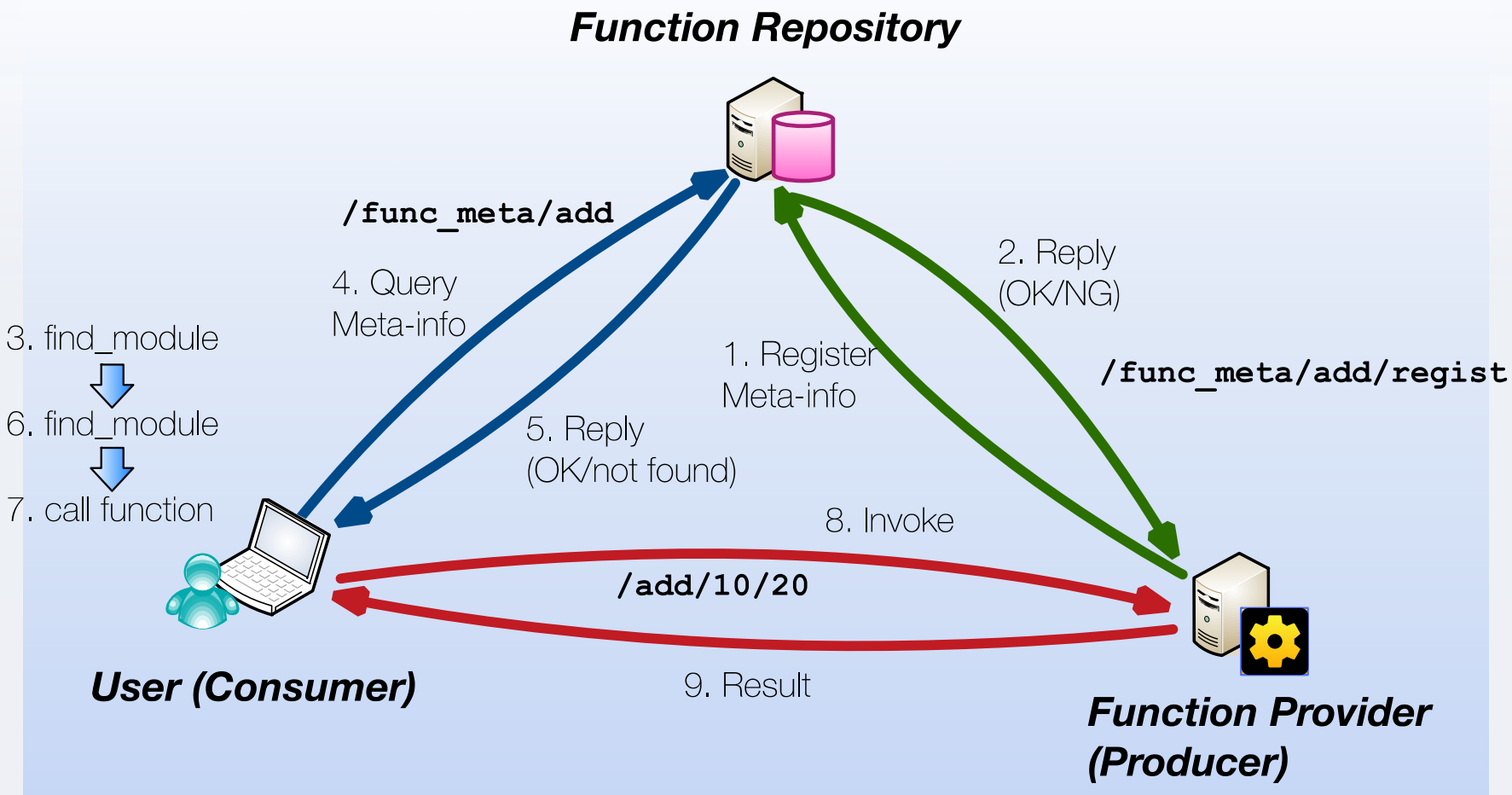
大まかな方針

- モジュールをインポートするタイミング
  - そのモジュールが ICN で提供可能か？
    - ICN でモジュールが存在するかを確認
  - 可能であれば ICN の API 呼び出し関数でラップ
    - 関数名による Interest-Data 交換
  - モジュールの読み込み
  - ICN に存在しないモジュールは通常のファイル読み込みよるインポート
- Python による動的モジュールインポート機構を利用
  - インポートフック
- ICN 部分は Cefore/cefpyco を利用
  - Cefore: NICT による ICN 実装（cefore.net）
  - cefpyco: Cefore の python ラッパ

インポートフック

- 詳細は Python マニュアルを参照
  - sys.meta_path にモジュール検索順を格納
    - sh におけるコマンドパスのようなもの
  - モジュール検索とモジュール読み込みのクラスを定義
    - インターフェース規定は importlib.abc を参照
    - モジュール検索クラス
      - find_module(self, fullname, path)
    - モジュール読み込みクラス
      - load_module(self, fullname)

全体の動き


Function Repository
/func_meta/add
4. Query
Meta-info
2. Reply
(OK/NG)
3. find_module
6. find_module
7. call function
1. Register
Meta-info
/func_meta/add/regist
5. Reply
(OK/not found)
8. Invoke
/add/10/20
9. Result
User (Consumer)
Function Provider
(Producer)

名前スキーム

- Function Repository
  - メタ情報の登録
    - /func_meta/function/regist
  - メタ情報の取得
    - /func_meta/function
- Function Provider
  - 関数呼び出し　function(param1, param2, …)
    - /function/param1/param2/…

関数シリアライズと呼び出し

- 手順
  - 関数名、引数のリストを / で連結した名前を作成
    - 引数リストを文字列配列に変換
  - 作成した名前で Interest を送信
  - 受信 Data のペイロードを返す

```
def invoke_icn(funcname, args = None):
  try:
    name = icn_prefix + funcname
    if (args):
      parms = '/'.join([str(n) for n in args])
      name += '/' + parms
    cefpyco_h.send_interest(name, 0)
    info = cefpyco_h.receive()
    if info.is_succeeded and info.name == name:
      return info.payload
  except Exception as e:
      return None
```

モジュールの検索と読み込み【抜粋】

- モジュール検索（fullname: import xxx の部分）

```
class IcnModuleFinder:
  def find_module(self, fullname, path = None):
    name = "func_meta/" + fullname
    if not invoke_icn(name) == "OK":
      return IcnModuleLoader()
    return None
```

- モジュール読み込み
  - setattr は文字列でクラスメンバへの値をセット

```
class IcnModuleLoader:
  def load_module(self, fullname):
    m = imp.new_module(fullname)
    m.__loader__ = self
    setattr(m, fullname,
      (lambda *args: invoke_icn(fullname, args)))
    sys.modules.setdefault(fullname, m)
    return m
```

Producer におけるモジュール読込みと関数呼出し

- 手順
  - 登録フェーズ（レポジトリへの登録と Interest 受信のための経路情報登録）
  - 実行フェーズ（Interest→関数呼び出し→結果を Data 返送）

```
module_name = sys.argv[1]
module = import_module(module_name)
icn_name = icn_prefix + module_name
with cefpyco.create_handle() as handle:
  handle.register(icn_name)
  while True:
    info = handle.receive()
    if (not info.name.startswith(icn_name)):
      continue
    parms = info.name.replace(icn_prefix, '').split('/')
    funcname = 'module.' + parms[0]
    ret = (eval(funcname))(*parms[1:])
    if info.is_succeeded and (info.name.startswith(icn_name)):
      handle.send_data(info.name, str(ret), info.chunk_num)
```

動作環境の作成

- 動作環境
  - Docker (compose) により作成
    - consumer, producer それぞれ1ノード
    - レポジトリは producer 上で実行
  - Cefore 0.8.1/cefpyco 0.3.7
  - 開発プログラム
    - icnsupport モジュール
    - meta_repo.py: レポジトリ
    - producer.py: producer アプリ

検証

- consumer, producer に同じモジュールを配置
  - 同じモジュール動作をさせるため
- ICN サポートが動的に切り替えられていることを確認
  - アプリに import icnsupport を入れる・入れない
  - レポジトリが応答しない（NG を返す）
- 動作デモ

定量的？な考察

- アプリ開発者：モジュールの ICN 化
  - 学習コスト、開発コスト
  - アプリ側はほぼコストなし（import icnsupport を挿入するだけ）
  - producer 側はモジュールを転送し専用アプリで起動すれば OK
- 仕掛けの開発者（私）
  - Python 開発経験：のべ50時間未満
  - プログラムコード数（デバッグコード含む）：150行程度
  - 開発時間：5時間くらい

メリット

- 一番は「簡単に ICN 対応できる！」
  - アプリ開発者はたった1行
  - 中身を考える必要がない
- モジュールの配置等については ICN が対応
  - ICN はもともとノード位置に依存しない
  - 負荷が高ければ producer の複製を分散配置
  - コンテンツキャッシュ・負荷分散アルゴリズムが適用可能
  - 同じ関数の呼び出しはコンテンツキャッシュにより答えを再利用（したくなければ切ることも可能）
- ICN ネットワーク制御による最適化
  - オーケストレータが不要
  - サービスにおける C/D 分離

現時点の制約

- icnsupport モジュールの制約
  - いまのところ1モジュール1関数
  - 型変換のチェックなし
    - メタ情報に型情報を追加すれば対応化
    - パラメータのシリアライゼーション
  - 非同期処理化
    - 長い処理、ネットワーク遅延などによりアプリがブロッキング
    - イベント通知（Push で関数起動など）
  - サイズの大きなデータ転送（マルチチャンク）
- Python 以外の適用の可能性
  - 言語の制約によってはそこまで簡単にならないかも

まとめと今後の課題

- まとめ
  - ICN のプログラマビリティをより広く適用したい
    - 既存のプログラムを簡単に ICN 化
    - 分散コンピューティング環境を構築可能
    - ICN の知見を活用すればモジュール制御をアプリが考える必要なし→C/D 分離が可能
- 今後の課題
  - 制約を解消するための改良
  - プログラムの公開