

情報指向ネットワークにおけるプログラマビリティ拡張のプロトタイプ作製

電子情報通信学会 ICN 研究会 2019.12

山澤 慶太*, 浅井 貴大*, 阿多 信吾**

* 大阪市立大学 工学部 電気情報工学科

** 大阪市立大学 大学院工学研究科 電子情報系専攻

研究背景

- ICNの対象
 - ストリーミングのような動的コンテンツにも適用
 - データ加工処理の要求といった適用も今後必要に
- ICNの持つプログラマビリティを活用
 - 静的コンテンツを要求するのと同じようにして、計算(ファンクション)を実行
 - 小さな処理を組み合わせて複雑で大規模な処理
- IoTとの組み合わせ
 - IoTデバイスの貧弱な計算能力をネットワークが補う
- 目標:ICN上でどれほどプログラマビリティを実現できるか?

目的

- 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の実現
 - API 呼び出し ⇔ Interest / Data 通信
 - 動的なコンテンツ取得
 - 計算資源の乏しいデバイスからも実行可能
 - ネットワーク層による最適化
 - アプリの開発者は下階層にICNを使うだけで恩恵
 - 分散型のアーキテクチャ
 - ノード数を増やすだけで水平スケーリング
 - サーバレスなアプリケーション
 - ネットワークに流せば自動で冗長化 耐障害性◎

課題

- ユーザビリティ
 - ICNの専門知識が必要
 - 学習コストの高さ
 - アプリケーション開発者への負担
 - コード記述量の増加
- ファンクションモビリティ
 - キャッシュできるのは実行結果のコンテンツのみ
 - 中間ノードの計算資源を利用できない
 - 必要なのはメカニズムのキャッシュ

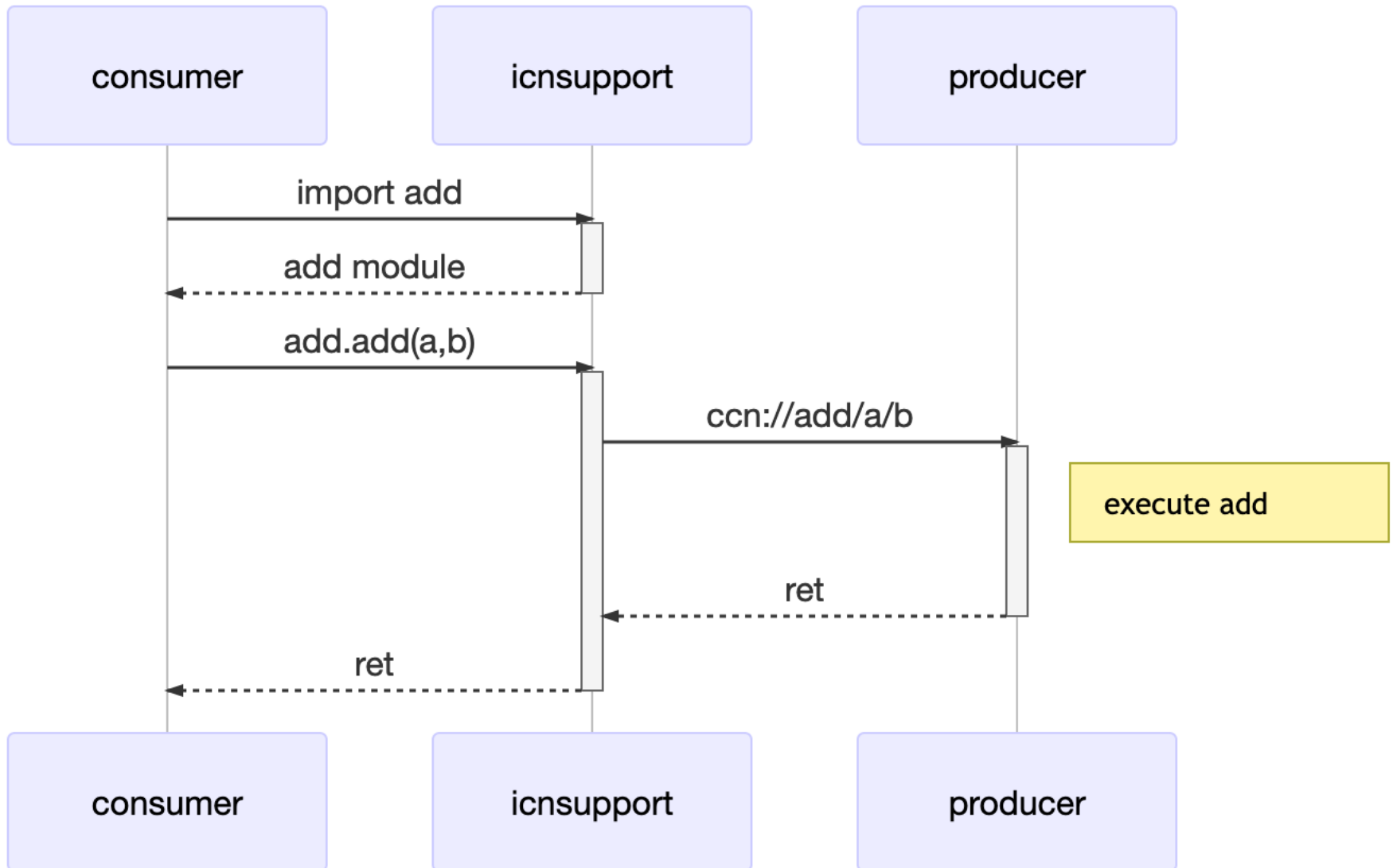
解決法

- Pythonアプリケーションにおける簡便なICNサポートの実現手法
 - ユーザビリティの課題を解決
 - Pythonの関数をファンクションとして使用
 - 将来的には全てのモジュールを使用可能に
- ファンクションキャッシュを行うアーキテクチャの開発
 - ファンクションモビリティの課題を解決
 - 中間ノードでファンクションのキャッシュと実行
 - 分散コンピューティング、エッジコンピューティングの実現

これまでの研究

- Pythonアプリケーションにおける簡便なICNサポートの実現手法を提示
 - “import icnsupport”の一行を挿入する
 - 以降, モジュールがインポートされた際に, ICNで利用できるかを調べ, 可能なら自動的にICN呼び出しに
 - インポートフックを利用
 - 開発したプログラム
 - consumer.py (アプリケーション)
 - icnsupport モジュール
 - producer.py
 - 具体例の紹介
 - add モジュールを例

これまでの研究



これまでの研究

consumer.py

```
1 from time import time
2 import random
3
4 import add
5
6 param1 = random.randrange(10)
7 param2 = random.randrange(10)
8 ret = add.add(param1, param2)
9 print(ret)
```



consumer.py

```
1 from time import time
2 import random
3
4 import icnsupport
5 import add
6
7 param1 = random.randrange(10)
8 param2 = random.randrange(10)
9 ret = add.add(param1, param2)
10 print(ret)
```

add.py

```
1 def add(param1,param2):
2     return int(param1)+int(param2)
3
```

```
$ python producer.py add.py
```

```
$ python consumer.py
```


icnsupportの目標

- アプリケーション上で、全てのモジュールのメソッドをICN呼び出し可能にする
 - まだまだ利用範囲が限られている
 - 特に改善すべき制約
 - 単一モジュールから単一関数
 - モジュールとメソッドが同名のものしか不可能
 - `add.py` から `add` メソッドしか呼び出せない
 - 型情報がない
 - `str` 型のみしか送受信できない
 - パケットサイズを上回るデータの送受信

具体的な目標

□ consumer.py

```
1 import random
2 import icnsupport
3
4 import calc
5 import make_list
6
7 param1 = random.randrange(10)
8 param2 = random.randrange(10)
9 ret_add = calc.add(param1, param2)
10 ret_sub = calc.sub(param1, param2)
11 ret_list = make_list.square()
12 print(ret_add, ret_sub)
13 print(type(ret_add))
14 print(len(ret_list))
15 print(type(ret_list))
```

□ calc.py

```
1 def add(param1, param2):
2     ret = int(param1)+int(param2)
3     return ret
4
5 def sub(param1, param2):
6     ret = int(param1)-int(param2)
7     return ret
```

□ make_list.py

```
1 def square():
2     ret = [i**2 for i in range(10000)]
3     return ret
```

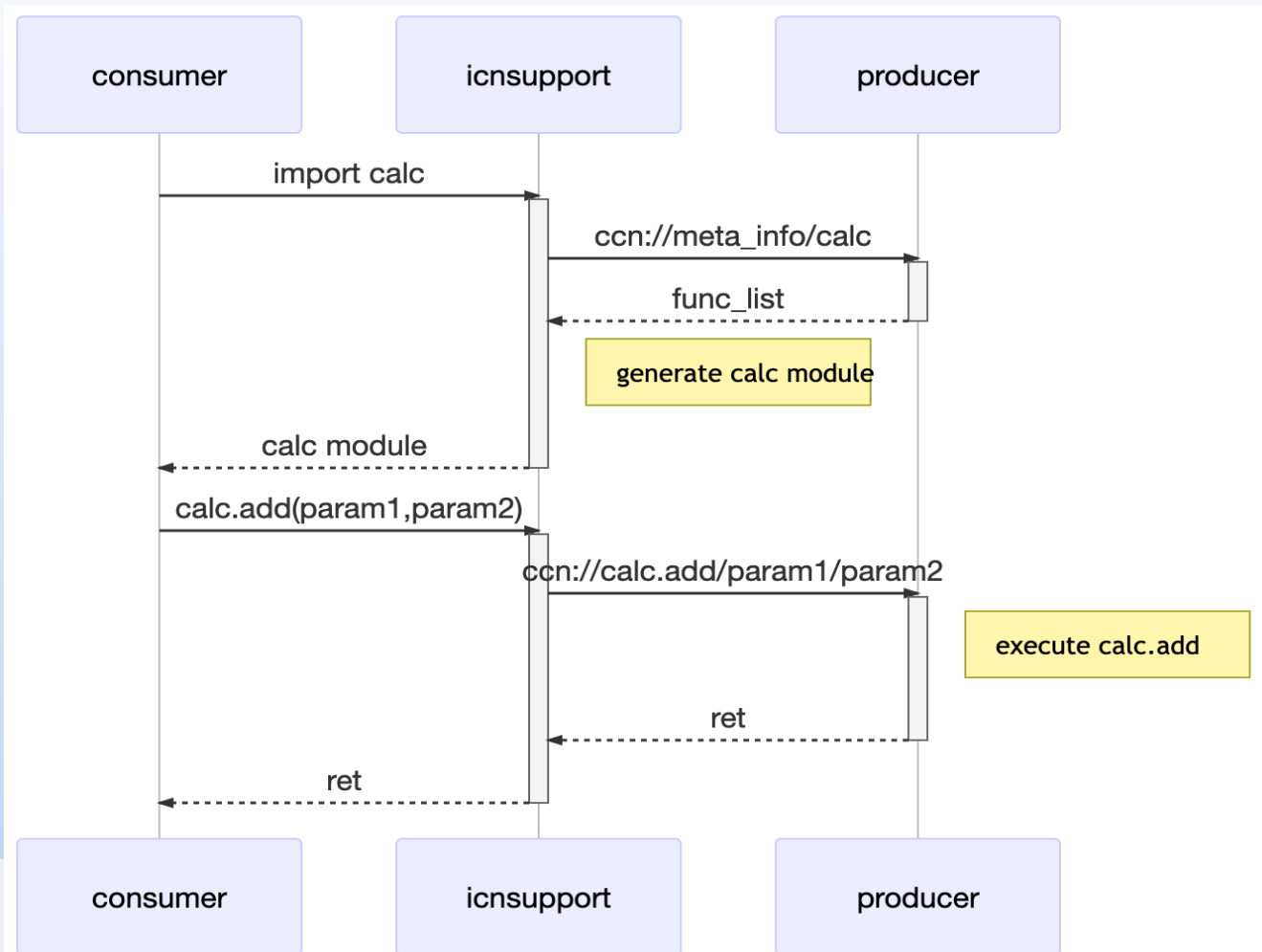
単一モジュールから単一関数の制約

- モジュールと関数が同一名のものに限られる
 - インポートフックで得られる情報はモジュール名のみ
 - そのモジュール内に、どのようなメソッドが存在するかが分からない
- icnsupport内の対応すべきコード

```
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
```

単一モジュールから単一関数の制約

consumerへレジストされた関数名の通知



単一モジュールから単一関数の制約

□ モジュール生成部分のコード

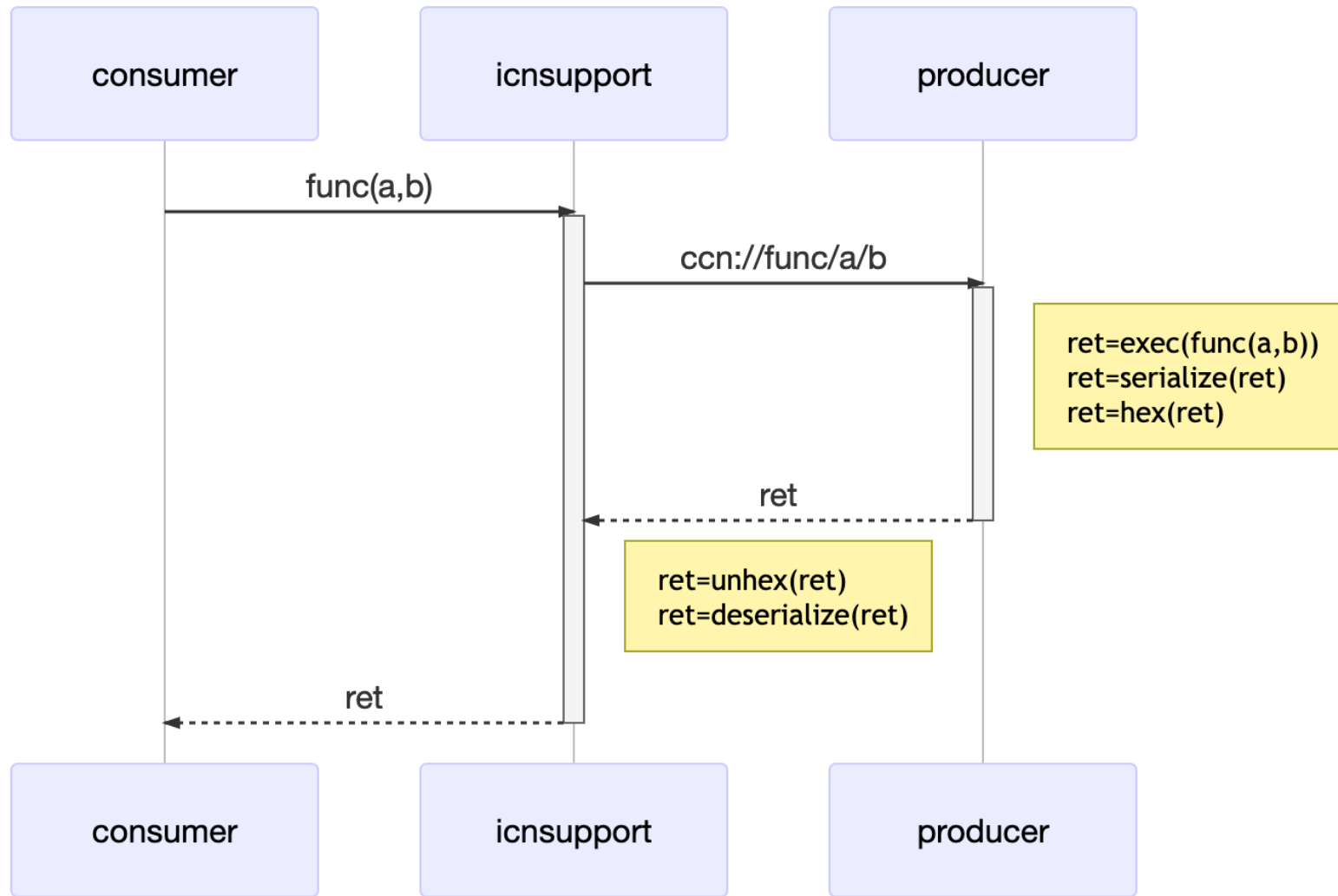
```
class IcnModuleLoader():
    def __init__(self, func_list, handle):
        self.func_list = func_list
        self.handle = handle
    def load_module(self, fullname):
        module = imp.new_module(fullname)
        module.__loader__ = self
        for attr in self.func_list:
            funcname = fullname+'.'+attr
            setattr(module, attr, lambda *args, name=funcname:
invoke_icn(name, args))
        sys.modules.setdefault(fullname, module)
        return 0
```

- consumer.py内で例えば, calc.add()が呼び出されたら,
ccn://calc.addへのICN呼び出しへ

戻り値の型に対する対応

- 元々送受信可能なのはstr型
 - データの型が分からない
 - データをシリアライズすることで、型情報も合わせて送信する
 - 解決方法
 - producer側でデータをシリアライズ
 - そのバイナリを16進数文字列に変換して送信
 - icnsupport側で16進数文字列をバイナリに変換
 - バイナリをデシリアライズする
 - シリアライズにはpickleモジュールを利用
 - base64はbytes型のため16進数文字列に変換

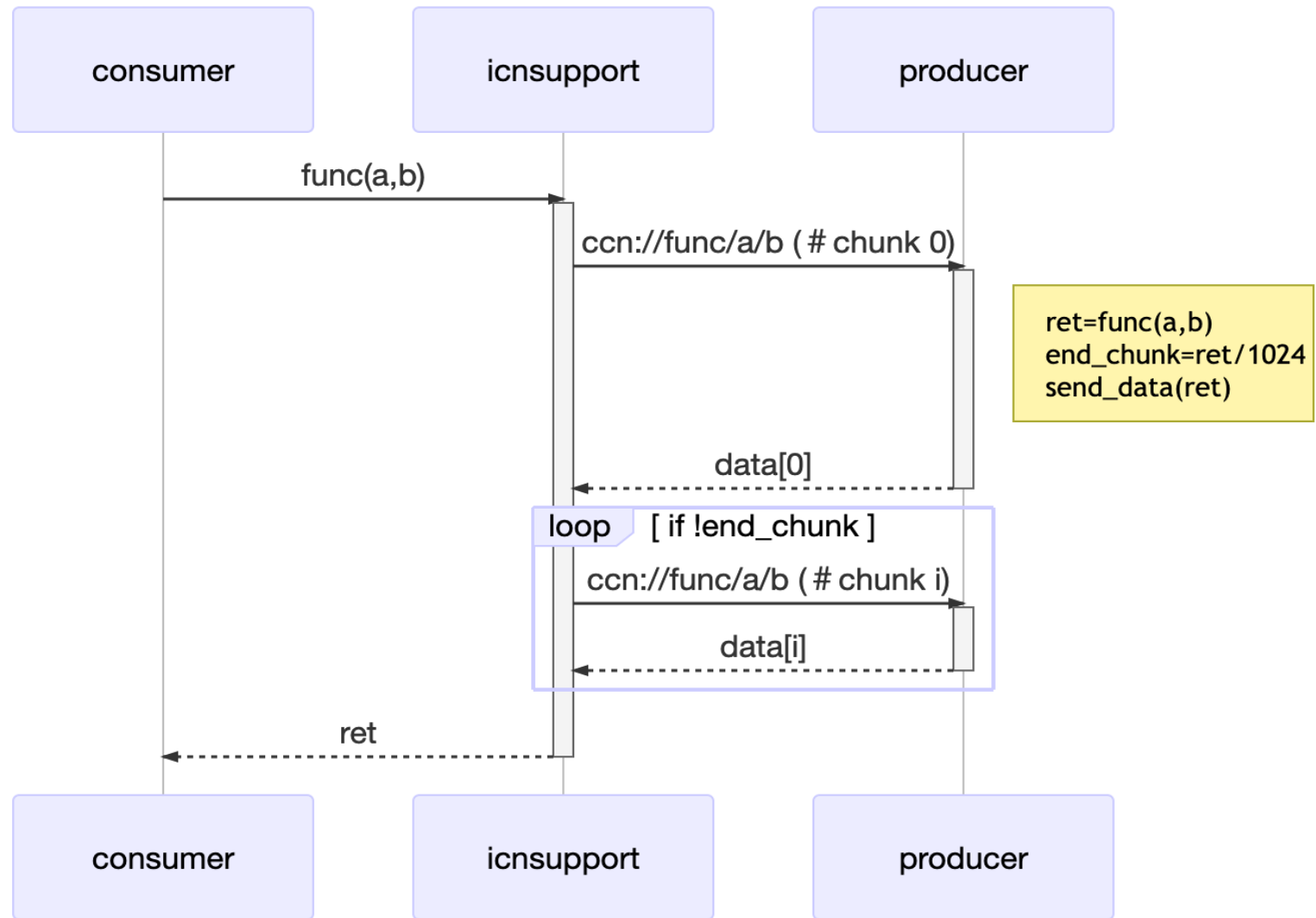
戻り値の型に対する対応



パケットサイズを上回るデータの送受信

- サイズが大きいデータを送受信
 - icnsupportとproducerで自動的に行わせる
- Dataのチャンク数
 - producer側は?
 - send_data()の第四引数で指定可能
 - icnsupport側は?
 - send_data()の第四引数の値を,
CcnPacketInfo.end_chunkから得られる

パケットサイズを上回るデータの送受信



パケットサイズを上回るデータの送受信

□ producerの一部抜粋

```
ret = exec_func(info, modules, func_list)
serialized_ret = pickle.dumps(ret)
data = divide_data(serialized_ret)
send_data(handle, info, data)
```

□ データ送信部分

```
while True:
    if info.is_succeeded and info.chunk_num == i:
        handle.send_data(info.name, data[i].hex(), i, len(data))
        i += 1
    if i >= len(data):
        break
    info = handle.receive()
```

パケットサイズを上回るデータの送受信

□ icnsupportの一部抜粋(受信部分)

```
data = bytes("").encode()  
while True:  
    handle.send_interest(name, chunk_num)  
    info = handle.receive()  
    if info.is_succeeded and info.name == name:  
        data += bytes.fromhex(info.payload)  
        chunk_num += 1  
        if chunk_num == info.end_chunk_num:  
            break  
return data
```

□ consumerにreturnする部分

```
data = send_interest(name, args)  
ret = pickle.loads(data)  
return ret
```

環境構築

□ 動作環境

□ Dockerのコンテナ

□ consumer,producerは1ノードずつ

□ Cefore 0.8.1/cefpyco 0.3.7

□ Python 3.7.4

□ 実行プログラム

□ consumer.py(アプリケーション)

□ icnsupport(consumer上のモジュール)

□ producer.py

□ calc.py, make_list.py(producer上で動作)

動作検証

- `consumer.py`の実行
 - 複数モジュール, 複数メソッドが呼び出されるか
 - `calc.py`, `make_list.py`モジュールをimport
 - `calc.add()`, `calc.sub()`, `make_list.square()`で確認
 - `str`以外の型を受け取れるか
 - ビルトインの`type()`関数で確認
 - データサイズが大きいものが送受信できるか
 - 返されるリストの長さは $4 * 10000$ (byte)
 - 自動的にマルチチャンクになる

デモ

```
022ab054a716eb054aa4baab054ad906ac054a1053ac054a499fac054a84ebac054ac137ad054a0084
a41d0ad054a841cae054ac968ae054a10b5ae054a5901af054aa44daf054af199af054a40eaf054a91
54ae47eb0054a39cbb0054a9017b1054ae963b1054a44b0b1054aa1fcb1054a0049b2054a6195b2054a
2054a292eb3054a907ab3054af9c6b3054a6413b4054ad15fb4054a40acb4054ab1f8b4054a2445b505
1b5054a10deb5054a892ab6054a0477b6054a81c3b6054a0010b7054a815cb7054a04a9b7054a89f5b7
042b8054a998eb8054a24dbb8054ab127b9054a4074b9054ad1c0b9054a640dba054af959ba054a90a6
a29f3ba054ac43fbb054a618cbb054a00d9bb054aa125cb054a4472cb054ae9beb054a900bbd054a39
54ae4a4bd054a91f1bd054a403ebe054af18abe054aa4d7be054a5924bf054a1071bf054ac9bdf054a
0054a4157c0054a00a4c0054ac1f0c0054a843dc1054a498ac1054a10d7c1054ad923c2054aa470c205
dc2054a400ac3054a1157c3054ae4a3c3054ab9f0c3054a903dc4054a698ac4054a44d7c4054a2124c5
071c5054ae1bdc5054ac40ac6054aa957c6054a90a4c6054a79f1c6054a643ec7054a518bc7054a40d8
a3125c8054a2472c8054a19bfc8054a100cc9054a0959c9054a04a6c9054a01f3c9054a0040ca054a01
54a04daca054a0927cb054a1074cb054a19c1cb054a240ecc054a315bcc054a40a8cc054a51f5cc054a
d054a798fcd054a90dccc054aa929ce054ac476ce054ae1c3ce054a0011cf054a215ecf054a44abc05
8cf054a9045d0054ac4fd0054a112dd1054a407ad1054a71c7d1054aa414d2054ad961d2
0afd2054a49fcd2054a8449d3054ac196d3054a00e4d3054a4131d4054a847ed4054ac9c9cb4054a1019
a5966d5054aa4b3d5054af100d6054a404ed6054a919bd6054ae4e8d6054a3936d7054a9083d7054ae9
54a441ed8054aa16bd8054a00b9d8054a6106d9054ac453d9054a29a1d9054a90eed9054af93bda054a
a054ad1d6da054a4024db054ab171db054a24bfdb054a990cdc054a105adc054a89a7dc054a04f5dc05
2dd054a0090dd054a81ddd054a042bde054a8978de054a10c6de054a9913df054a2461df054ab1aedf
0fcd054ad149e0054a6497e0054a
Data: f9e4e0054a9032e1054a2980e1054ac4cde1054a611be2054a0069e2054aa1b6e2054a4404e305
1e3054a909fe3054a39ede3054ae43ae4054a9188e4054a40d6e4054af123e5054aa471e5054a59bfe5
00de6054ac95ae6054a84a8e6054a41f6e6054a0044e7054ac191e7054a84dfe7054a492de8054a107b
ad9c8e8054aa416e9054a7164e9054a40b2e9054a1100ea054ae44dea054ab99bea054a90e9ea054a69
54a4485eb054a21d3eb054a0021ec054ae16ecc054ac4bcecc054aa90aed054a9058ed054a79a6ed054a
d054a5142ee054a4090ee054a31deee054a242cef054a197aef054a10c8ef054a0916f0054a0464f005
2f0054a0000f1054a014ef1054a049cf1054a09eaf1054a1038f2054a1986f2054a24d4f2054a3122f3
070f3054a51bef3054a640cf4054a795af4054a90a8f4054ad9f6f4054ac444f5054ae192f505652e
raceback (most recent call last):
  File "producer.py", line 143, in main
    launch_nfn(handle, modules, func_list)
  File "producer.py", line 120, in launch_nfn
    nfo = handle.receive()
  File "/root/programs/cefpyco/core.py", line 159, in receive
    es = cefpyco.receive(self.handler, i, timeout_ms)
KeyboardInterrupt
producer:~/programs#
```

```
05J\xa1%\xbc\x05JDr\bc\x05J\xe9\xbe\xbc\x05J\x90\x0b\xbd\x05J9\xbd\x05J\xe4\xa4\xbd
5J\x91\xf1\xbd\x05J@>\xbe\x05J\xf1\xa8\xbe\x05J\xa4\xd7\xbe\x05JY$\xbf\x05J\x10q\xbf
J\xc9\xbd\xbf\x05J\x84\n\xc0\x05JAW\xc0\x05J\x00\xa4\xc0\x05J\xc1\xf0\xc0\x05J\x84=
x05Ji\xa8\xc1\x05J\x10\xd7\xc1\x05J\xd9#\xc2\x05J\xa4p\xc2\x05Jq\xbd\xc2\x05J@\n\xc3
J\x11W\xc3\x05J\xe4\xa3\xc3\x05J\xb9\xf0\xc3\x05J\x90=\xc4\x05Ji\xa8\xc4\x05Jd\xd7\x
05J!$\xc5\x05J\x00q\xc5\x05J\xe1\xbd\xc5\x05J\xc4\n\xc6\x05J\xa9W\xc6\x05J\x90\xa4\x
05Jy\xf1\xc6\x05Jd>\xc7\x05JQ\x8b\xc7\x05J@\xd8\xc7\x05J1%\xc8\x05J$r\xc8\x05J\x19\x
c8\x05J\x10\x0c\xc9\x05J\tY\xc9\x05J\x04\xa6\xc9\x05J\x01\xf3\xc9\x05J\x00@\xca\x05J
\xa8\xca\x05J\x04\xda\xca\x05J\t'\xcb\x05J\x10t\xcb\x05J\x19\xc1\xcb\x05J$\x0e\xcc\
1[\xc\x05J@\xa8\xcc\x05JQ\x0f5\xcc\x05Jd8\xcd\x05Jy\x8f\xcd\x05J\x90\xdc\xcd\x05J\xa9
ce\x05J\xc4v\xce\x05J\xe1\xc3\xce\x05J\x00\x11\xcf\x05J!\xcfc\x05Jd\xab\xcf\x05Ji\xfb
f\x05J\x90E\xd0\x05J\xb9\x92\xd0\x05J\xe4\xdf\xd0\x05J\x11-\xd1\x05J@z\xd1\x05Jq\xc7
\x05J\xa4\x14\xd2\x05J\xd9a\xd2\x05J\x10\xaf\xd2\x05Ji\xfc\xd2\x05J\x84I\xd3\x05J\xc
6\xd3\x05J\x00\xe4\xd3\x05JA1\xd4\x05J\x84-\xd4\x05J\xc9\xcb\xd4\x05J\x10\x19\xd5\x0
\xd5\x05J\xa4\xb3\xd5\x05J\xf1\x00\xd6\x05J@N\xd6\x05J\x91\x9b\xd6\x05J\xe4\xe8\xd6\
96\xd7\x05J\x90\x83\xd7\x05J\xe9\xd0\xd7\x05JD\x1e\xd8\x05J\xa1k\xd8\x05J\x00\xb9\x
5Ja\x06\xd9\x05J\xc45\xd9\x05J)\xa1\xd9\x05J\x90\xee\xd9\x05J\xfb9;\xda\x05Jd\x89\xda
J\xd1\xd6\xda\x05J@$\xdb\x05J\xb1q\xdb\x05J$\xbf\xdb\x05J\x99\x0c\xdc\x05J\x10Z\xdc\
\xa8\xa7\xdc\x05J\x04\xf5\xdc\x05J\x81B\xdd\x05J\x00\x90\xdd\x05J\x81\xdd\xdd\x05J\x
xde\x05J\x89\xde\x05J\x10\xc6\xde\x05J\x99\x13\xdf\x05J$a\xdf\x05J\xb1\xae\xdf\x05J@
c\xdf\x05J\xd1I\xe0\x05Jd\x97\xe0\x05J\xf9\xe4\xe0\x05J\x902\xe1\x05J)\x80\xe1\x05J\
xcd\xe1\x05Jd\x1b\xe2\x05J\x00i\xe2\x05J\xa1\xb6\xe2\x05JD\x04\xe3\x05J\xe9Q\xe3\x05
0\x9f\xe3\x05J9\xed\xe3\x05J\xe4:\xe4\x05J\x91\x88\xe4\x05J@\xd6\xe4\x05J\xf1#\xe5\x
xa4q\xe5\x05JY\xbf\xe5\x05J\x10\r\xe6\x05J\xc9\xe6\x05J\x84\xa8\xe6\x05JA\xf6\xe6\x
00D0\xe7\x05J\xc1\x91\xe7\x05J\x84\xdf\xe7\x05JI-\xe8\x05J\x10{\xe8\x05J\xd9\xc8\xe8
J\xa4\x16\xe9\x05Jqd\xe9\x05J@\xb2\xe9\x05J\x11\x00\xea\x05J\xe4M\xea\x05J\xb9\xb6\xe
05J\x90\xe9\xea\x05Ji7\xeb\x05JD\x85\xeb\x05J!\xd3\xeb\x05J\x00!xec\x05J\xe1n\xec\x
c4\xbc\xec\x05J\xa9\n\xed\x05J\x90X\xed\x05Jy\xa6\xed\x05Jd\xf4\xed\x05JQ8\xee\x05J@
0\xee\x05J1\xde\xee\x05J$, \ref\x05J\x19z\xef\x05J\x10\xc8\xef\x05Jt\x16\xf0\x05J\x04
f0\x05J\x01\xb2\xf0\x05J\x00\x00\xf1\x05J\x01N\xf1\x05J\x04\x9c\xf1\x05Jt\xea\xf1\x
x108\xf2\x05J\x19\x86\xf2\x05J$\xd4\xf2\x05J1"\xf3\x05Jp\xfb3\x05JQ\xbe\xf3\x05Jd\x0
4\x05JzY\xf4\x05J\x90\xa8\xf4\x05J\xa9\xf6\xf4\x05J\xc4D\xf5\x05J\xe1\x92\xf5\x05e.'
param1 = 6 param2 = 4
ret_add : 10
ret_sub : 2
Ln 4, Col 1 Spaces: 4 UTF-8 LF Python
list(ret_list) : 10000
type(ret_list) : <class 'list'>
root@consumer:~/programs#
```

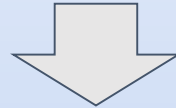
ファンクションキャッシュ

□ ICN でファンクションの実行を要求する拡張

- しかし、キャッシュできるのは計算の実行結果のみ
中間ノードの計算リソースを利用する事はできない

□ 中間ノードによる最適化の効果

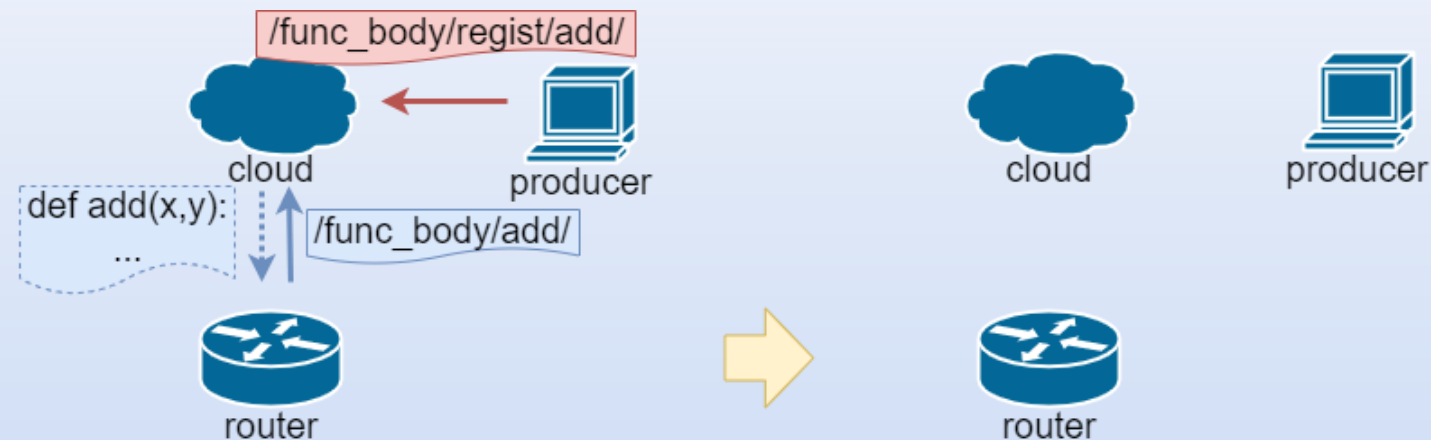
- 計算を行うノードを分散する ⇒ 負荷分散
- エッジに近いノードが計算を行う ⇒ 応答時間の短縮



- 中間ノードでファンクションキャッシュができるアーキ
を開発

動作原理 (1)

- 中間ノードがファンクションを実行するためには、ファンクションのプログラム本体が必要
 - ファンクション名に加えプログラムもレポジトリへ

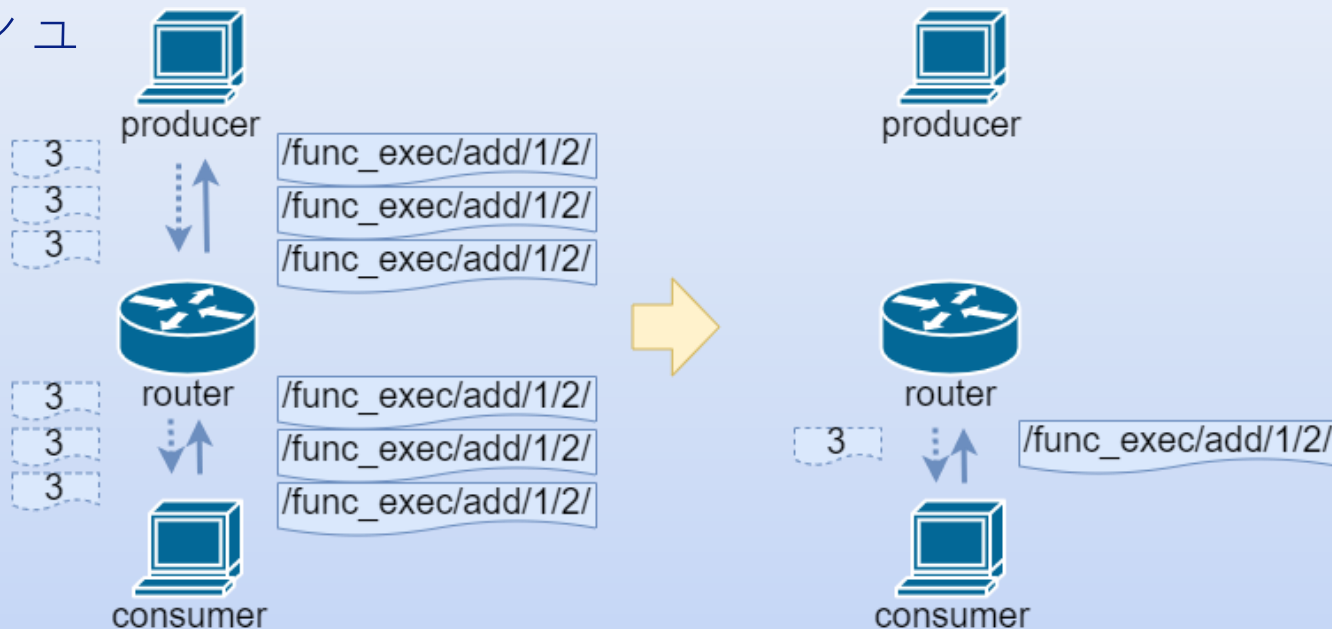


producerがファンクションaddを登録
routerがそれを取得

routerはaddの本体を持っているので、
producerではなくrouterがaddを実行

動作原理 (2)

- キャッシュしたいのは、要求の多いファンクション
 - ルータを通過するファンクション実行要求パケットを監視
 - 実行回数をカウント、一定の閾値を超えたらキャッシュ



addの実行パケットがrouterを何度も通過
⇒キャッシュ実行

以降はproducerに転送する事なく
routerが自分で実行

追加する仕様

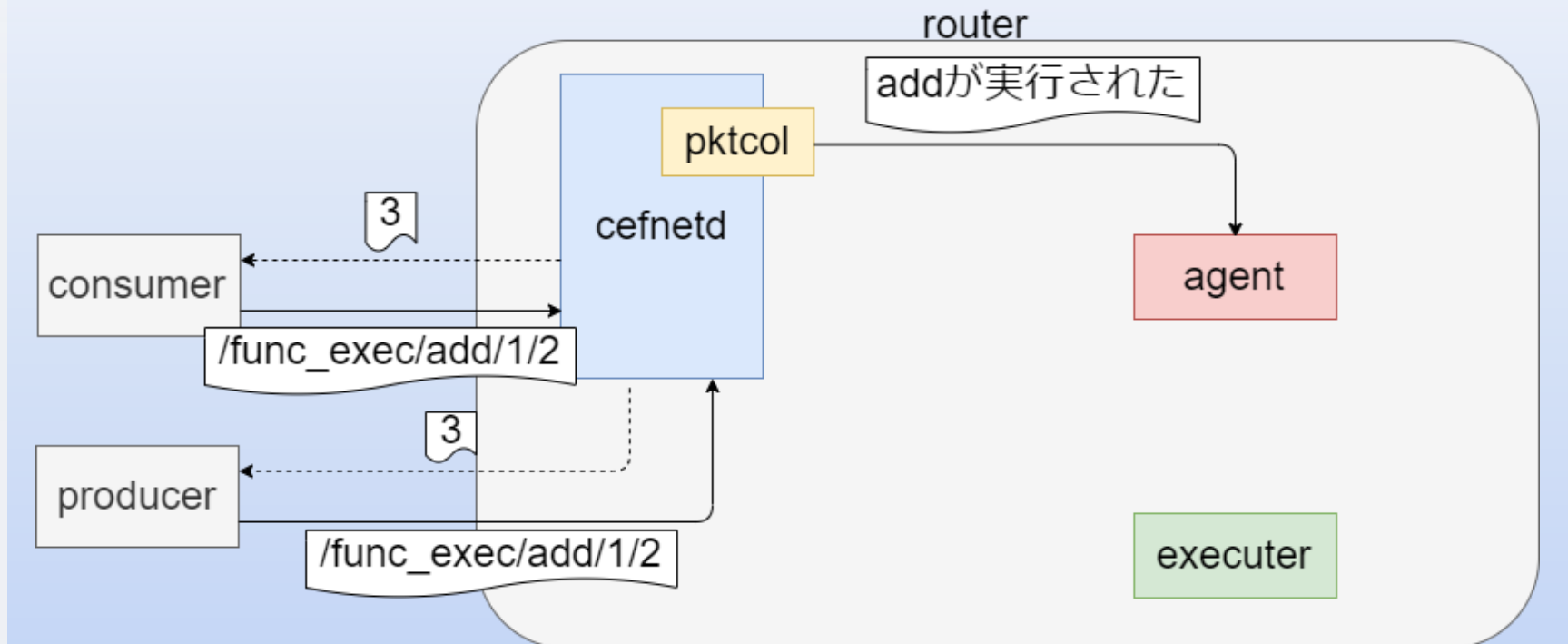
- ファンクション本体の取得要求をできるようにする
 - ccn:/func_body/regist/ で登録
 - ccn:/func_body/ で取得
- 実行回数を数えるために、ファンクションの実行とそれ以外のパケットを区別
 - ccn:/func_exec/ というプリフィクスの付いたパケットを実行要求とする
 - フォワーディングするパケットを監視する変更を cefnetd に加える

システムの構成

- 3つのコンポーネントをルータに配置
 - *agent*
 - システムの中心となる制御プログラム
 - *pktdcol* (packet collector)
 - cefnetd 内を通過するパケットを監視する
 - *executer*
 - ファンクションの実行を行う

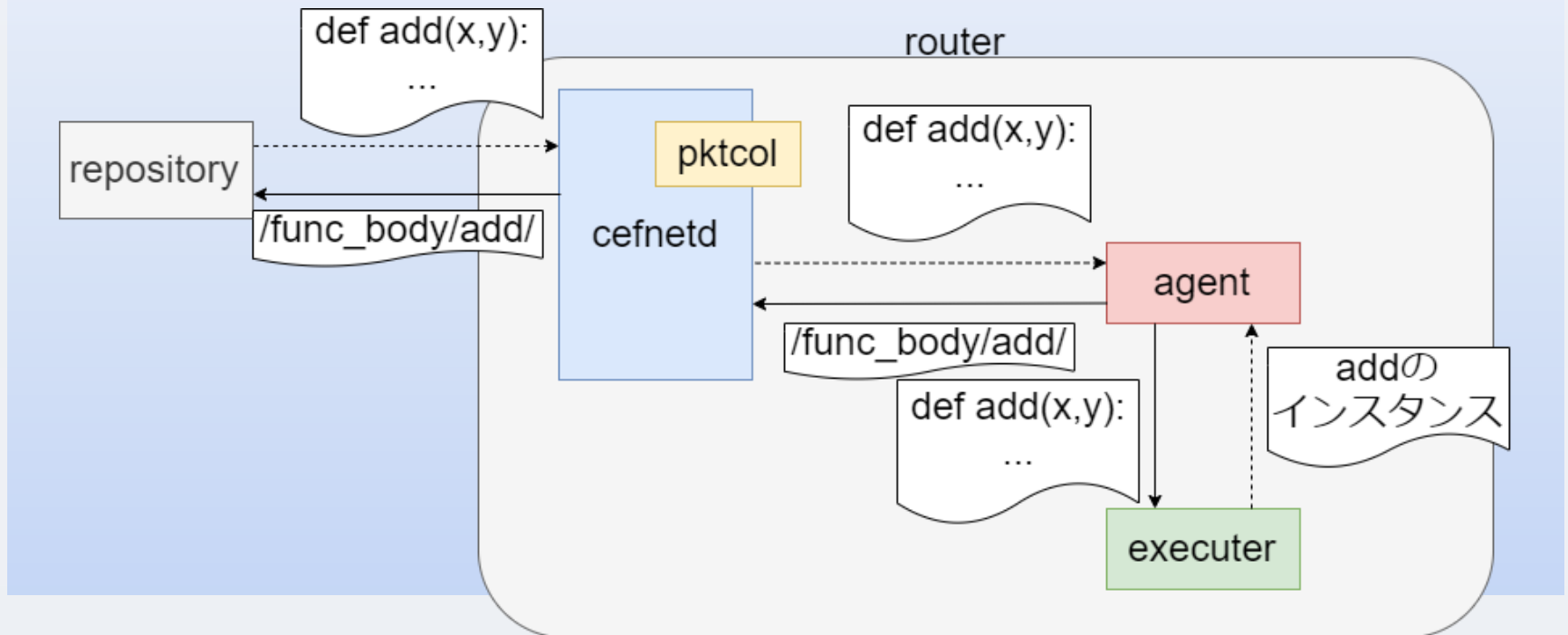
パケットの監視

- **pktcol** が **cefnetd** 内を通過する **ccn:/func_exec/** で始まるパケットを監視
 - **agent** に該当するパケットに関する情報を渡す



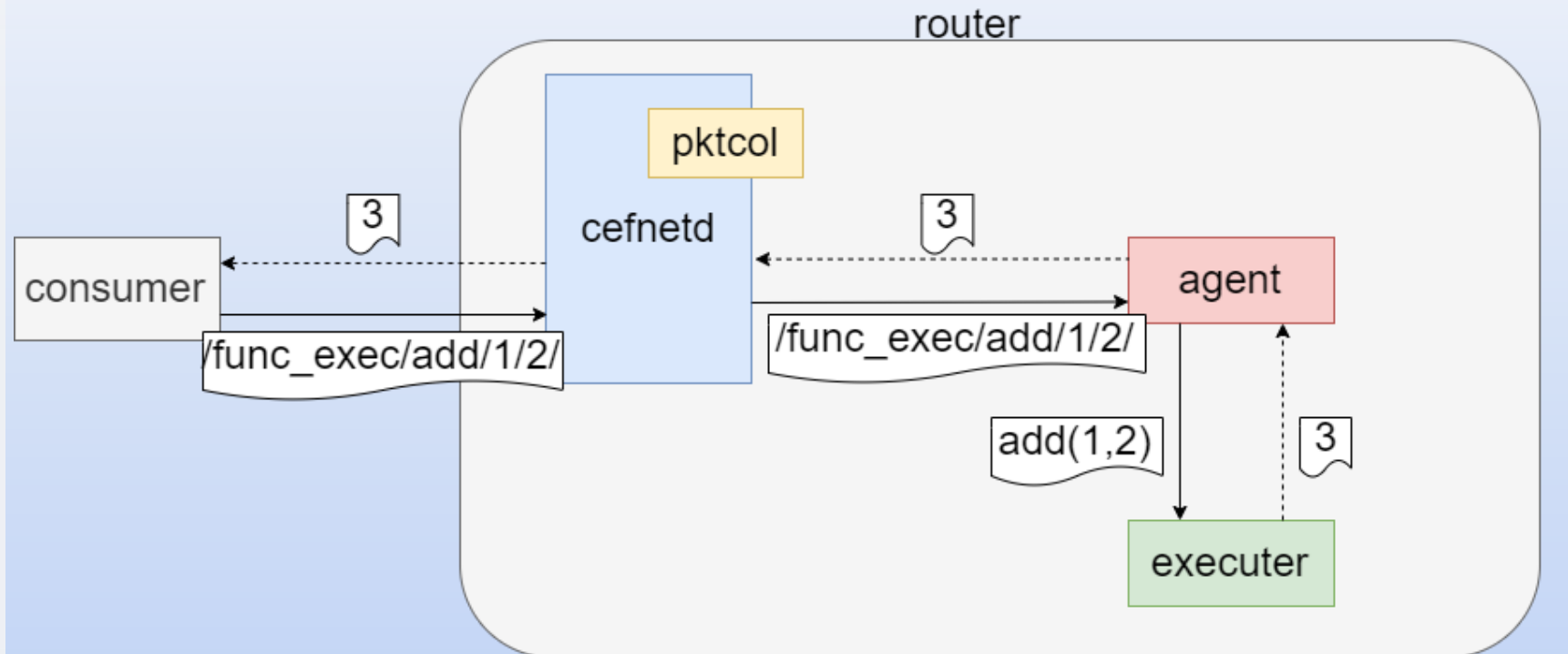
キャッシュの実行

- 一定回数実行パッケージが通過すると、agent がレポジトリからファンクションを取得
 - 取得したものは executer へ渡され、インスタンス(実行のためのハンドル)を作成して agent に返す

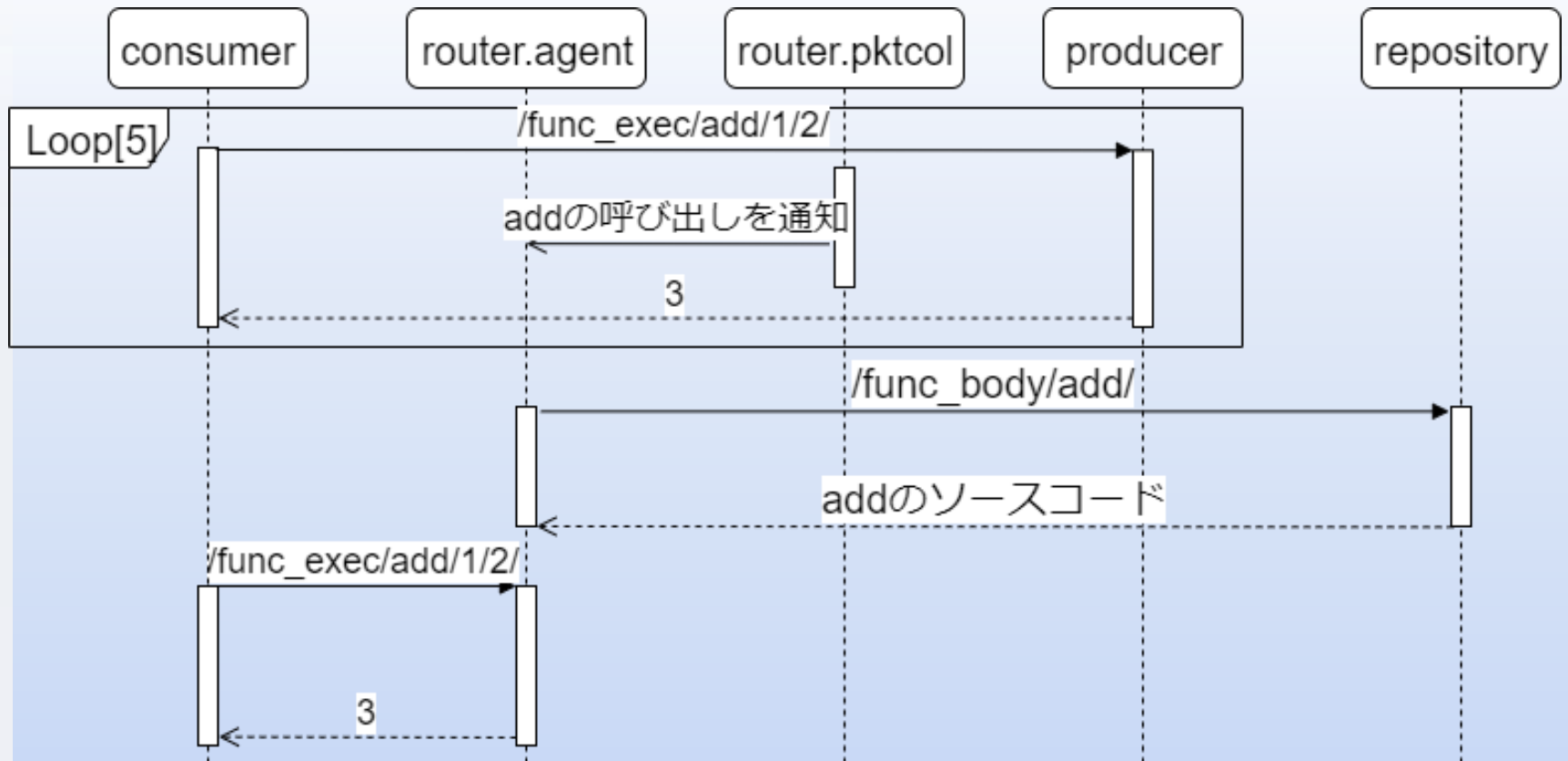


ルータによるファンクション実行

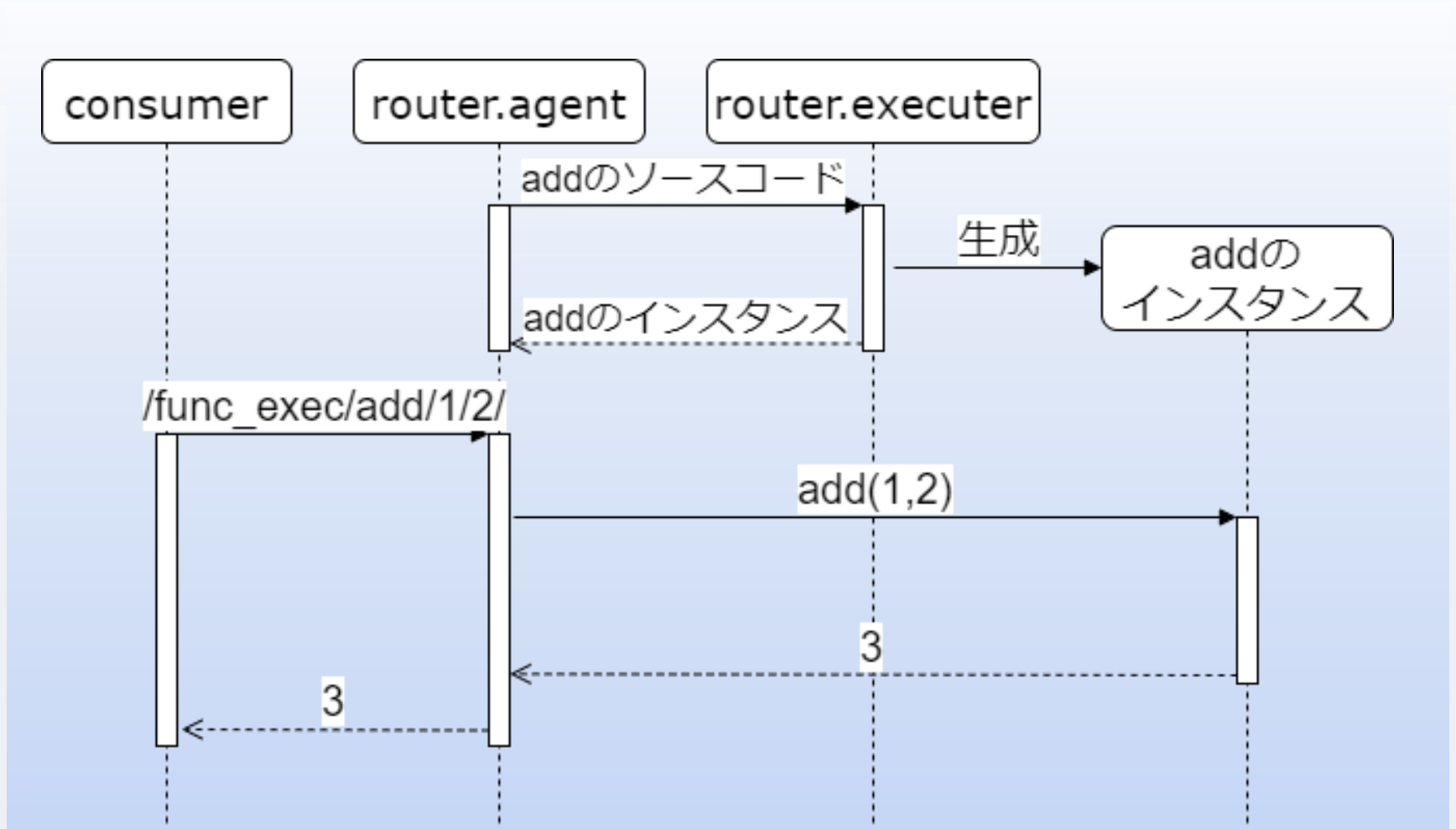
- キャッシュが完了すると、ルータがファンクションを実行できる
 - agent がインスタンスを通じて executer で実行



キャッシュ実行のシーケンス図



ファンクション実行要求への返答のシーケンス図

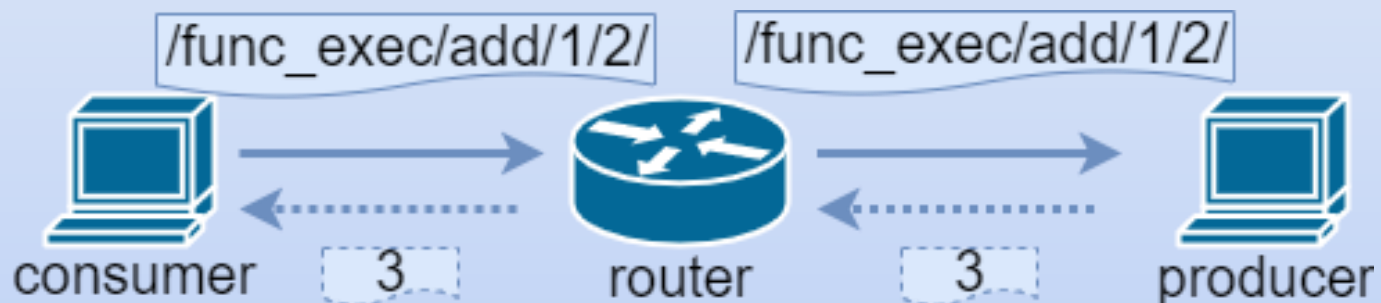


動作確認 (1)

動作環境

- Ubuntu 18.04.3 LTS
 - VMWare Workstation 15 Player on Windows10
- Cefore 0.8.1/cefpico 0.3.7 のカスタム版
- Cefore-Emu 0.1.0
 - Cefore が提供するエミュレータ
 - Mininet-CCNx を元になっている

確認に使用するネットワーク



動作確認 (2)

□ 条件

- producer がファンクション add を提供

 - 2つの整数を加算して返す

- ファンクション本体は Python プログラム

- router 上でキャッシュシステムを動作

- 5回実行要求があるとキャッシュ

□ 確かめること

- router-producer 間の接続を切断して、consumer が add の実行結果を取得できるかどうか

- router がキャッシュしていれば結果が返ってくるはず

- ネットワーク操作コマンドの実行には Cefore-Emu のスクリプトを用いる

動作確認 (3)

- 初期状態での実行結果を確認する

- script_uncached.txt

```
consumer cefgetfile ccn:/func_exec/add/1/10 -f linked.txt  
link router producer down  
consumer cefgetfile ccn:/func_exec/add/2/20 -f ¥  
unlinked.txt
```

- linked.txt の中身は"11"

- unlinked.txt は生成されない(受信失敗)

動作確認 (4)

□ キャッシュ後の動作を確認する

□ script cached.txt

```
consumer cefgetfile ccn:/func_exec/add/1/10  
consumer cefgetfile ccn:/func_exec/add/2/20  
consumer cefgetfile ccn:/func_exec/add/3/30  
consumer cefgetfile ccn:/func_exec/add/4/40  
consumer cefgetfile ccn:/func_exec/add/5/50  
link router producer down  
consumer cefgetfile ccn:/func_exec/add/6/60 -f cached.txt
```

□ cached.txt の中身は "66"

- add を router が実行して返している

まとめと課題

- ICNのプログラマビリティを向上させるファンクション実行システムの改良
 - icnsupport の機能拡張
 - ファンクションキャッシュ機能の追加
- 課題
 - ファンクションへの引数の制約
 - マルチチャック、型情報の送受信
 - 基底クラス型以外のデータの送受信
 - キャッシュの判断を行う戦略の評価
 - 一定回数実行後キャッシュという戦略の妥当性
 - キャッシュ戦略の評価方法