国立研究開発法人
情報通信研究機構

第19回ICN研究会ワークショップ

Cefore/Cefpycoを用いたICNによるネットワーク機能呼び出し実装

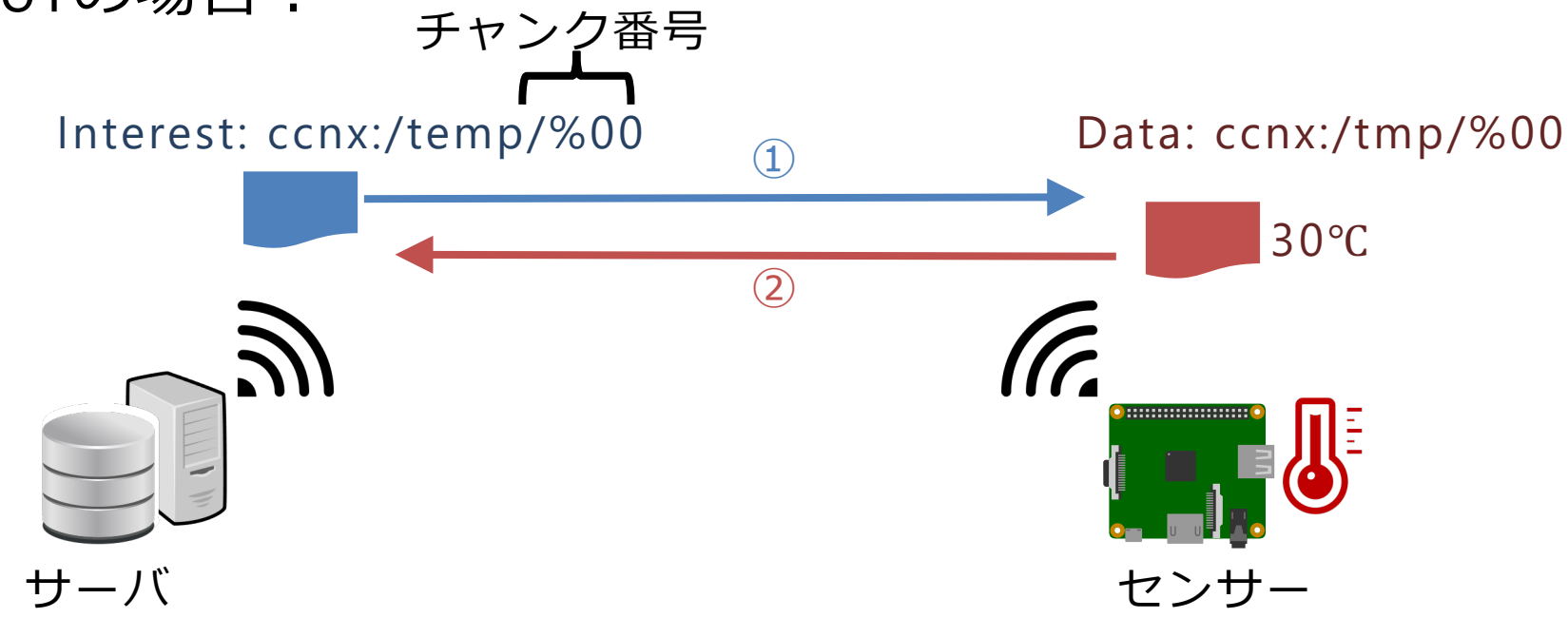
2021年8月26,27日

ICNによる名前を用いた通信

■ICNの基本原則

- Pull型通信：
  - 1) 取得したい情報/コンテンツの名前を指定したInterestを送信
  - 2)ネットワーク(のキャッシュ)から当該Dataを取得

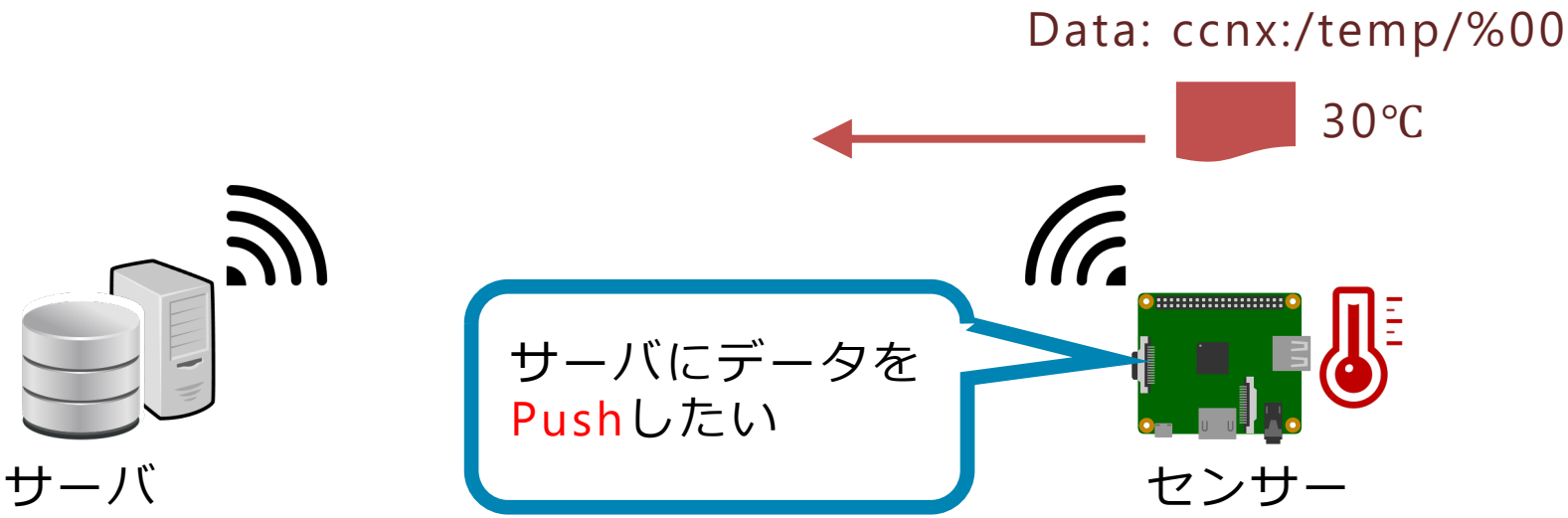
■IoTの場合：



ICNによるPush型通信

■IoT環境では、Push型通信もしばしば必要

- どうする？



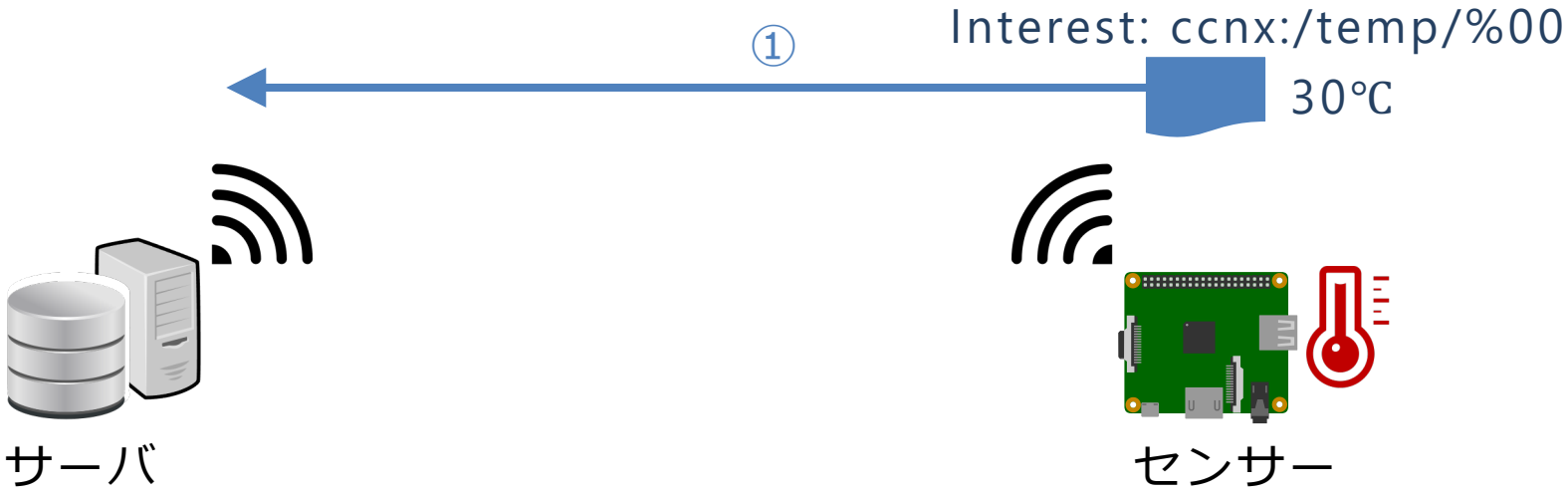
ICNによるPush型通信（方法１）

■方法１：

- センサーがInterestに情報を載せてサーバに送る
  - CeforeではPayload TLVが定義されている(参照: CCNx(RFC8609))

■方法１の特徴

- メリット：シンプルで分かりやすいが、
- デメリット：一つのInterestで一つのデータをネットワークから引き出すというICNの原則から乖離



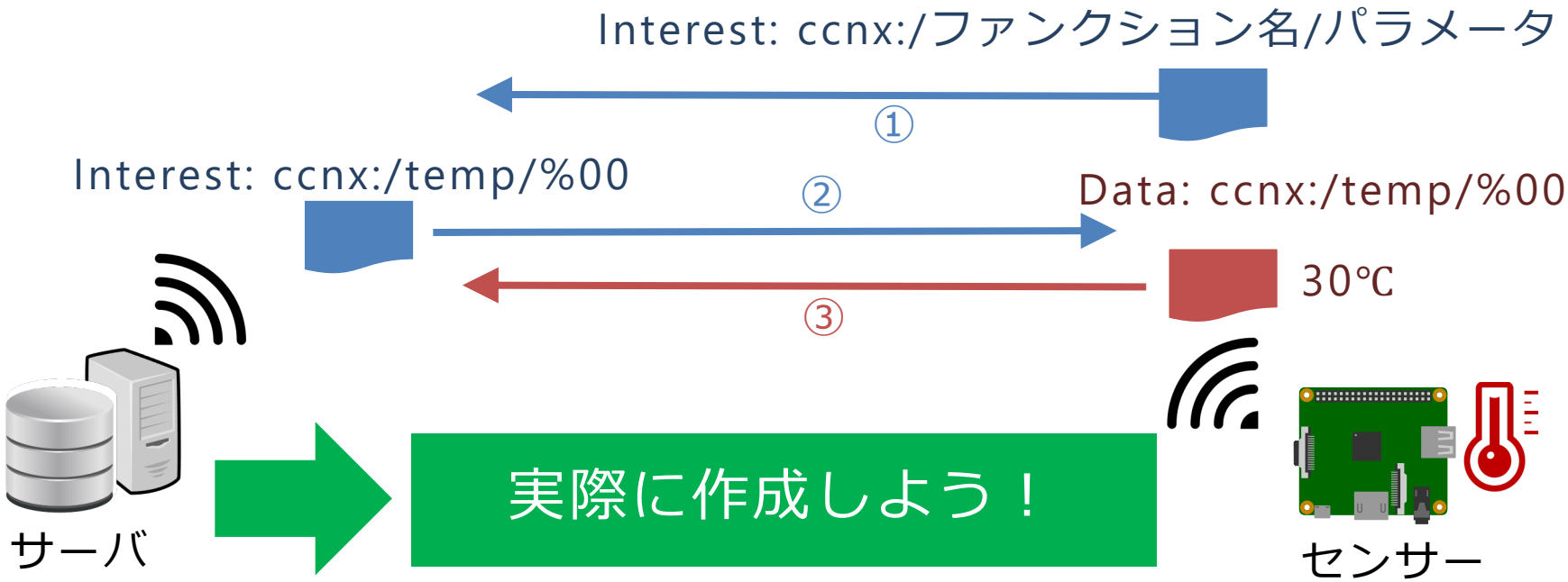
ICNによるPush型通信（方法２）

■情報の名前を指定したPull型通信

- ネットワーク内のキャッシュからデータを引き出すネットワーク機能と捉えることができる

■方法２：

- ネットワークにデータをキャッシュさせるPush機能をネットワーク機能として定義し名前で呼び出す



国立研究開発法人
情報通信研究機構

ネットワーク機能呼び出し Practice（１）

Push型通信基本編

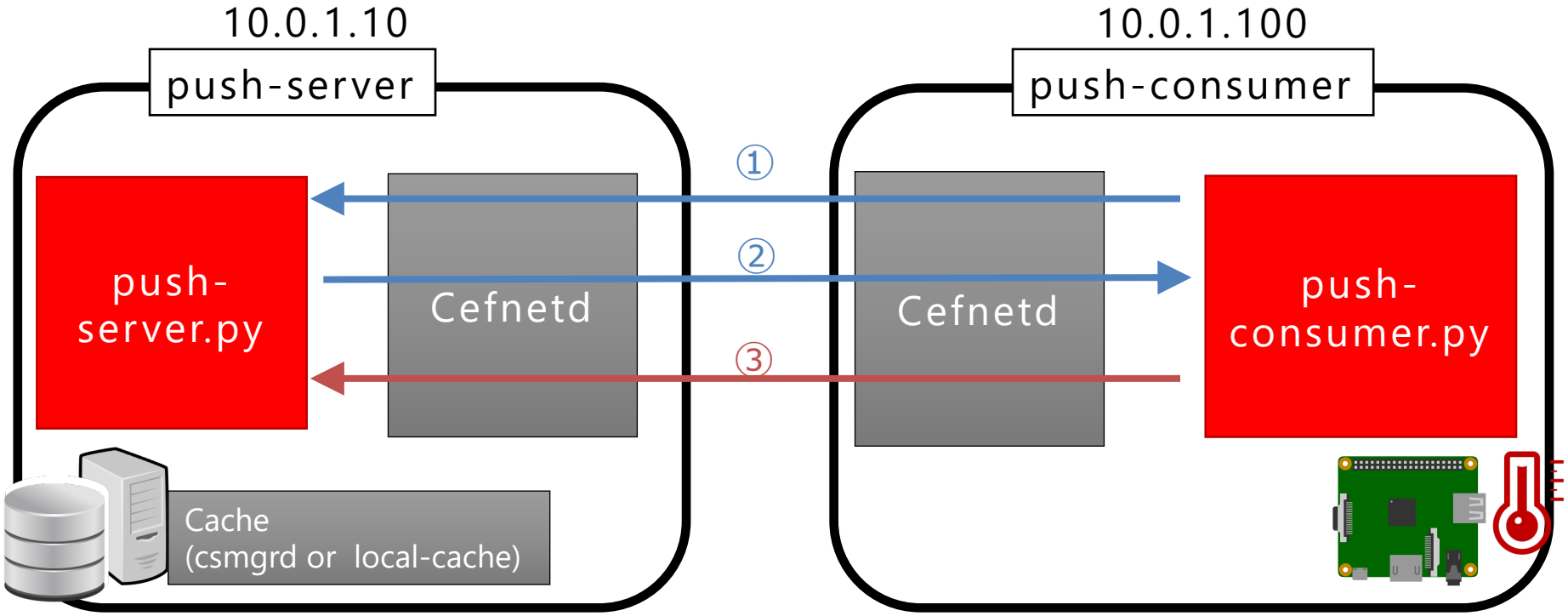
ICNによるPush型通信: 全体図(方法２)

■ ゴール：

① センサー(push-consumer)は、PUSHイベントをネットワーク機能として呼び出すためのInterestを送信し、

② サーバー(push-server)は、指定された名前を使ってInterestをセンサーに送信し、

③ センサーは、該当するデータ(文字列: 30 degree celsius)をサーバに送信する



ICNによるPush型通信:ネーミングルール

■ 下記は、２つのサンプルプログラム(push-server.py, push-consumer.py)で利用しているネーミングルール

- ccnx:/_SF_abcdef_/K/NAME/%00

①　②　③

①ネットワーク機能名(_SF_abcdef_)

- push-consumer.py及びpush-server.pyにて定義している箇所を確認してみよう

②チャンク数(K)

- push-consumer.pyで定義している箇所を確認してみよう

③キャッシュさせたい**データの名前(NAME)**

- push-consumer.pyにて定義している箇所を確認してみよう

■ センサーがPush機能を呼び出すために送信するInterestの名前は、サンプルコードそのままの場合、以下となる

- ccnx:/_SF_abcdef_/5/Current-Temp/%00

ICNによるPush型通信: 実行準備

■前準備

- “push-consumer”コンテナ、及び”push-serverコンテナを起動
  - $ ./start-C-1.bash
- ターミナルから”push-consumer”にログイン
  - $ docker exec –it push-consumer bash
- 別のターミナルから”push-server”にログイン
  - $ docker exec –it push-server bash

ICNによるPush型通信: 実行準備

■ 前準備

- センサー(**push-consumer**)
  - FIB設定を行う。以下のコマンドを実行。

    $ cefroute add ccnx:/_SF_abcdef_ udp 10.0.1.10
- サーバー(**push-server**)
  - FIB設定を行う。以下のコマンドを実行

    $ cefroute add ccnx:/Current-Temp udp 10.0.1.100
  - キャッシュを行う必要があるため、cefstatusコマンドにて、Cache ModeがLocalcacheになっているか確かめよう
    - $ cefstatus

  - キャッシュを行う必要があるため、CS_MODE=1 (Local-Cacheを有効化)でceforeが起動している必要あり。(または、CS_MODE=2 (use csmgrd)でCsmgrdを使って実行しても良い)。
    - Local-Cacheを有効にする手順
      - 1) /usr/local/cefore/cefnetd.confを編集し、CS_MODEを以下にする
        - CS_MODE=1
      - 2) cefnetdの再起動
        - $ cefnetdstop
        - $ cefnetdstart

ICNによるPush型通信: 実行準備

■ 実行する前に確かめよう

- センサー(**push-consumer**)のFIB設定はできているか？
  - コマンド"cefstatus"を実行して、下記が表示されるか？
    - faceid = XX : address = 10.0.1.10:9896 (udp)
    - FIB: 1
      - ccnx:/_SF_abcdef_
        - Faces: XX (-s-)
- サーバー(**push-server**)のFIB設定は出来ているか？
  - コマンド"cefstatus"を実行して、下記が表示されるか？
    - faceid = YY : address = 10.0.1.100:9896 (udp)
    - FIB: 1
      - ccnx:/Current-Temp
        - Faces: YY (-s-)
- サーバーのキャッシュモードは、LocalCache)になっているか？
  - コマンド"cefstatus"を実行して、下記が表示されるか？
    - Cache Mode : LocalCache

Practice

ICNによるPush型通信: 実行

- 1) サーバー(**push-server**)で下記コマンドを実行
  $ python3 bin/push-server.py
- 2) センサー(**push-consumer**)で下記コマンドを実行
  $ python3 bin/push-consumer.py
- 3) サーバー(**push-server**)で適切にキャッシュされているか確認してみよう
  - (Ctrl+c を入力し、push-server.pyを停止し、下記を実行)
  - $ cefgetfile ccnx:/Current-Temp –f ./cache-data.txt
  - $ cat cache-data.txt
    - (“30 degree celsius”という内容がチャンク数分だけ表示されるはず!!)

- 再度、このシナリオを実行したい場合
  - サーバーにキャッシュ(ccnx:/Current-Temp)が残っているので、キャッシュを消すために、サーバーにてcefnetdを一旦終了し、再起動しよう
    - $ cefnetdstop
    - $ cfnetdstart
    - $ python3 push-server.py

※ push-server.pyなどのプログラムを強制終了したい場合、Ctrl+c を入力

Practice

ICNによるPush型通信: 試してみよう

- **Push機能を呼び出す名前とPushするデータ名を自分で決めてみよう**
- 下記は、２つのサンプルプログラム(Push-Server.py, Push-Consumer.py)から抜粋
  - ccnx:/_SF_abcdef_/K/Name/%00

①　②　③

①ネットワーク機能名

- "_abcdef_"の部分を自分で定義して、push-consumer.pyの該当箇所を変更してみよう

②チャンク数

- チャンク数は一つでも良いので、 push-consumer.pyの該当箇所"k"を1に変更してみよう。

③キャッシュさせたいデータの名前

- データの名前を決めて、２つのサンプルプログラムの該当箇所"Name"を変更してみよう。

センサーとサーバーのFIB設定を適切に反映することを忘れずに

- PUSH機能を呼び出すInterestの名前例:
  - ccnx:/_SF_PUSH_/1/Current-Temp-MySensor/%00

ICNによるPush型通信:発展課題

■色々考えてみよう

- サーバー側のFIB設定に関して
  - サーバーがセンサーからPUSH機能要求を受信した際に、自動的にFIB設定するにはどうしたら良いだろうか？
- 中継ノード(ルータ)を介してPushを行う場合、ルータはどの様な機能が必要か？
  - （サーバ)<--> (中継ノード(ルータ))<--> (センサー)
- 様々な場所に温度センサー等が散らばっている状況で、複数センサーが現在の温度データを同一サーバにPushする場合、
  - どの様なPUSH通信を行う必要があるだろうか？
  - また、どんなメリットやデメリットがあり得るだろうか。

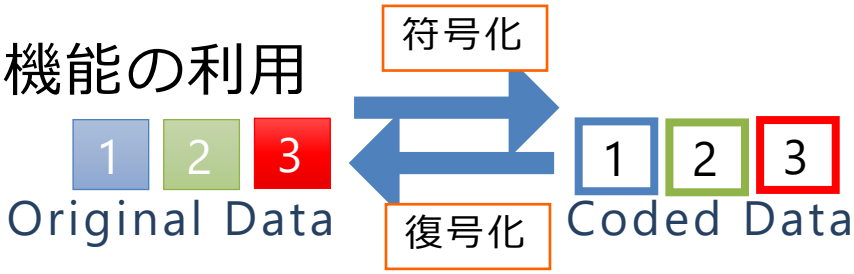
国立研究開発法人
情報通信研究機構

ネットワーク機能呼び出し Practice（2）

Push型通信応用編

Practice

ICNによるPush型通信: 応用編

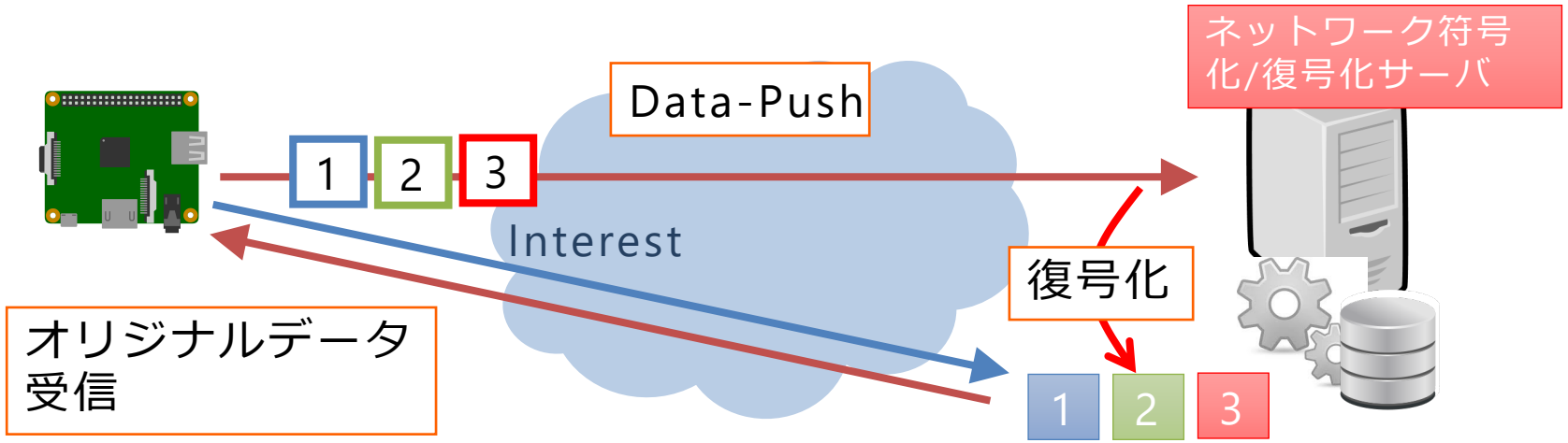


- PUSH型通信用いたネットワーク機能の利用
- 例えばネットワーク符号化(NC)
  - シナリオ例:

    1) 計算機性能が乏しいノード、あるいは復号化機能が無いノードが符号化データK個受信。
       - 符号化データ: オリジナルデータ(K個)を元に作成され、オリジナルデータとは異なるデータ
       - 復号化：符号化データ(K個)を元にオリジナルデータ(K個)を生成

    2) 符号化データをサーバに渡し、復号化・キャッシュしてもらう

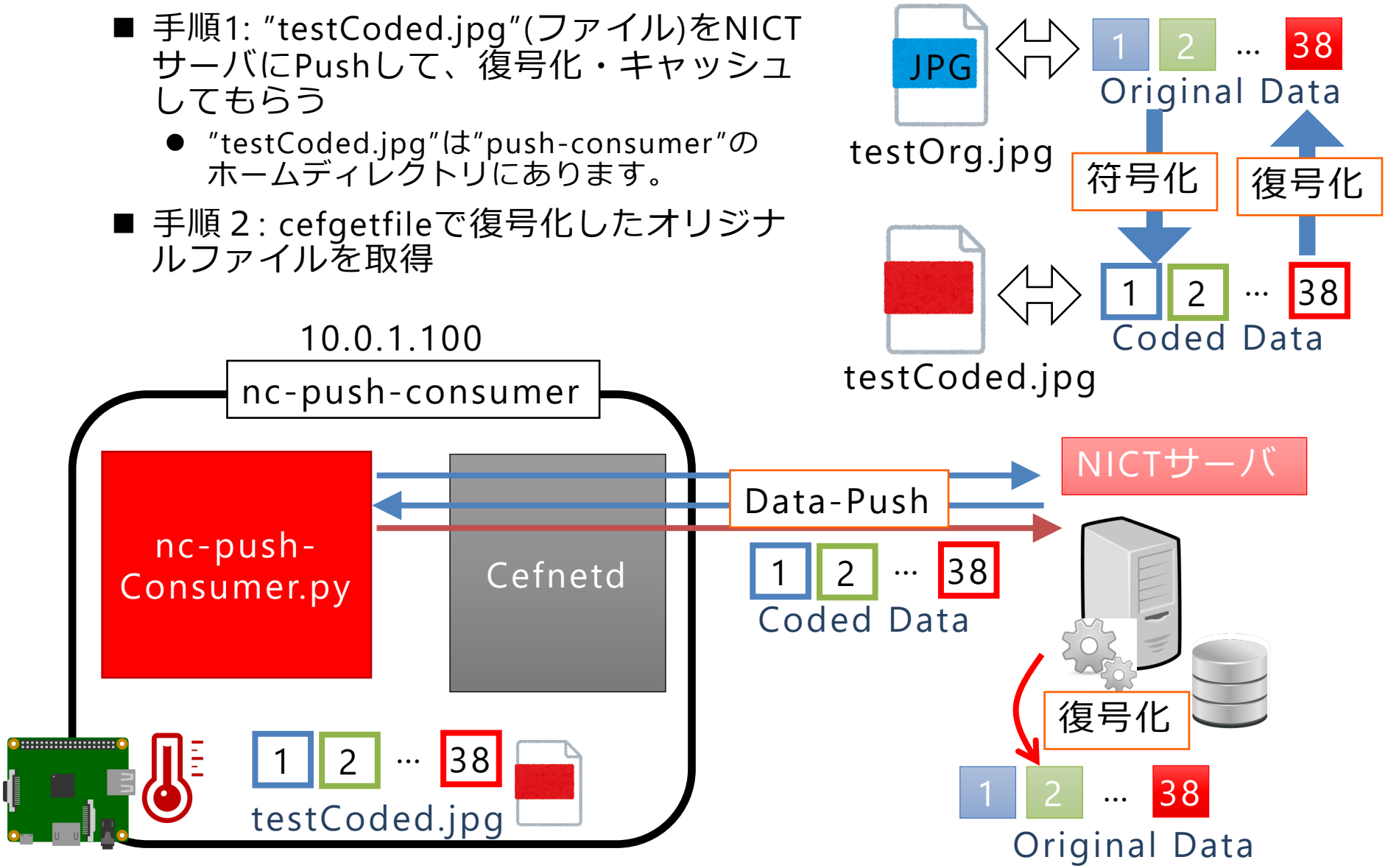
    3) サーバにキャッシュされてあるオリジナルデータを受信



Practice

ICNによるPush型通信: 応用編

- 手順1: "testCoded.jpg"(ファイル)をNICTサーバにPushして、復号化・キャッシュしてもらう
  - "testCoded.jpg"は"push-consumer"のホームディレクトリにあります。
- 手順２: cefgetfileで復号化したオリジナルファイルを取得



ICNによるPush型通信: 応用編

(前準備)

■ホストの"practice-C"ディレクトリにて、

- 既に起動してあるコンテナを終了させよう
  - $ ./teardown.bash
- 設定ファイルを変更するために、下記を実行
  - $ cp docker-compose_global docker-compose.yml
- "nc-push-consumer"コンテナを起動
  - $ ./start-C-2.bash

■ターミナルから"nc-push-consumer"にログイン

- $ docker exec –it nc-push-consumer bash

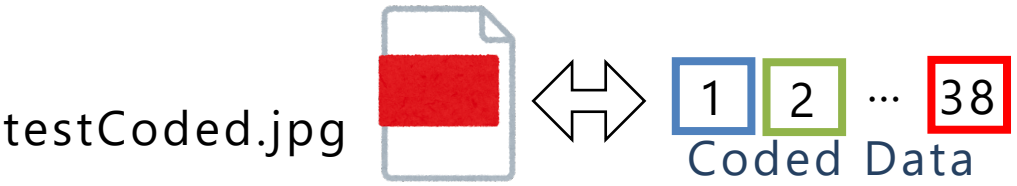
Practice

ICNによるPush型通信: 応用編

- ”nc-push-consumer”にて、符号化データをサーバに送信し復号化を要求
  - Fibを追加
    - $ cefroute add ccnx:/_SF_ tcp [NICTサーバのIPアドレス]
  - nc-push-consumer.pyを実行
    - $ python3 bin/nc-push-consumer.py bin/testCoded.jpg 自身の苗字
      - 例：$ python3 bin/nc-push-consumer.py bin/testCoded.jpg Matsuzono

上手くサーバからInterestを受信できない場合、“cefstatus”コマンドで、FIBが適切に設定されているか確かめてみよう。どうしても動かない/動かなくなった場合、cefnetdを再起動してよう

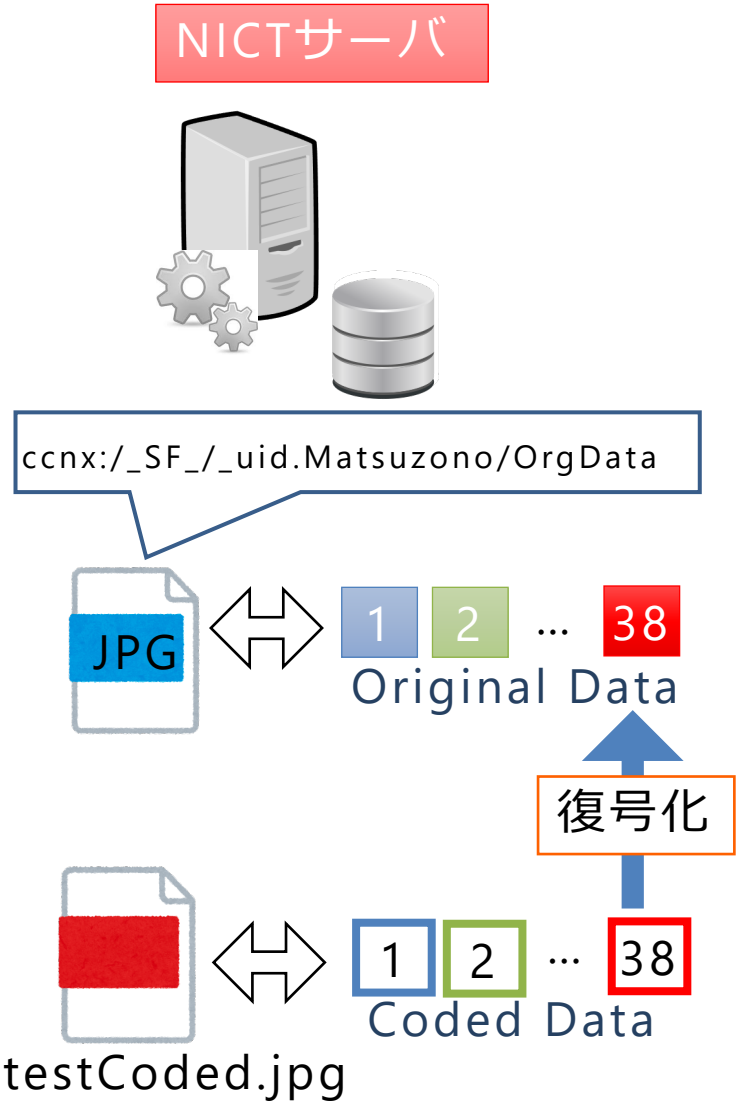
  - 実行内容
    - 符号化データをPushするためのInterest送信
      - 名前: ccnx:/_SF_/_CS.STORE_/_K.38/_uid.Matsuzono_/
    - NICTサーバは、consumerに38個のInterestを送信してくれます
      - 名前: ccnx:/_uid.Matsuzono_/chunk=0,...,37
    - testCoded.jpgファイルを1024byteづつ読み込み、サーバに符号化データを38個送信(PUSH)
      - 名前：ccn:/_uid.Matsuzono_/chunk=0,...,37



ICNによるPush型通信: 応用編

■ サーバ側の実行内容：

- NICTサーバは、consumerに38個のInterestを送信
  - 名前: ccnx:/_uid.Matsuzono_/chunk=0,...,37
  - このために、"cefroute"コマンドを自動で実行し、適切なFIBを作成する
- 38個全ての符号化データを受信した後、オリジナルデータを復元し、以下のファイル名でキャッシュする
  - ファイル名：ccnx:/_SF_/_uid.Matsuzono_/OrgData
- その際、チャンクデータをいくつ受信したかをステータス情報として、以下のファイル名でキャッシュ
  - ファイル名：ccnx:/_SF_/_uid.Matsuzono_/Status



Practice

ICNによるPush型通信: 応用編

■ 復号化されたオリジナルデータをNICTサーバから受信しよう

- cefgetfileの利用
  - 38個のオリジナルデータを受信し、“testOrg.jpg”というファイル名で保存
  - $ cefgetfile ccnx:/_SF_/_uid.Matsuzono_/OrgData –f /tmp/log/testOrg.jpg
- 保存したtestOrg.jpgが適切に開けるか見てみよう
  - ホストの“/tmp/log”ディレクトリのtestOrg.jpgを開こう
- うまくtestOrg.jpgが受信できない場合、サーバのステータスを調べてみよう。
  - $ cefgetfile ccnx:/_SF_/_uid.Matsuzono/Status -f ./Status
  - $ cat ./Status

■ 課題：

- “push-server”コンテナを使って、サーバプログラムを自作し、”push-consumer”コンテナを使って、独自の機能を実装してみよう。