情報サービスによるネットワーク制御

是津耕司

IoT の発展に伴い、モノや人によって提供される情報サービスを様々に組み合わせ、人々の生活や社会システムに直接働きかけることができるネットワークへの期待が高まってきている。Service-Controlled Networking (SCN) は、サービス連携に応じて動的にネットワーク資源を制御するための技術で、SDN とクラウドの中間に位置するミドルウェアとして実現される。本稿では、SCN の基本概念と実装方法、応用について概説する。

1　まえがき

Internet of Things (IoT) に代表されるように、今や情報機器だけでなく、あらゆるモノや人までがネットワークに接続し情報をやり取りするようになりつつある。ネットワークは、もはや単なるデータの伝送路としてだけではなく、様々なスマートサービスが情報を収集、分析、配信するプラットフォームとしての役割を担おうとしている。その結果、情報機器からモノ、ヒトまで、情報を発信、取得、処理する多種多様な“情報サービス”を、アプリケーションの情報要求に応じて柔軟かつ迅速につなぎあわせ、限られたネットワーク資源を有効活用しながら情報サービスを効率的・効果的に連携させるネットワーク技術の必要性が高まってきている。例えば、スマートシティでは、平常時には、環境や交通など、種々のセンサネットワークを使ったスマートサービスを提供するが、災害などの非常時には、これらを横断的につなぎ、多種多様な被害情報を集中的に収集・解析したり、避難や救援に関する情報を人々に配信することが考えられている[1]。アプリケーション専用のネットワークを構築するための技術として、従来はアプリケーションごとに通信を仮想的に分離することが可能な VPN などの技術が用いられてきたが、エンドノード間で接続を事前に調停する必要があり、状況に応じて情報サービスの連携を変えるネットワークをオンデマンドに構成することには適していない。一方、近年では、クラウドと Software-Defined Networking (SDN) を連携させたプラットフォームを開発しようという試みが盛んに行われているが[2][3]、ネットワーク資源や計算資源を仮想化するための技術が中心で、アプリケーションの要求に応じた資源の割当ては依然管理者により人手で行われている。しかし、IoT 時代に入り、対象とする情報サービスが爆発的に増えると、それらの構成を全て人手で行うことは困難であり、アプリケーションの情報サービス連携の要求を解釈し自動的にネットワークの構成を設定・調整する技術が必要となる。

こうした要望に応えるため、我々は、Service-Controlled Networking (SCN) という概念を提唱し、それを実装したミドルウェアを開発している[4]-[7]。SCN は、アプリケーションの情報サービス連携に応じて、ノード発見やパス生成、データ処理、QoS 設定などのネットワーク制御を動的に行うための技術である。SCN は、新世代ネットワークのための国際標準 ITU-T Y.3001[8] における、“データアウェア、あるいはサービスアウェアなネットワーク”の概念を実現するものとして位置付けられる。本稿では、SCN の基本設計、広域ネットワークテストベッド上への実装、応用システム開発、及び今後の展望について述べる。

2　SCN の基本概念

Service-Controlled Networking（SCN）は、アプリケーションごとに様々な情報サービスの間でやり取りを行う情報フローを、ネットワーク上にマッピングする技術である。従来は、手作業でネットワークを構成していたが、SDN に代表されるプログラム可能なネットワークの登場により、ソフトウェア的に仮想ネットワークを動的に構成できるようになってきた。そこで SCN では、アプリケーションの情報フローを実行するために必要なネットワーク構成を判断し、プログラム可能なネットワークの設定を自動的に行うことにより、情報フローに連動した仮想ネットワークを動的に構成できるようにする。これを実現するために、SCN は図 1 に示す要素から構成される。**宣言的サービスネットワーキング（Declarative Service Networking: DSN）**は、情報サービス間の情報フローを宣言的に記述するためのルール言語と、そのインタプリタで構成

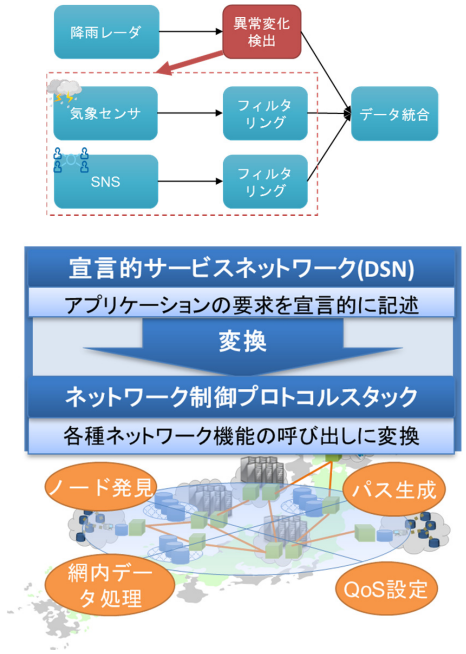

図 1　SCN の基本概念

```
spl path(@Src,Dest,Path,Cost) :- link(@Src,Dest,Cost),
    Path=f_init(Src,Dest).
sp2 path(@Src,Dest,Path,Cost) :- link(@Src,Nxt,Cost1),
    path(@Nxt,Dest,Path2,Cost2), Cost=Cost1+Cost2,
    Path=f_concatPath(Src,Path2).
sp3 spCost(@Src,Dest,min<Cost>) :- path(@Src,Dest,Path,Cost).
sp4 shortestPath(@Src,Dest,Path,Cost) :-
    spCost(@Src, Dest,Cost), path(@Src,Dest,Path,Cost).
Query shortestPath(@Src,Dest,Path,Cost).
```

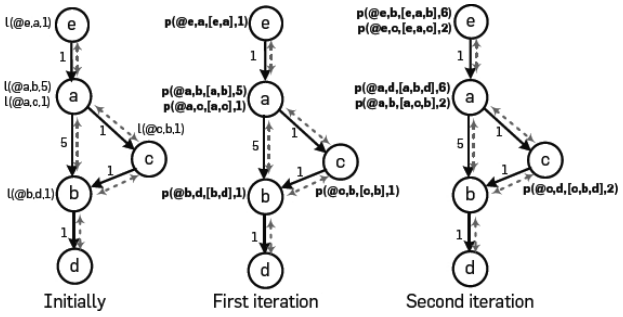

図 2　Declarative Networking（NDlog）による最短経路プロトコルの記述例（[9] より抜粋）

される。アプリケーションは、図 1 に示された例のように、様々な情報サービスからデータを収集し処理するフローを設計し、DSN を使って記述する。一方、**ネットワーク制御プロトコルスタック（Network Control Protocol Stack：NCPS）**は、DSN で記述された情報フローを実行するアプリケーション専用の仮想ネットワークを構築し実行するために、情報サービスを実行するノードを発見し、それらの間にパスを生成することを行う。ネットワークの構成方法はネットワークプロトコルごとに異なるため、それぞれのプロトコルに応じた NCPS が用意される。

SCN の基本となるのは、Declarative Networking 技術[9] である。Declarative Networking は、元々、グラフ構造データベースにおける宣言的ルールと再帰的クエリ言語である Datalog[10] に端を発しており、Datalog をネットワークのデータフロー記述として拡張したものが Network Datalog（NDlog）[11] である。宣言的ルールと再帰的クエリ言語は、演繹データベースで盛んに研究され、元のデータと派生データの関係性や相関制約の表現に適している。NDlog では、ネットワークをデータ交換の場として抽象化し、ノードの状態を示すデータ集合をある状態から他の状態に移すための計算ルールをネットワークプロトコルとして記述する。図 2 に、NDlog による最短経路プロトコル記述の例を示す。この例では、sp1 から sp4 までの 4 つのルールで構成されている。path（Src, Dest, Cost Path）は、ノード Src からノード Dest までコスト Cost の経路 path が存在することを示している。ルール sp1, sp2 は経路（パス）の生成ルールを示しており、sp1 は既存のノード間リンク link（Src, Dest, Cost）をそのままパスとし、sp2 はあるパスに隣接するリンクを組み合わせ新しいパスを生成している。sp3 は、Src から Dest までの最小のコストを特定するルールを示し、sp4 は Src と Dest をコスト最小となるパスでつなぐルールを示している。図 2 の下部は、これらのルールを再帰的に適用して各ノードからの最短経路を計算する様子を示している。各ノードには、そのノードから他のノードへの経路を示す path タプルが表示されている。初期状態では、隣接するノードとのリンクが示されており、1 回目の繰り返しで、ルール sp1 により 1 ホップ分の path タプルが生成される。さらに 2 回目の繰り返しでは、ルール sp2 により 2 ホップ分の path タプルが生成される。これらを再帰的に繰り返し、経路情報を生成した後、ルール sp3、sp4 によりあるノードからあるノードへの最短経路 shortestPath が算出される。NDlog を拡張し、さらにオーバーレイネットワークを記述できるようにしたものが Overlog[12] であり、物理的なネットワークの上に仮想的なノードと論理的なリンクを構成できるようにしている。Overlog は、P2P ネットワーク[13] やコンテンツ配信ネットワークなどに応用され、単なるルーティングだけでなく、メッセージ配信や受信確認（ACK）、故障検出、タイムアウトなども扱えるように NDlog を拡張している。

SCN は、こうした Declarative Networking 技術を拡張し、情報サービス間のデータのやり取りに合わせ、オーバーレイネットワークを動的に構成できるようにする。図 3 に SCN によるネットワーク構成の概要を示す。SCN では、ノード上で稼働する情報サー

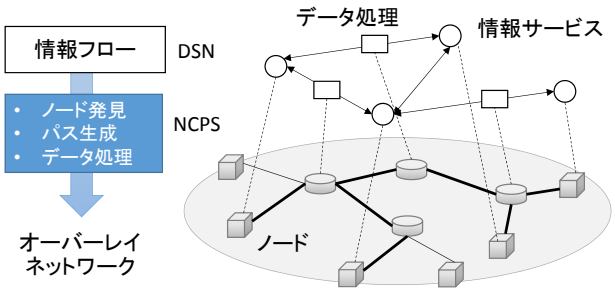

図３　SCN によるネットワークの動的構成

ビスの間で、論理リンクを介してやり取りする情報のフローを、Overlog を拡張した DSN 言語により記述する。そして、DSN 記述の実行時に、NCPS によって情報サービスをノードにマッピングし、情報サービス間のデータ転送のためのパスをそれらのノード間に設定する。さらに、情報フローに指定されたデータ処理を、経路途中のノードで実行する（網内データ処理）。その際、変化し続けるネットワークの状況に対応し、データ転送の輻輳や遅延を抑えられるよう、トラフィックやノード負荷を監視しながら、継続的にパスや網内データ処理の再設定を行う。こうした SCN により、アプリケーション開発者は、情報サービス間の情報フローが変化しても、継続的に安定稼働させられるネットワーク資源を弾力的に確保することが可能になり、従来のようにピーク時の負荷を想定したネットワーク資源を事前に確保しなくても済むようになる。一方、ネットワークプロバイダにとっても、アプリケーションに契約ベースでネットワーク資源を分配していた従来に比べ、アプリケーションの実行状況に応じて動的かつ自動的に資源の割当てを変えられるようになれば、限られたネットワーク資源をより多のアプリケーションに再利用してもらうことができ、利用率の向上につながると期待される。

3　関連研究

ネットワークをソフトウェア的に制御可能にする技術として、Software-Defined Networking（SDN）が近年注目されている。SDN では、プログラムから、API を通じて、ネットワークの経路設定を行うことができ、この機能を使ってネットワークをより効率的に制御すべく様々なドメイン固有プログラミング言語が提案されている。Frenetic[14]、NetCore[15]、NICE[16]、Nettle [17] などは、SDN の代表的な実装である OpenFlow の上で、ルーティングやトポロジ発見などのネットワーク機能を開発するためのプログラミング言語である。これらは、OpenFlow スイッチの状態に応じて経路設定（フローテーブル）を書き換えるためのプログラムを開発することを主眼としており、OpenFlow ネットワーク内の状態に閉じたプログラムのシミュレーション実行やバグ検出などの機能が中心である。一方、Procera[18] は、OpenFlow スイッチ以外のノードで起きるイベント、例えばユーザ認証や、日時、帯域の利用率、サーバ負荷などに反応するルールを記述しフロー制御を行えるようにすることで、アプリケーションの管理ポリシーに応じたネットワーク制御を可能にしている。このように、様々に異なるレベルでの SDN 制御が存在する中で、SCN は、アプリケーションごとの情報フローを Declarative Networking に基づく DSN 言語で記述し、それらの情報フローを実行するためのネットワークを OpenFlow 等の SDN 上に自動的に設定することを行う。また、アプリケーションの実行中にも、ネットワークの輻輳や情報フローの遅延が発生した際には、自動的に再設定を行う。このように、SCN は、従来の SDN 制御技術に比べ、アプリケーションの処理とネットワーク制御との連動を強化したものと言える。

Information-Centric Networking（ICN）[19]（もしくは Content-Centric Networking[20]）は、従来の端末に依存した ID（IP アドレスなど）ではなく、情報コンテンツの識別子（コンテンツの名前など）に応じてネットワーク制御を行う技術である。ICN では、コンテンツ ID とそのネットワークアドレスが登録され、アプリケーションがネットワークにコンテンツ ID を push したり get したりすると、ICN が名前解決をしてデータ転送を実行する。アプリケーションは、ネットワークの構成を意識することなくコンテンツ ID のやり取りに集中でき、ネットワーク管理者もネットワーク資源を弾力的に変更することができるようになる。SCN でも、DSN と NCPS により情報フローとネットワーク構成を分離し、情報サービスとノードの対応付けを動的に発見するようにしている。さらに、SCN では、DSN で記述された情報フローに応じて、情報サービス間にパスを動的に設定したり網内でデータ処理を行ったりするなど、コンテンツの分散処理にまで踏み込んだ制御を可能にしている。

4　SCN の設計と実装

SCN は、動的構成の対象となるネットワークのノード上で稼働するミドルウェアとして実装・導入され、これらが協調して動作し SCN の機能を実行する。また、アプリケーションが SCN を使って連携させる情報サービスは、既存のシステムや Web サービスを共通インタフェースにより隠蔽化（ラッピング）し、サー

ビスメタデータと共に SCN に登録する。さらに、サービス間でやり取りするデータも、このラッパーにより AMON 標準[21] に準拠した共通フォーマットに変換される。こうした実行環境の下で、SCN の機能を実現する方法について、以下に説明する。

4.1　Declarative Service Networking

DSN 記述は、Overlog 言語の実装系の 1 つである Bloom[22][23] に基づいて実装されている。Bloom は、汎用プログラミング言語である Ruby のコードに埋め込まれるドメイン固有言語であり、プロダクションルールによりデータフローを記述する。DSN では、Bloom を拡張し、サービス間で効率的な情報フローを構築できるよう、表 1 に示す関数を実装している。

図 4 に、DSN の記述例（抜粋）を示す。このプログラムでは、降雨データをモニタリングし、一定以上の降雨量（豪雨）を検知すると、周辺の地域から気象データや、交通データ、ソーシャルメディアデータなど、被害に関するセンシングデータを追加で収集する。以下に、プログラムを構成する主要なルールについて説明する。

① state 節では、DSN で用いられる情報サービスの宣言が行われる。@xxxxx は、各々のサービス名を示し、discover 関数によりサービス発見が行われる。

② scratch_xxxxx は各サービスの入出力を表し、channel_xxxxx は、サービス間のデータ転送チャネルを表す。“<~” は、右辺から左辺へのデータ送信を表し、例えば “scratch_evwh <~ channel_panda” は、channel_panda（@panda サービスと @ evwh サービス間のデータ転送チャネル）を通じ scratch_evwh（@evwh サービスの入出力）へデータを転送することを示す。また、scratch_panda から送信されるデータは、

表 1　DSN 関数

関数名	機能概要	例
discover	サービス発見	@jma_rainfall: discover（category=sensor, type=rain）
filter	データのフィルタリング	channel_datastore <～ filter（scratch_panda, average_rainfall > 25.0）
cull_time, cull_space	指定された間隔（時間、場所）でデータを間引き	channel_datastore <~ cull_time（scratch_panda, 1, 2, time（time, "2015 /01 /01 T00 :00 :00 ", "2015 /03 /20 T23：59：59 ", 30, "second"））
aggregate	複数のデータを 1 つに集約	channel_name1 <～ aggregate（scratch_panda, aggregate_data, time（time, "2015 /01 /01 T00：00：00 ", "2015 /03 /20 T23：59：59 ", 15, "second"）, space（latitude, longitude, -12.34, -5.67, 34.56, 78.9, 0.1, 0.3））
trigger	イベント検出	event_heavyrain <+ trigger（channel_datastore, 30, count > 130, average_rainfall > 25.0）

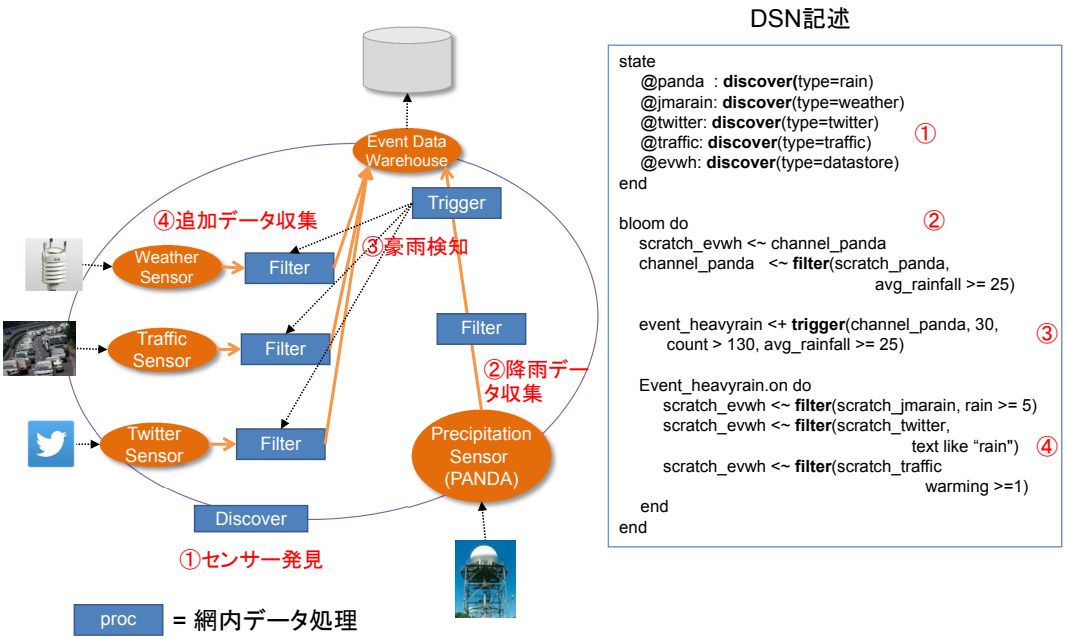

図 4　DSN による情報フローの記述例

channel_panda を通る際、filter 関数により引数に指定された条件でデータのフィルタリングが行われる。

③ event_xxxxx は、ある条件を満たすデータが転送された際に発生するイベントを表し、trigger 関数は引数に示す条件を検知した時にイベントを発生させる。

④ event_xxxxx on do 節は、対応するイベントが発生した時に実行される。

DSN 記述を解釈し実行するインタプリタは、(1) DSN 記述に沿ったルール実行、(2) サービス間のデータ送受信、及び (3) NCPS 機能の呼び出しを行う。DSN インタプリタは、Bloom プログラミング言語の分散データフロー実行エンジン (Bud) に基づいて実装されており、上記のルール実行とサービス間のデータ送受信は Bud によって実行される。Bloom を拡張した DSN 関数は、Bloom コードが埋め込まれている Ruby プログラムのサブルーチンとして実装されており、外部ライブラリを経由して NCPS 機能を呼び出すようになっている。例えば、discover 関数では、引数からサービス検索のためのクエリを生成し、NCPS のノード発見機能を呼び出す。他のデータ処理関数も、同様にして NCPS の対応する網内データ処理機能を呼び出す。

4.2　Network Control Protocol Stack

NCPS では、DSN に基づくオーバーレイネットワークの動的構成を実現するために、ノード発見、パス生成、網内データ処理の各機能を実装している。まず、ノード発見機能では、NCPS が、指定された条件を満たすサービスを検索し、その実行ノードを特定する。ノード発見は、サービス検索とノード探索 (ルックアップ) の 2 つ処理から構成される。サービス検索では、サービスの種類や名前、入出力データ形式など、サービス属性を記述したメタデータに対し、クエリ検索を行う。例えば、“type=rain” というクエリに対しては、降雨データを提供するサービスが検索される。一方、ノード探索では、サービス識別子とノード識別子 (通常は IP アドレス) の組み合わせを NCPS に登録し、サービス識別子をキーとしたハッシュテーブルを作成する。ノード発見の際は、まずサービス検索処理で、指定されたクエリにヒットするサービスメタデータを検索し、次にメタデータからサービス識別子を取得しノード探索処理によりノードを特定する。このように 2 つの処理を組み合わせることで、アプリケーションから様々なクエリを使って柔軟にサービスを検索できるとともに、サービスのノードへの配置が頻繁に変更されても高速にノードを探索することができる。さらに、ネットワークが大規模化した際は、分散ハッシュを使ったノード探索 [24] によりスケーラビリティを高める。

パス生成機能では、サービスを実行するノード群をつなぐ最短経路発見を行う。ネットワークを重み付け有向グラフで表現し、トラフィック量とノード負荷に基づく重み付けを行い、経路発見を行う。一旦生成したパスに対し、サービスノード間のトラフィック量をモニタリングし、DSN の channel に指定されたスループット条件を満たさなくなった場合、NCPS 内でトリガーを発生させ、パスの再計算処理を行う。オーバーレイネットワークごとにパスの設定とトラフィック統計の取得を行う必要があるため、対象となるネットワークのプロトコルごとにこれらの処理を実装する。OpenFlow の場合、パスの情報は OpenFlow コントローラのフローテーブルに設定され、フロー識別子によって制御下にある OpenFlow スイッチの接続が一元管理される。また、OpenFlow スイッチを流れるトラフィック量が OpenFlow コントローラによってフロー識別子ごとに集計され、標準装備されている Stats Request/Replay 機能を使って統計情報を取得する。

上記で生成されたパスを経由して、DSN で指定された情報フローに沿ったデータのやり取りを効率的に実行できるようにするため、網内データ処理機能では DSN のデータ処理関数 (filter, cull, aggregate, trigger) を経路途中のノード上で実行する。NCPS は、各ノードの負荷情報をモニタリングし、経路上のどのノードで処理を行うかを決定する。

5　SCN の応用

SCN によるネットワーク制御の効果を示すため、SCN を使用する場合と使用しない場合で、サービス間のデータのやり取りの性能がどのように変化するか、比較実験を行った。実験では、サービス間で異なるスループットでデータ送受信を行うアプリケーションを、時間の経過と共に次々と実行した際、各アプリケーションのスループットがどのように変化するかを調べた。ネットワークテストベッド StarBED[25] 上に OpenFlow の実験ネットワークを構築し、6 台の OpenFlow スイッチを 10 Mbps の回線によってフルメッシュ構造で接続し、これらを 1 台の OpenFlow コントローラで制御する。また、各 OpenFlow スイッチには、情報サービスが動作するノード (サービスノード) を 2 台ずつ接続する。OpenFlow スイッチと OpenFlow コントローラはソフトウェアとして実装され、それぞれ Open vSwitch[26] と NOX[27] のソフトウェ

表 2　実験ノードの構成

	CPU	メモリ	OS
Open Flow コントローラ	Xeon 2.668 GHz × 4	2048 Mbyte	Ubuntu 11.10
Open Flow スイッチ	Xeon 2.668 GHz × 1	8192 Mbyte	
サービス ノード	Xeon 2.668 GHz × 1	8192 Mbyte	

アを Linux OS（Ubuntu）が稼働するサーバ上で実行する。表 2 にそれぞれの仕様を示す。

実験結果を図 5 に示す。このグラフでは、縦軸がスループット、横軸が経過時間、各折れ線グラフが個々のアプリケーションのスループット変化を表しており、水平な線が長く続くほど要求したスループットでのデータ送受信が安定して行われていることを示している。SCN 有りの場合では、アプリケーションが次々と実行され、様々に異なるスループットでのデータのやり取りが増えても、個々のアプリケーションの性能は安定して維持し続けているのに対し、SCN 無しの場合では、ある時点からスループットが乱れ始め、アプリケーションの性能は不安定になる。これは、SCN が、アプリケーションごとにサービスノード間の end-to-end でのスループットが維持されるよう、ネットワークの輻輳を回避しパスを動的に切り替えているためで、SCN のないベストエフォート型のネットワーク制御に比べ、より多くのアプリケーションをより長い間、安定稼働させられることが分かる。

SCN によるネットワークの動的構成により情報フローの安定的な実行が可能になる特性を生かし、非常時など、突発的に発生する事態に対し、センサやソーシャルメディアなど様々な情報源から、実世界の状況をセンシングしたデータを集中的に収集するセンシングデータ収集解析プラットフォームを、センサネットワークテストベッド JOSE（Japan-wide Orchestrated Smart/Sensor Environment）[28] 上に開発している。JOSE は、カスタマイズが可能な大規模ネットワーク、サーバ設備、無線センサ設備から構成され、広域に配備された大量のセンサから得られる観測データを、高速ネットワークで結ばれた分散拠点上の分散計算機を用いて利活用するテストベッドである。SCN は、JOSE 上で、アプリケーション専用のセンサデータ収集ネットワークを自動的に設定する。図 4 に示したアプリケーションを、実際にこのテストベッド上で動かしデータを収集している様子を図 6 に示す。この図は、

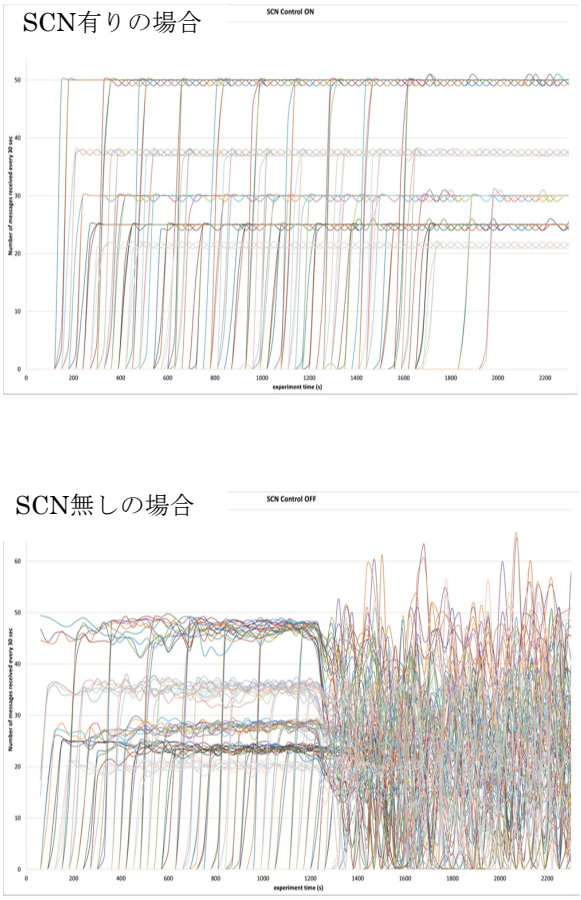

図 5　スループット変化の比較実験

地理空間（平面）と時間軸（縦軸）から成る 3 次元空間に、アプリケーションによって収集された様々なデータを可視化した画面を示しており、左右の画面はそれぞれ SCN 有の場合と無しの場合に対応している。いずれの画面にも、中央には 1 時間に 25 mm 以上の強い雨を観測した時の降雨センサのデータが表示されており、さらにこれらの豪雨発生時に、豪雨地域の周辺から収集された気象データや交通情報、及びソーシャルメディアデータが一緒に表示されている。また、図 6 の左下に表示された小さなウィンドウは、SCN の動作状況を可視化したもので、DSN の trigger 関数で指定したイベント（豪雨）が発生したりデータ転送の遅延が発生した際に、NCPS が自動的にネットワークの再構成を行う様子をリアルタイムにモニタリングできるようになっている。実際のセンサデータ収集の性能を、SCN を用いた場合と用いなかった場合で比較すると、SCN を用いない場合（右）は、降雨センサの値とは無関係に全国から全てのデータを収集し続けるが、SCN を用いている方（左）は、trigger などの DSN 関数により、豪雨が発生した時に周辺地域のデータを集中的に収集する。SCN がイベント発生時にネットワーク資源を弾力的に確保しスループットを維持す

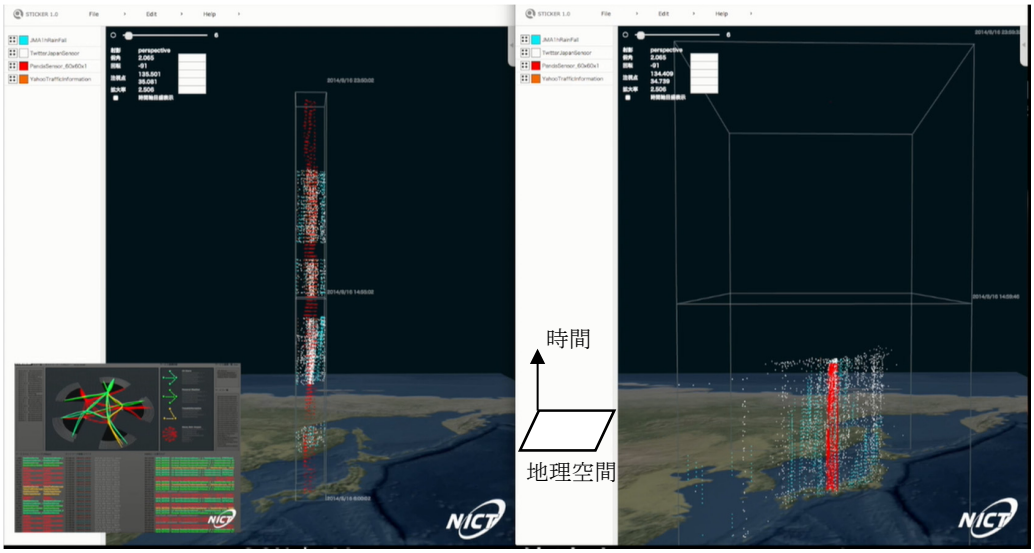

図６　センシングデータ収集解析プラットフォームでのアプリケーション実行例

るため、ほぼ遅延なくリアルタイムなデータ収集が可能になっている。一方、SCN 無しの場合、全国のデータを常に集め続けるため輻輳が発生しやすく、しかもベストエフォート方式のため、図５で示されたように、一旦輻輳が発生するとスループットを回復させるのに時間がかかり、データ収集に大幅な遅延が発生してしまう。

6　今後の展望

IoT の進展により、情報機器やモノ、人々が提供する情報サービスをネットワーク上で様々に連携させるアプリケーションが今後増えるに伴い、SCN のような、サービス間の情報フローに応じてネットワークを動的に構成する技術の重要性は益々高まってくると考えられる。そうした中､SCN には､今後更に高いスケーラビリティが求められる。サービスインタフェースの共通化や DSN 記述の標準化、機械学習等による情報フローとネットワーク構成の効果的な対応付けと能動的な制御、モバイルノード上で稼働するサービスの発見や追跡などが、今後取り組むべき主要な課題として挙げられる。SCN の技術は、ソーシャル ICT の下で、常時・非常時における分野横断的な参加型センシングによる環境問題対策などへの応用展開が進められており、今後 IoT データ利活用基盤としての発展を目指す。

【参考文献】

1　Presser M., Vestergaard L., and Ganea S., “Smart City Use Cases and Requirements,” D2.1, FP7-SMARTCITIES-2013, available at http://www.ict-citypulse.eu/page/sites/default/files/citypulse_d2.1_requirements_v1.0_0.pdf
2　OpenDaylight, http://www.opendaylight.org/
3　Numhauser, B.-M., et al., “Fog Computing Introduction to a New Cloud Evolution. Escrituras silenciadas” paisaje como historiografía. Spain: University of Alcala. pp.111–126. ISBN 978-84-15595-84-7, 2013.
4　Toyomura T., Kimata T., Kim K.-S., and Zettsu K., “Towards Information Service-Controlled Networking,” Proceedings of the 5th International Universal Communication Symposium (IUCS2011), pp.155–163, Oct. 2011.
5　木全 崇, 豊村 鉄男, 金 京淑, 是津 耕司, “Service-Controlled Networking に基づくネットワークフローの動的制御手法,” 電子情報通信学会技術研究報告, Vol.112, SC2012-2, pp.7–12, 2012 年 6 月.
6　木全 崇, 杉浦 孔明, 董 冕雄, 是津 耕司, “Service-Controlled Networking,” アプリケーション構造やユーザ要求に連動したネットワーク制御技術, 第 6 回データ工学と情報マネジメントに関するフォーラム (DEIM2014), C9-3, 2014 年 3 月.
7　Dong M., Kimata T., and Zettsu K., “Service-Controlled Networking,” Dynamic In-Network Data Fusion for Heterogeneous Sensor Networks, Proceedings of the 33rd IEEE International Symposium on Reliable Distributed Systems Workshops (SRDSW), Nara, Japan, pp.94–99, Oct. 2014.
8　Future networks “Objectives and design goals,” ITU-T Y.3001, May, 2011.
9　Loo B. T., et. al.: Declarative Networking, Communications of the ACM, Vol.52 No.11, pp.87–95, Nov. 2009.
10　Ramakrishnan R. and Ullman J.D., “A Survey of Research on Deductive Database Systems. Journal of Logic Programming,” Vol.23, No.2, pp.125–149, 1993.
11　Nigam V., Jia L., Wang A., Loo B. T., and Scedrov A., “An Operational Semantics for Network Datalog,” Proceedings of the Third International Workshop on Logics, Agents, and Mobility (LAM), July 2010.
12　Loo B.T., Condie T., Hellerstein J.M., Maniatis P., Roscoe T., and Stoica I., “Implementing declarative overlays,” Proceedings of ACM Symposium on Operating Systems Principles, 2005.
13　P2, “Declarative Networking System” http://p2.cs.berkeley.edu.
14　Foster N., et. al., “Frenetic: A Network Programming Language,” Proceedings of the 16th ACM SIGPLAN international conference on Functional programming (ICFP '11), pp.279–291, 2011.
15　Monsanto C., Foster N., Harrison R., and Walker D., “A Compiler and Run-time System for Network Programming Languages,” Proceedings of ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), pp.217–230, Jan. 2012.
16　Canini M., Venzano D., Peres P., Kostic D, and Rexford J., “A NICE Way to Test OpenFlow Applications,” Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation

(NSDI'12), pp.127–140, 2012.
17 Voellmy A. and Hudak P., “Nettle: Functional Reactive Programming for OpenFlow Networks,” Proceedings of the Workshop on Practical Aspects of Declarative Languages pp.235–249, 2011.
18 Voellmy A., Kim H., and Feamster N., “Procera: A Language for High-Level Reactive Network Control,” Proceedings of the First Workshop on Hot Topics in Software Defined Networks (HOTSDN'12), pp.43–48, 2012.
19 Ahlgren B., Dannewitz C., Imbrenda C., Kutscher D.,and Ohlman B., “A Survey of Information-Centric Networking, IEEE Communications Magazine,” Vol.50, No.7, pp.26–36, July 2012.
20 Jacobson V., et. al., “Networking Named Content,” Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '09), pp.1–12, 2009.
21 AMON data format, http://amee.github.io/AMON/ .
22 BOOM, http://boom.cs.berkeley.edu/
23 Alvaro P., Conway N., Hellerstein J. M., and Marczak W. R., “Consistency Analysis in Bloom: a CALM and Collected Approach,” Proceedings of the 5th Conference on Innovative Data Systems Research (CIDR '11), pp.249–260, 2011.
24 Stoica I. et. al., “Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications,” ACM SIGCOMM Computer Communication Review, Vol.31, No.4, pp.149–160, 2001.
25 StarBED, http://starbed.nict.go.jp/
26 Open vSwitch, http://openvswitch.org/, accessed at June 2015.
27 NOX OpenFlow Controller, http://www.noxrepo.org/, accessed at June 2015.
28 大規模オープンテストベッド JOSE, http://www.nict.go.jp/nrh/nwgn/jose.html .

是津耕司　（ぜっつ　こうじ）
ユニバーサルコミュニケーション研究所情報利活用基盤研究室室長／ネットワーク研究本部ネットワークシステム総合研究室研究マネージャー
博士（情報学）
データ工学、データベース、情報検索