修士論文

センサ・アクチュエータ間のデータフロー展開における
ネットワーク機器の制御システム
Network Equipment Management System in Data Flow
Deployment between Sensors and Actuators

平成 24 年 2 月 8 日提出

指導教員 江崎 浩 教授

東京大学大学院 情報理工学系研究科
電子情報学専攻

学生証番号：106405　氏名：川口 紘典

概要

近年のビルディングオートメーションシステムにおいては，ネットワーク機能を持ったさまざまなデバイスがユビキタスに設置され，それらが協調動作するようなシステムが展開されようとしている．ビルディングオートメーションシステムの基本的な動作は，センサから取得した値を分析・加工などのデータ処理を施してアクチュエータを制御するというものである．本研究では，センサ・アクチュエータデバイスの制御とデータ処理のアプリケーションを分散配置されたエンドホストで実行するようなシステムの実現に焦点をあてた．デバイス同士の協調動作を自律分散的に，かつデータフロー的に実現するためにはエンドホスト上のアプリケーション間で，適切なデータフローを生成・展開する必要がある．しかし，実際のシステム環境においては，NAT, ファイアウォール，Application Level Gateway ルータなどのさまざまなネットワーク機器が存在しており，エンドホストで実行されるアプリケーションの設定だけでなくデータフロー上のネットワーク機器の設定も必要となる．実運用の観点から考えると，データフロー展開に合わせたネットワーク機器を手動で設定することは，管理コストが大きく柔軟なセンサアプリケーション展開の障壁となるだけでなく，誤設定によるシステムの不具合の発生やそのトラブルシュートの負荷が無視できないくらいに大きくなってしまうことが多い．本研究では，デバイスの協調動作のためにデータフロー展開を行う際のネットワーク機器の管理制御に着目し，データフロー展開に必要なエンドホストのアプリケーションとネットワーク機器の設定を統合的に扱い設定を行うことで，柔軟で誤設定のないセンサアプリケーション展開を実現することを目指した．本研究で提案するシステムアーキテクチャは中央管理方式を採用しており，中央のコントローラが，分散的に実行されるアプリケーション間で通信を行うために必要となるアプリケーションとネットワーク機器の設定を計算し制御する．プロトタイプシステムを用いた実験により，提案手法のアーキテクチャを用いることでネットワーク機器が存在する環境下でもデータフローが柔軟なデータフロー展開が可能となり，デバイス同士が協調動作できたことを確認した．

目次

図目次

第 1 章

序論

1.1 背景と目的

近年の経済の発展に伴うエネルギー消費量が増加している状況に対し，ICT を活用したグリーン化 (Green by ICT) によるエネルギー消費量の削減が期待されている．Green by ICT により，企業や一般家庭での活動に際して，ICT を用いて環境情報の計測及び予測を行いつつ，エネルギー利用効率の改善，物の生産・消費の効率化・削減，人・物の移動の削減につなげることでエネルギー消費量を削減し，CO2 の排出量を削減することが可能である [1]．このような技術の適用例として，ビルディングオートメーションシステム (BAS) によるエネルギー利用効率の改善が挙げられる．BAS とは，ビル内に設置されている電気，空調などの設備を統合的に制御，監視，管理するシステムで省エネルギー化や環境の快適化が図ることを目的としており，展開されるシステムの基本動作は，温度センサ，人感センサ，照度センサ，カメラといった周囲の環境情報を取得するセンサデータに対してデータ処理を行い，そのデータに基づいてディスプレイ，スイッチ，ブザーなどの周囲の環境へ何らかの影響を及ぼすアクチュエータを制御するというものである．展開されるシステムの動作例としては，温度に基づいた空調の自動制御や，人感センサによる照明の自動点灯，消灯などが挙げられる．

近年の BAS は，ビル内の既存の TCP/IP ネットワーク上に展開してインターネット技術を用いることで [2, 3]，取得したデータの保存，分析を行うサーバと連携させて運用し外部のネットワークから制御監視を行う傾向にあるが，ビル内の TCP/IP ネットワークは BAS 展開に適さず，BAS を展開することが難しい場合がある．BAS の基本的な動作は，センサデータを分析，加工してその値に基づいてアクチュエータを制御するというものであるため，展開されるシステムには，センサからアクチュエータへの一連のデータフローが含まれる．このようなシステムの場合，web サービスなどクライアントサーバ型のサービスとは異なりあるネットワークセグメントからアクチュエータなどへの能動的なアクセスが必要となるが，IPv6 の知識不足やグローバル IPv4 の枯渇の問題から [4]，現在でも外部からのアクセスが難しい Network Address Translator (NAT) によるプライベートアドレスを用いたネットワークが構築されている．NAT 以外にも，ファイアウォール (FW) や Application Level Gateway (ALG) などのアクセス制御によるネットワーク機器のために BAS のシステム展開が難しい場合があるが，これらのネットワーク機器の設定は BAS の展開に伴うデータフローに合わせて手動で設定されているのが現状である．そこで本研究では，BAS の展開を想定しセンサからアクチュエータへのデータフローを展開する際に必要なネットワーク機器の制御の自動化を行うことで，システムの管理コストを削減することを目的とする．本研究で想定している展開されるシステムは，センサ・アクチュエータデバイスの制御を行うアプリケーションと，センサデータに対しデータ処理を行うアプリケーションを分散配置されたホストで実行し，ア

プリケーション間のデータのやり取りを行うことで，データフローを展開するものである．

ネットワーク上にデータフローを展開の際に必要なネットワーク機器の設定を自動的に行うものに，OpenFlow[5] が挙げられる．OpenFlow は，物理層からトランスポート層の任意の組み合わせで通信制御が行えるもので，この組み合わせの単位をフローと呼ぶ．そして中央コントローラにフローの単位で通信制御を定義し，その定義に基づいて分散配置されたスイッチのフローテーブルの設定を行う．OpenFlow はトランスポート層以下のみを扱っており，センサ・アクチュエータ制御，データ処理などの分散展開されるアプリケーションを想定していない．

センサ・アクチュエータ制御，データ処理アプリケーションを分散展開することで，センサからアクチュエータのデータフローを展開するものに CCDM[6] がある．CCDM は，センサ・アクチュエータ制御とデータ処理などの分散展開されるアプリケーションのみを扱い，データフロー上のネットワーク機器の存在を想定していない．

NAT による外部からの能動的なアクセスの難しさを解決する NAT 越えの技術については，STUN[7], TURN[8], UPnP[9], ALG[10] が挙げられる．これらは NAT によるプライベートネットワーク内に配置されるクライアント同士を P2P で接続するためのものであり，本研究が想定するようなデータ処理を含むアプリケーションをネットワーク上で分散展開する場合のような複雑なデータフロー展開への適用は難しい．

OpenFlow のように，管理者がトランスポート層以下の通信制御を定義することでネットワーク機器の自動設定するものと，CCDM のように，デバイス制御，データ処理のアプリケーションを分散的に実行することでデバイス間のデータフローを展開する研究はすでになされているものの，本研究が対象とするようなデバイスの制御，データ処理などのアプリケーションが分散実行されることを想定した上で，アプリケーション間のデータフロー上に存在するネットワーク機器の設定について扱う研究はされてない．本研究では，センサ・アクチュエータ制御とデータ処理アプリケーションが分散展開される状況下で，センサからアクチュエータへのデータフロー展開に必要なネットワーク機器の自動設定を行うことで，ネットワーク機器が存在する環境下でも良好にセンサネットワークアプリケーションを展開することを目的とする．提案するアーキテクチャにより，管理者はネットワーク機器の存在を意識せず，アプリケーションレベルでデバイス制御できるようになることを実験により示す．

1.2　本論文の構成

本論文の構成は以下のようになっている．第 2 章では，本研究が対象とする問題点を明確にし，第 3 章では，関連研究・技術について紹介する．第 4 章では，提案するシステムのアーキテクチャについて述べ，第 5 章では，第 4 章で述べたアーキテクチャに基づいたシステムの動作について述べる．第 6 章では提案したアーキテクチャの動作検証の結果について述べ，第 7 章で考察を行い，第 8 章で結論を述べる．

第 2 章

データフロー展開における問題点

本研究が対象とする BAS で展開されるシステムと，対象とするネットワーク環境，そして本研究が焦点を当てるシステム展開における問題点について述べる．

2.1 本研究で対象とする展開されるシステム

本研究で具体的に展開されるシステムについて述べる．BAS は，センサから取得したデータを分析，加工などのデータ処理を施し，その値に基づいてアクチュエータを制御するものが基本的な動作となり，本研究では，データ処理が予め決められたホストではなく，任意のホストで動作することを想定する．本研究で展開されるアプリケーションは，図 2.1 のような複数のセンサ (この図では S1, S2) から値を取得し，その値をあるホストでデータ処理を施し，その値に基きアクチュエータ (この図では A1) の動作を制御するというセンサからアクチュエータへ一連のデータフローが展開されるものを想定している．

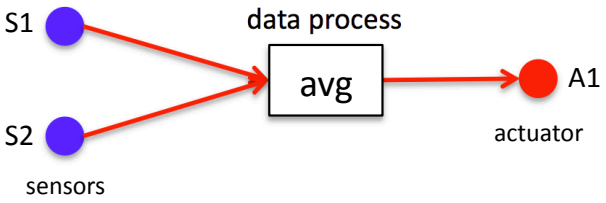

図 2.1 展開されるアプリケーション例

図 2.1 は 2 つのセンサ S1, S2 から平均値 (avg) を計算し，アクチュエータ A1 へ計算結果を出力していることを示している．図 2.1 のような動作は，図 2.2 のようにエンドホストへデバイス制御とデータ処理をおおなうアプリケーションが分散的に展開されることで実現される．

この例のような，センサからアクチュエータへのデータフローを展開のために，各ホストへアプリケーションを分散展開する際には本研究では CCDM の機能を用いた．

2.2 ネットワーク機器によるデータフロー展開の障壁

今回想定しているシステム展開の際は，あるネットワークから他のネットワークセグメントへ能動的にアクセスを行う必要がある．しかし，NAT によるプライベートアドレス空間のネットワークが存在する場合，外部ネットワークからプライベートアドレス空間へデータを転送するためには，NAT ルータの

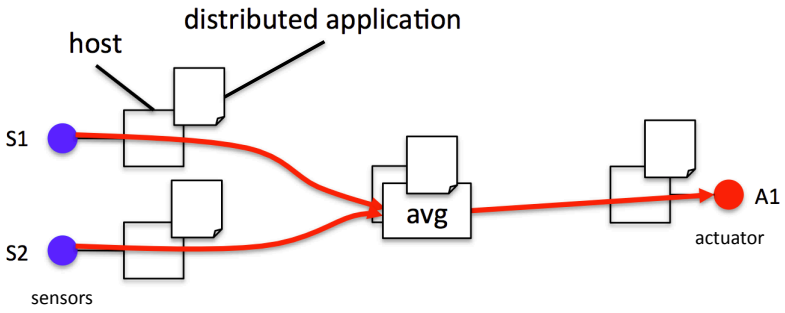

図 2.2　アプリケーションが分散展開されてデバイス間のデータフローが展開される様子

ポートフォワードの設定が必要である．図 2.3 は図 2.1 のような動作を実現させたいものの，プライベートアドレス空間の中にアクチュエータが存在していることで，アクチュエータへデータが転送できない様子を示している．

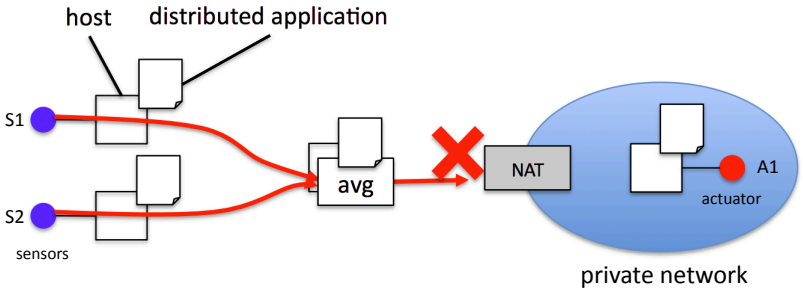

図 2.3　NAT によりデータフローが展開できない様子

また，FW，ALG などのアクセス制御によりルータがデータを転送せず，データフロー展開ができないケースも考えられる．

2.3　本研究が対象とする問題点

本研究では，2.2 で述べた問題に対し，ホストとネットワーク機器を統合的に管理することで，ホストで実行されるアプリケーションの設定と，アプリケーション間のデータフロー上にある NAT, FW, ALG といったネットワーク機器の設定を自動的に行うことで，管理者がネットワークトポロジやネットワーク機器を意識しなくてもセンサからアクチュエータへのデータフロー展開が可能にすることを目的としている．各ホストの動作を決定するアプリケーションの分散展開は CCDM の機能を用い，本研究は，ホスト間のデータフロー上にあるネットワーク機器の設定を対象とする (図 2.4)．

CCDM の機能に加えてネットワーク機器がデータフロー展開に併せて自動的に設定されると，センサからアクチュエータのデータフロー展開に必要なトランスポート層以下の設定は全て自動的に設定されることになる．それに伴い管理者は，デバイスの制御とセンサから取得されるデータに対するデータ処理を記述したアプリケーションレベルでのネットワークトポロジに依らないデータフロー定義が可能となる．これにより，デバイスを識別子や属性といった抽象的な表現で記述することが可能となるため，デバイスの状態変化に対しても柔軟な動作が可能なアプリケーションを展開することが可能となる．具体的に本研究で想定したデバイスの状態変化は，デバイスの移動，追加，削除である．以下，これらのデバイスの状

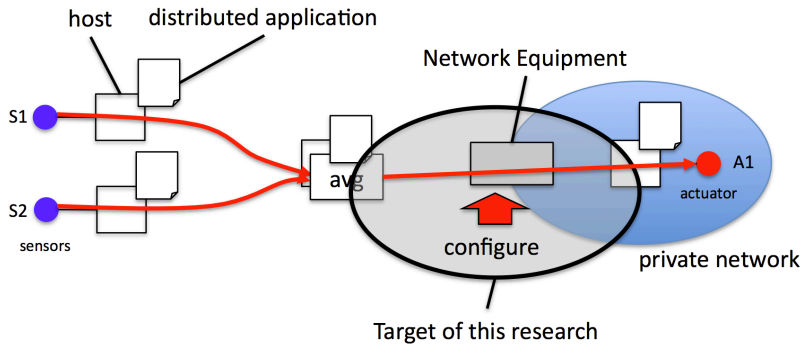

図 2.4　本研究の対象

態変化が起こった際に生じるデータフローの変化やネットワーク機器の設定の必要性を述べる．

- デバイスの移動
  デバイスが他のネットワークセグメントへ移動した際に，データフローも同様に変化する．例えば管理者は，「S1 という識別子が付けられたセンサの値を A1 という識別子が付けられたアクチュエータへ出力する」というアプリケーションを記述しているとする．この際，A1 が NAT によるプライベートネットワーク空間にあった場合には，その NAT には A1 へのポートフォワードの設定が必要となり，S1 のデータの送信先もそれに応じて設定する必要がある．ここで，A1 が他の NAT 以下のプライベートネットワークへと移動した場合，それに応じて NAT を再設定する必要があり，S1 のデータの転送先もそれに合わせて再設定する必要がある．このように，デバイスがネットワーク空間を移動した際には，ネットワーク機器の再設定と，データの転送先の両方を再度設定する必要がある．
- デバイスの追加，削除
  BAS で展開されるアプリケーションには，デバイスに属性を設定しておいて同じ属性のデバイスに一括処理を行うというものが考えられ，例えばある属性のセンサの全ての値の平均値をある属性が設定されたアクチュエータ全てへ出力する，というものである．具体的な例としては，room1 に設置された”room1_temperature”という属性が設置された任意の個数の温度センサの平均値を，”room1_display”という属性が設置されたディスプレイ全てに表示させる，という動作が挙げられる．このようなアプリケーションの場合，センサやアクチュエータの数に応じてデータフローやデータ処理プログラムは変更しなければならないが，例えば新しく温度センサを導入し増やすといったことや，デバイスの故障によりシステムから取り除くといった，デバイスの追加，削除は頻繁に発生しうる．また，デバイスの追加に伴う新しく形成されたデータフロー上のネットワーク機器の設定も必要なる．そのため長期継続的にシステムを運用するには，このような変更に対しても継続動作が可能なアプリケーションインタフェースを管理者に提供する必要がある．

本研究の提案するシステムで，デバイスの操作がアプリケーションレベルで記述できるようになり，前述のデバイスの移動，追加，削除といった状態変化に柔軟に対応できるようになることを，実験によって検証する．

第 3 章

関連研究

3.1 CCDM

CCDM は，多種多様なデバイスがインターネットに接続されている現状に対して，それらのデバイスを適切に継続動作させることが可能な管理アーキテクチャを確立させることを目的としている．キャンパスネットワークや企業のネットワークなどの大規模かつ複雑なネットワークを想定し，デバイスはセンサ，アクチュエータを対象としている．

CCDM が前提としている状況は以下のよう．

- キャンパスやオフィスビルなどの生活空間に多数のデバイスが存在する
- デバイスはインターネットへの接続性があるものとする
- センサから収集したデータに対して，集約，解析などのデータ処理を行うノードが存在する
- デバイスは様々なプロトコルを持つ

このような状況に対して，多数存在するプロトコルが異なるデバイスの管理が複雑であること，ネットワークの管理は煩雑で間違いを起こしやすいものであることを問題点として，それらの管理を簡素化，自動化することを目的としている．

アーキテクチャは中央管理方式を採択している．システムの構成要素は，センサやアクチュエータなどのデバイス，センサから取得したデータに対して処理を行い，デバイスを制御する分散配置されたノード，ノードの動作を制御する中央コントローラがある．ネットワーク管理者は中央コントローラで提供されるノードやデバイスの情報が記述されているリソーススクリプトと，デバイスの操作やデータ処理を記述するサービススクリプトを用いて，デバイスの操作を行う．リソーススクリプトは，ノードの IP アドレス，使用可能なポート番号や，デバイス毎のプロトコルが記述されている．サービススクリプトは，高級言語によるネットワーク汎用 I/O 制御とプロトコル翻訳機構 [11] で提案されている四則演算，条件分岐などの命令セットに加え，CCDM で提案されているデバイスの操作のための命令セットにより記述する．提案されたデバイス操作のための命令セットを用いて，リソーススクリプトで記述されているデバイスのプロトコルを隠蔽することにより，ネットワーク管理者は下位レイヤを考慮する必要なくデバイス制御を記述でき，デバイスの管理が簡素化される．また，リソーススクリプトとサービススクリプトが分離されることにより，ネットワークの状態更新時やサービスの変更，追加が容易となる．上記のアーキテクチャにおいて，リソーススクリプトとサービススクリプトの情報から，中央コントローラが分散配置されているノードの処理内容を適切に割り当てることで，デバイスの制御を行う．

CCDM が掲げる貢献は以下のものとしている．

1. 中央管理方式によるアーキテクチャのデバイス管理への適用
2. デバイスの管理に適した命令セットの定義
3. デバイスの管理のための命令セットを分散的に実行するための命令セット分割アルゴリズムの提案

本研究では，CCDM で提案されているデバイス管理のための命令セットを分散的に実行するための命令セット分割アルゴリズムを用いている．CCDM は，分散配置されているデバイスの管理を目的としているが，本研究では分散配置されたアプリケーション間のデータフロー展開に必要なネットワーク機器の設定を対象としている．

3.2　OpenFlow

OpenFlow は，普段使用するネットワークに実験的なプロトコルを実行させる方法を研究者に提供する．OpenFlow は，イーサネットスイッチの内部フローテーブルのエントリを追加，削除するためのインタフェースを規定し，ベンダの製品に OpenFlow のインタフェースを加えさせることで，キャンパスのバックボーンなどにも OpenFlow を用いたネットワークを展開できるようにすることを目的としている．OpenFlow により，研究者が様々なスイッチ上でも決まった手順により，各スイッチがラインレートで動作させることが可能となり，さらに，各ベンダはスイッチの内部構造を公開する必要がない．研究者が実験的なプロトコルを実際のネットワークで動作させられることに加えて，キャンパスネットワークのような大規模なネットワーク上でテストベッドを形成することも可能であるとしている．OpenFlow の機能を搭載している製品は既に製造され，実験的に運用されている例も存在する．

現在広く利用されているのルータは，機器内に経路計算処理とパケット転送の処理が一緒に組み込まれており，複数のルータが OSPF などのルーティングプロトコルを用いて自律分散的に協調動作をして適切なパケット転送処理を行う．転送処理はパケット単位で処理されるため，経路情報が書き換わらない限り経路が変更されることはない．一方，OpenFlow を用いたネットワークは OpenFlow コントローラと OpenFlow スイッチと呼ばれるもので構成されており，経路計算処理を OpenFlow コントローラへと分離し，OpenFlow スイッチは OpenFlow コントローラの経路計算結果による指示通りにパケット転送処理を行う．トラフィックの転送処理は従来のパケット転送ではなくフローと呼ばれる単位で処理が行われ，フロー単位で記述されたユーザプログラムにより，トラフィックの転送処理を制御することが可能となる．

OpenFlow におけるフローとは，物理層，データリンク層，ネットワーク層，トランスポート層の組み合わせのにより定義されるもので，フローの識別に可能なものは物理ポート，MAC アドレス，IP アドレス，ポート番号など 12 種類が存在する．ユーザはフロー識別子を用いて，各フロー内の情報を「ルール」として，そのフローに対する処理内容を「アクション」として OpenFlow コントローラのフローテーブルへと予め記述しておく．そして各 OpenFlow スイッチはパケットを受信すると，まずスイッチ内のフローテーブルを参照し，適用すべきルールが存在するか確認を行う．フローテーブルに該当するルールが存在するとアクションに従い転送処理を行い，存在しない場合には，OpenFlow コントローラへ適切なルールとアクションを問い合わせる．問い合わせを受けた OpenFlow コントローラは，ユーザが定義したフローテーブルに従い，各 OpenFlow スイッチの転送処理を計算し，OpenFlow スイッチへ指示する．OpenFlow スイッチは，一度 OpenFlow コントローラへ問い合わせて得たルールとアクションは各スイッチのフローテーブルへと登録し，次回同じルールに合致するフローに関しては，OpenFlow コントローラへ問い合わせることなく，転送処理を行う．

表 3.1　NAT の分類

NAT の種類	マッピングの有効範囲	対称性
Full Cone NAT	NAT のマッピングは全ての外部アドレスから有効	非対称
Restricted NAT	NAT のマッピングは，そのマッピングが生成された宛先と同じアドレスに対して有効	非対称
Port Restricted NAT	NAT のマッピングは，そのマッピングが生成された宛先と同じアドレスとポート番号に対して有効	非対称
Symmetric NAT	NAT のマッピングは，そのマッピングが生成された宛先と同じアドレスとポート番号に対して有効	対称

OpenFlow を用いることにより，トラフィックに応じてフローを変え，ネットワークのリソースを動的に割り当てることや，物理構成とは異なる論理構成のネットワークを構築することが可能になることが OpenFlow ネットワークを用いる利点として挙げられる．

3.3　NAT Traversal

NAT はプライベートアドレス空間とグローバルアドレス空間のネットワークの境界に存在し，プライベート IP アドレスに変換を行うことでグローバルアドレス空間のサーバと通信ができるようにするネットワーク技術である．プライベートアドレス空間からグローバルアドレス空間へパケットが送信される場合には，パケットの送信元の IP アドレスが NAT ルータのグローバル IP アドレスに変換されると同時に，そのプライベート IP アドレスとグローバル IP アドレスの変換対応表が生成される．その後，NAT ルータがグローバル IP アドレス側で受信するパケットの送信先は，変換対応表に従ってプライベート IP アドレスへ書き換えられ，該当するホストへ転送される．

また，IP アドレスの対応だけではなくトランスポート層の TCP，UDP のポート番号も含めた変換を行うものを Network Address-Port Translator (NAPT) といい，近年では，NAT は NAPT を示していることが多い．NAPT では，アドレス変換にポート番号も用いることで，NAT ルータに割り当てられた 1 つのグローバル IP アドレスを複数のプライベート IP アドレスと対応付けを行い，プライベートアドレス空間に存在する複数のホストが同時に 1 つのグローバル IP アドレスを共有することができる．なお，NAT の変換には，表 3.1 のような種類が存在する．

NAT によるプライベートネットワーク空間で用いられるプライベート IP アドレスは，そのネットワーク内のみ通用するアドレスであり，外部ネットワークから直接アクセスできない．これを実現するための NAT 越えの技術に関して，以下 STUN, TURN, UPnP, ALG の 4 点の関連技術を挙げる．なお，これらの NAT 越えのための技術は，NAT ルータ以下に存在する 2 端末間の P2P 通信を可能とすることを目的としているが，本研究では，センサ，アクチュエータ間の直接的な P2P のデータフロー展開だけではなく，データ処理を含んだデータフロー展開を想定している．

3.3.1　STUN

STUN は，プライベートアドレス空間に存在する STUN クライアントがグローバルアドレス空間に設置された STUN サーバと呼ばれるサーバを利用することで，NAT による変換されるグローバル IP アド

レスとポート番号の対応の情報を得る手法である．STUN クライアントが STUN サーバへリクエストを送信すると，サーバは受信したパケットからそのクライアントの NAT によるアドレス変換の IP アドレス，ポート番号の情報を取得する．この IP アドレス，ポート番号を用いれば，その STUN クライアントへアクセスできるということである．STUN クライアントが他のクライアントへ接続する場合には STUN サーバへ問い合わせて，そのクライアントへアクセスを行うための NAT のグローバル IP アドレス，ポート番号を取得し，接続を行う．

STUN は，一般的な NAT ルータのアドレス変換の機能を用いているため，NAT に特殊なソフトウェアを要求しないが，NAT ルータのアドレス変換の種類が Symmetric NAT の場合や UDP 以外の TCP などの別のプロトコルでは使用できない．

3.3.2　TURN

TURN はグローバルアドレス空間にパケットを中継するための TURN サーバと呼ばれるサーバを設置し，TURN サーバを経由することで NAT 内のホスト同士の通信を行う仕組みである．TURN は STUN と異なり，全てのパケットを TURN サーバを使用して中継するため，UDP だけでなく TCP も通信が可能であり，NAT の種類による制限もない．

ただし，TURN はホスト間の通信をすべて中継するため，TURN サーバへのトラフィックが集中し，また通信中は TURN サーバとホスト間でマッピング情報を保持しなければならず，サーバへの負荷が集中する．

3.3.3　UPnP

UPnP は PC や AV 機器，家電などの機器をネットワークを通じて相互に利用するための技術であり，機器同士が相互に通信をするための，機器の発見，機能の情報交換，自動設定を行うためのプロトコルが規定されている．UPnP の NAT に関する機能としては，NAT ルータの自動検出，NAT ルータへの IP アドレスとポート番号の変換対応の設定要求，NAT ルータのグローバル IP アドレスの取得などの機能がある．UPnP を用いることで，クライアントは NAT ルータのマッピング情報を取得でき，パケットのペイロード中に送信元の IP アドレスやポート番号が必要なアプリケーションにも利用できる．

ただし，UPnP は，NAT と NAT の内側に存在する機器が共に UPnP に対応している必要があり，また，NAT が多段となっていた場合には使用できない．

3.3.4　Application Level Gateway

NAT ルータが Application Level Gateway の機能を持ち，パケットのペイロードを書き換える仕組みである．プライベートアドレス空間のホストからグローバルアドレス空間のサーバへとパケットを送信する際，IP パケットの送信元 IP アドレスと送信元ポート番号を書き換えると同時に，アプリケーションレベルでペイロードの中身を解析，理解し，ペイロード中の送信元の情報も同時に書き換える．サーバからクライアントへの返答のパケットについても，NAT ルータで設定されたアドレス変換表に従い，ペイロードを書き換えてクライアントへ転送される．

ALG はアドレスを変換する NAT ルータ上に設定されていることが前提となっているため，一般的な NAT ルータでは使用出来ず，また，ペイロードを適切に書き換えなければならないため，柔軟性，拡張性に課題がある．

3.4　データストリーム処理

本研究において分散展開されるデータ処理のアプリケーションは，データストリーム処理システムというものに分類される．データストリーム処理とは，予めクエリを発行しておいてデータが発生する度に処理を実行する Data-initiative 型のデータ処理方式で，蓄積されたデータに対してクエリを発行し結果を得る Query-initiative 型のデータベースシステムと異なる [12].

データストリーム処理システムは大きく集中型と分散型に分かれる．集中型のデータストリーム処理システムは，Aurora[13, 14, 15], STREAM[16], NiagaraCQ[17], TelegraphCQ[18] などが挙げられ，これらの研究は主に発生したデータに対する処理の最適化に重点が置かれている．これらの研究は，データを集中的に集めた後の処理について扱い，分散配置されたデバイス間のデータフロー展開については考慮しない．分散型のデータストリーム処理システムは，Stream Spinner エンジン [19] を用いた ORINOCI[20] が挙げられる．これは，管理者が発行したクエリに対する結果を，分散配置されたノードを連携させることで効率よく取得することを目的としており，データフロー中のネットワーク機器の設定については考慮しない．

第 4 章

システムアーキテクチャ

本研究では第 2 章で述べた問題点に対し，中央管理方式のアーキテクチャを用いた手法を提案する．提案手法では, 1 台の中央コントローラがネットワークトポロジと管理者が記述したデバイス制御とデータ処理の定義に基き，分散配置されたネットワーク機器とデバイス，ホストの動作を設定を行う．中央コントローラから全てのネットワーク機器，ホストを制御することでコントロールプレーンを集中的に扱い，センサ，データ処理のアプリケーション，アクチュエータ間のデータの転送は分散配置されたホスト同士行うことで，データプレーンは分散的に実行する．

コントロールプレーンは，データフローを適切に展開するために一貫性に優れている集中管理方式を採用している．集中方式は，負荷集中のためのスケーラビリティに問題があることや，集中点が単一障害になるということが一般的に指摘されるが，中央コントローラからネットワーク機器やホストを一度制御すれば，それ以降中央コントローラにアクセスを行わず継続的に動作するため，コントロールプレーンに集中方式を採用するのは負荷集中，単一障害点となることに関して問題ないと考えられる [6, 5]．一方，データプレーンに関しては分散的に展開することで，センサから取得したデータのトラフィックが一箇所に集中することを避けて単一障害点を設けないことで，あるネットワークに障害が発生してもシステム全体に影響を及ぼすことを防ぐことができる．

以上の理由から，本研究ではコントロールプレーンを集中管理的に扱い，データプレーンを分散展開する，中央管理分散実行のアーキテクチャを採用した．

4.1　システム構成要素

提案手法のシステムは，以下の要素で構成される．

- 中央コントローラ
- ネットワーク機器
- ホスト
- デバイス
- ネットワーク構成スクリプト
- データフロースクリプト
- 設定ファイル

概要図は図 4.1 のようになっており，以下，それぞれの構成要素の役割を述べる．

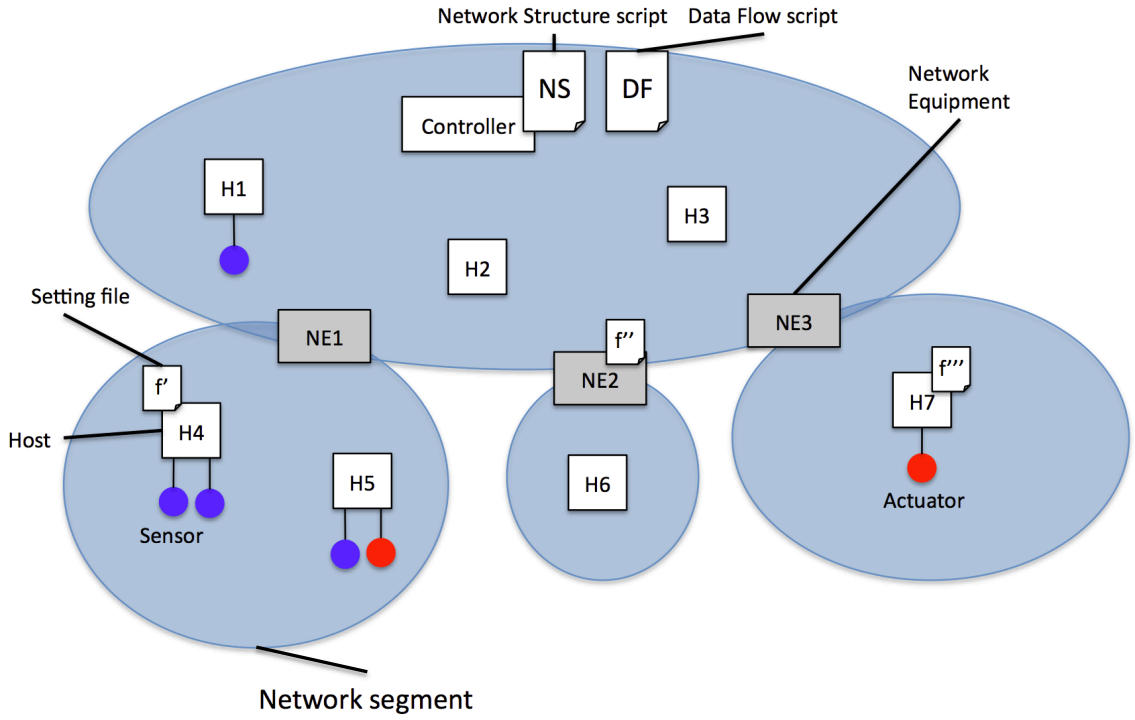

図 4.1　システム構成要素

- 中央コントローラ (図中の Controller)
  システム内で 1 台のみ存在し，ネットワーク機器，ホストの動作を制御する．管理者は中央コントローラが保持するネットワーク構成スクリプトを参照することでネットワーク上に存在するデバイスの存在を知ることができ，その情報を元にデバイスの制御とデータ処理を定義したデータフロースクリプトを中央コントローラへ記述する．中央コントローラはこれらのスクリプトから設定ファイルを生成し，そのファイルを各ネットワーク機器，ホストへ配布してそれらの動作を設定することで，データフロースクリプトで記述した動作をネットワーク上へ展開する．
- ネットワーク機器 (図中の Network Equipment)
  データフローを展開する際，データを通過，転送させるために設定が必要な機器．中央コントローラから配布される設定ファイルに記述された通りに自身の設定を行う．本研究では，ネットワーク機器として NAT，FW，ALG ルータを対象としている．
- ホスト (図中の Host)
  ホストは物理的に分散配置されており，中央コントローラから配布される設定ファイルに基づいてデバイスの制御とデータ処理を行う．デバイスの制御は主にセンサからデータをすること取得や受信をしたデータに基づいてアクチュエータの操作というもので，データ処理とは受信したデータに対する演算を行い，次のホストへデータを転送することを指す．以下，ネットワーク機器とホストをネットワーク構成要素と表す．
- デバイス (図中の Sensor, Actuator)
  センサ，アクチュエータデバイスを指す．デバイスはホストによって制御され，センサはホストからデータ取得のリクエストがあった場合，そのホストへのセンサデータの送信を，アクチュエータはホストからの制御で動作するという単純なものを想定している．センサは周辺の環境情報を取得するデバイスを指し，具体例として温度の情報を取得する温度センサ，人の所在を検知する人感セ

ンサ，照度を取得する照度センサ，周囲の映像を配信するカメラなどが挙げられる．アクチュエータは周辺の環境へ影響を及ぼすデバイスを指し，照明，ディスプレイなどが挙げられる．デバイスには属性が設定されており，データフロースクリプトにおいて同じ属性を持つデバイスに対する一括処理を定義するために用いられる．

- ネットワーク構成スクリプト (図中の Network Structure script)
  システム全体のネットワーク構成を保持している XML で記述されたスクリプトで，中央コントローラが保持している．ネットワーク構成の情報には，ネットワーク構成要素の IP アドレス，使用可能なポート番号，デバイスの通信プロトコルなどが記述されている．詳細は 4.3 で述べる．
- データフロースクリプト (図中の Data Flow script)
  中央コントローラが保持し，管理者によってデバイスの制御とデータ処理，データフローを XML で記述したものである．このスクリプト内で定義されるのは，ネットワーク構成スクリプトで定義されているデバイス制御，データ処理，データフローであり，データの転送に用いられる IP アドレスやポート番号は定義されない．このため，ネットワークトポロジに依らずにデータフローが記述できる．詳細は 4.4 で述べる．
- 設定ファイル (図中の Setting file)
  中央コントローラがネットワーク構成スクリプトとデータフロースクリプトを 元に生成するファイルで，各ネットワーク機器の設定，各ホストの動作が記述されたファイルである．中央コントローラは設定ファイルを生成した後，そのファイル各ネットワーク構成要素に配布をすることで，サービスを展開する．ネットワーク構成スクリプトとデータフロースクリプトから設定ファイルを生成することについての詳細は第 5 章で述べる．

このようなシステム構成のもとで，本研究においては，以下のことを前提条件としている．

1. 中央コントローラはネットワークのトポロジ情報を自動的に収集するものとし，ネットワーク構成スクリプトが自動的に生成されるものとする．
2. ネットワークトポロジはツリー構造として扱い，任意のホスト間のパスは一意に定まるものとする．
3. 中央コントローラからは各ネットワーク機器，ホストへは常にコネクションが張られており，各動作を制御できる．

1 に関しては，Simple Network Management Protocol (SNMP)[21, 22] など既存のネットワークトポロジを収集するための技術を用いれば良い．2 に関しては，Open Shortest Path First (OSPF)[23] でも用いられている，グラフ構造のネットワークをダイクストラのアルゴリズム，またはスパニングツリープロトコルなどを用いることで，ツリー構造として扱うことが可能である．3 については，NETCONF[24, 25] などを用いて外部からネットワーク機器の設定や，ホストのアプリケーションの設定を行えるものと仮定する．

4.2　ネットワーク構成スクリプトとデータフロースクリプトに基づく設定ファイルの生成

中央コントローラは，ネットワーク構成スクリプトと，管理者が定義したデータフロースクリプトを元に，各ホストの動作やネットワーク機器の設定を計算し設定を行う．データフロースクリプトは，デバイ

ス制御，データ処理，データフローについて記述された抽象的なプログラムであり，データの転送先の IP アドレスやポート番号などについては記述されず，ネットワーク構成に依らないものである．実際のネットワーク上にデータフロースクリプトで定義されたアプリケーションを展開するためには，データの転送先の IP アドレスやポート番号，ネットワーク機器の設定などが必要となるため，中央コントローラは，データフロースクリプトで定義されたアプリケーションをネットワーク上へ展開するためのデータの転送先やネットワーク機器の設定をネットワーク構成スクリプトを元に計算し，設定ファイルを生成して各ホスト，ネットワーク機器の設定を行う．ネットワーク構成スクリプトと，データフロースクリプトから各ホストとネットワーク機器の設定ファイルが生成される概念図が図 4.2 であり，この設定ファイルが配布され設定されることで図 4.3 のようにデータフローが展開される．

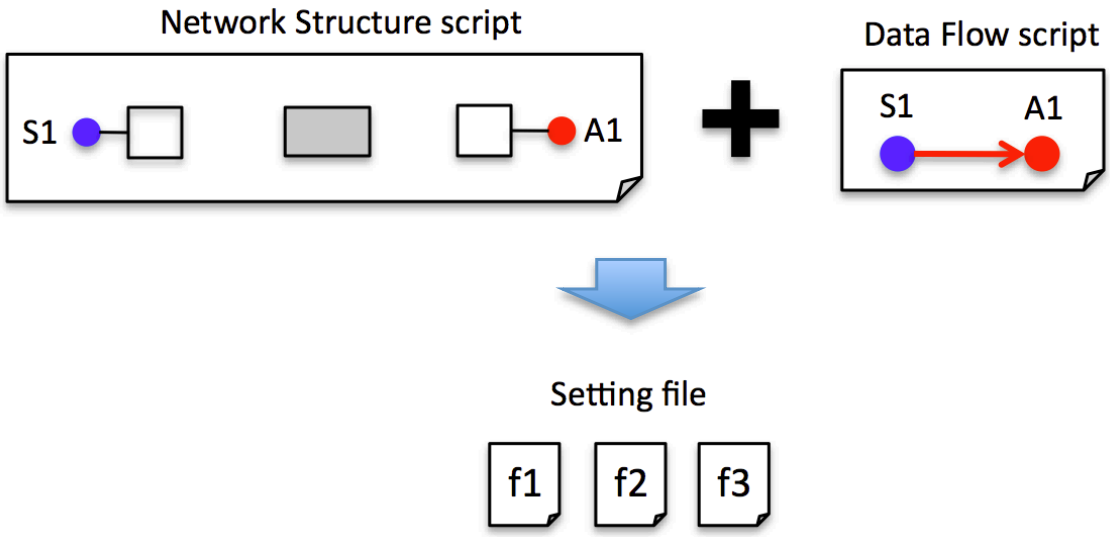

図 4.2　ネットワーク構成スクリプトとデータフロースクリプトから生成される設定ファイル

データフロー展開に伴う，設定ファイルの生成の過程については第 5 章で述べる．

4.3　ネットワーク構成スクリプト

ネットワーク構造ファイルは，システム全体のネットワーク構成を記述した XML で記述されたファイルである．本研究では，4.1 の前提条件で述べた通り，中央コントローラがネットワークのトポロジ情報を SNMP などを用いて何らかの方法で自動的に収集し，トポロジを木構造として計算していることを仮定している．図 4.1 のようなネットワーク構成は，図 4.4 のような木構造に置き換えることができ，中央コントローラを根ノードとして，ネットワーク機器を内部ノード，ホストを葉ノードと見ることができる．

ネットワーク構成スクリプトの構造は，図 4.5 のように resource を根ノードとして，子ノードに Node タグのノードが存在し，この Node タグがネットワーク構成要素 1 つに対応している．

なお，記述方法は CCDM を参考にしているが，CCDM は本研究のようなネットワーク構成を仮定していないため，記述方法を拡張している．Node タグは，name と type の属性を持ち，name はネットワーク構成要素につけられる任意の名前で，トポロジ情報を保持するために用いられるもので，本研究では管理者が定義するものとしている．type はその Node の種類を表し，Host, NAT, FW, ALG のいずれかである．Node タグの子ノードは，その Node の IP アドレス，使用可能なポート番号など Node の詳

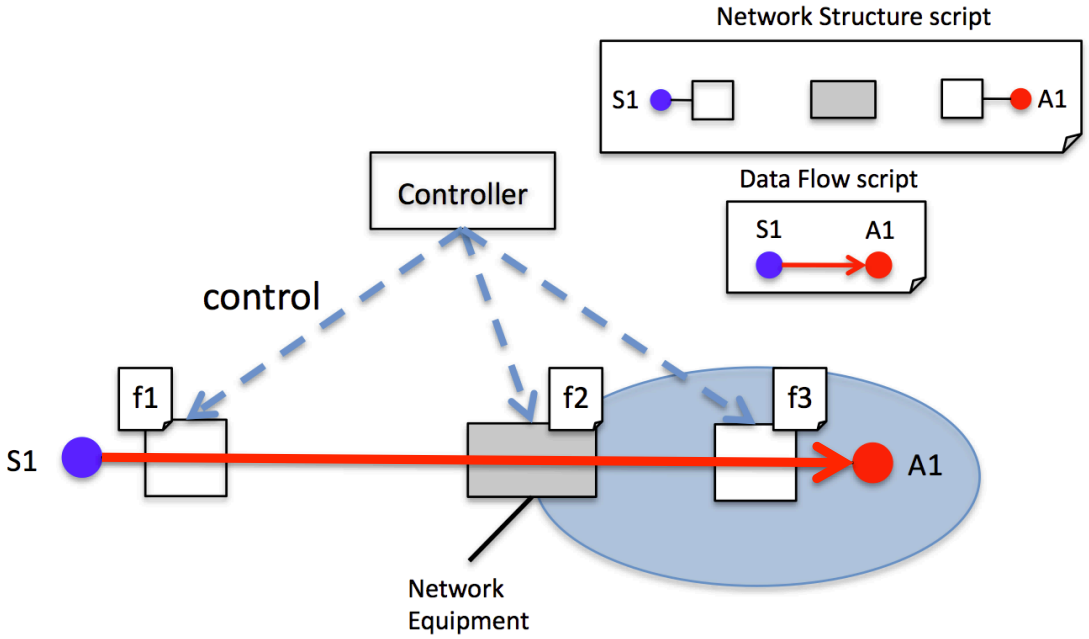

図 4.3　各ネットワーク機器，ホストの設定が行われ，データフローが展開される様子

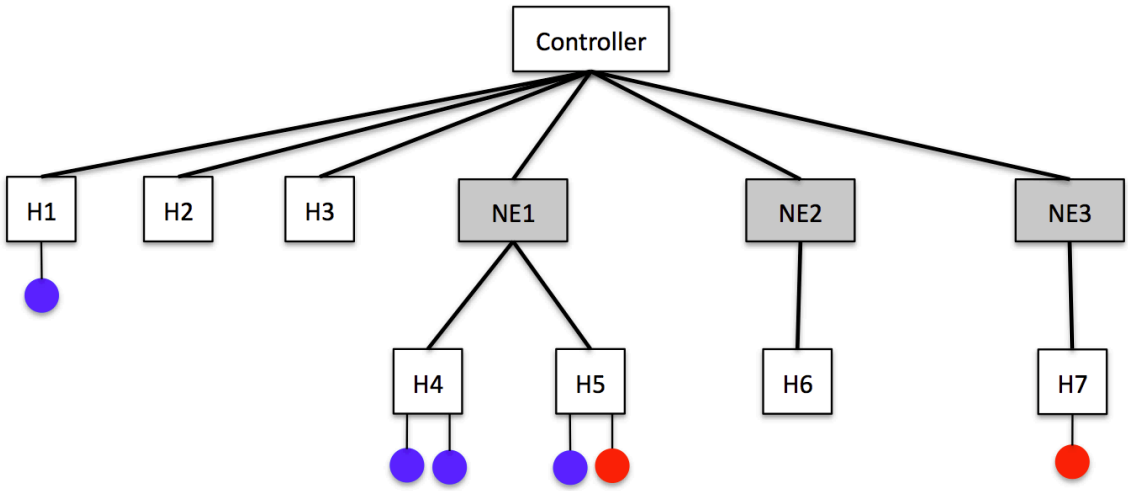

図 4.4　木構造

細情報を示しており type ごとに異なり，詳細はそれぞれの type ごとに述べる．本研究で実装したシステムにおいては，どの type の Node も parent_host タグのノードを持ち，その Node の親ノードにあたる name 属性の名前を記述することで，ネットワークトポロジ情報を保持している．本研究ではネットワーク構成スクリプトは，前述のとおり SNMP などを用いてネットワークトポロジが収集され，自動的に生成されるものと仮定している．

4.3.1　Node の種類ごとのネットワーク構造スクリプトの書き方

4.3 で述べた通り，Node の type によって構成要素で扱う情報が異なる．ホスト，NAT，FW，ALG のネットワーク構成スクリプトの記述について，それぞれ述べる．

```
<resource>
  <Node name="{Node name1}" type="{Node type}">
  ...
  </Node>
  <Node name="{Node name2}" type="{Node type}">
  ...
  </Node>
  ...
</resource>
```

図 4.5　ネットワーク構成スクリプトの概観

```
<resource>
...
  <Node name="{Node name}" type="Host">
    <parent_host>{Parent node name}</parent_host>
    <ip>{IP address}</ip>
    <port>
      <begin>{port}</begin>
      <end>{port}</end>
    </port>
    <devicelist>
      <device name="{device name}" func="INPUT" type="{device type}">
        <!-- device protocol -->
        ...
      </device>
    </devicelist>
  </Node>
  ...
</resource>
```

図 4.6　ネットワーク構成スクリプトにおけるホストの詳細情報に関する記述

4.3.1.1　ホスト

ホストに関して定義すべき情報は，ホストの IP アドレス，使用可能なポート番号，管理するデバイスの情報である．図 4.6 にネットワーク構造スクリプトにおけるホストの情報の記述方法を示す．

図中の Node は type 属性”Host”によりホストの情報であることを示している．Node タグの子要素にはホストの情報が記されており，parent_host タグはこのホストの親ノードの名前，ip タグにはホストの IP アドレス，port タグは begin と end の子要素を持ち，begin タグに示されたポート番号から end タグに示されたポート番号までがこのシステムで用いることができるポート番号であることを示している．devicelist タグにはデバイスのプロトコル情報が記されており，1 つ以上の device タグの要素を持

```
<resource>
...
  <Node name="{Node name}" type="NAT">
    <parent_host>{Parent node name}</parent_host>
    <ethlist>
      <eth name="{eth name}" type="out">
        <ip>{IP address}</ip>
        <port>
          <begin>{port}</begin>
          <end>{port}</end>
        </port>
      </eth>
      <eth name="{eth name}" type="in">
        <ip>{IP address}</ip>
          <port>
             <begin>{port}</begin>
             <end>{port}</end>
          </port>
      </eth>
    </ethlist>
  </Node>
...
</resource>
```

図 4.7　ネットワーク構成スクリプトにおける NAT の詳細情報に関する記述

つ．device には，type 属性で定義されている属性が存在し，データフロースクリプトで属性が定義されているデバイスに対する一括処理の記述で使用される．図中の device protocol という部分はデバイスのプロトコルが記述されている部分であり，デバイスの操作に用いられる．

4.3.1.2　NAT

NAT ルータは複数のネットワークインタフェースを持つ．図 4.7 に NAT ルータのネットワーク構成スクリプトの例を示す．parent_host が表す意味はホストノードと同様であり，他に ethlist タグを持つ．ethlist タグは eth タグを子要素として持ち，eth タグの type は in と out の 2 種類があり，eth タグの in は NAT ルータのプライベートアドレス空間側のインタフェースを示し，out はもう片方のインタフェースの情報を示す．なお，eth タグ内の ip, port タグについてはホストノードと同様の意味を示す．NAT ルータは，ポートフォワードの設定が必要なため，使用可能なポート番号を示している．

4.3.1.3　FW

FW も NAT と同様，複数のネットワークインタフェースを持ち，必要なのはそれぞれの IP アドレスである．図 4.8 にネットワーク構成スクリプトの FW の例を示す．

```
<resource>
...
  <Node name="{Node name}" type="FW">
    <parent_host>{Parent node name}</parent_host>
    <ethlist>
      <eth type="out">
        <ip>{IP address}</ip>
      </eth>
      <eth type="in">
        <ip>{IP address}</ip>
    </ethlist>
  </Node>
...
</resource>
```

図 4.8　ネットワーク構成スクリプトにおける FW の詳細情報に関する記述

4.3.1.4 Application Level Gateway

ALG については NAT と同様の構成となっている．図 4.9 に例を示す．

4.4 データフロースクリプト

データフロースクリプトは管理者がデバイスの制御とデータ処理について XML を用いて定義したものである．なお，データフロースクリプトは，CCDM，[11] で提案されているものを用いているが，デバイスの属性に対する一括処理を実現するための命令セットとして，本研究では新たな命令セット INPUTmultichannel, OUTPUTmultichannel を定義した．

データフロースクリプトはデバイスの制御，データ処理の小さなプログラムの集合で構成されており，そのプログラム間のデータフローが定義されている．スクリプトはネットワーク構成スクリプトで定義されたデバイスと，センサから取得したデータ処理についてのアプリケーションレベルのみ定義されており，このスクリプト内では IP アドレス，ポート番号といったトランスポート層以下の記述はされない．これにより，管理者はデバイスの制御とデータ処理のみを意識すればよく，ネットワークトポロジに依らないデータフローの定義が可能となる．図 4.10 がデータフロースクリプトの概念図である．

データフロースクリプトは，最終的に小さなプログラムに分割されて適当なホストへ割り当てられるが，スクリプト内のプログラム間のデータフローの定義は，中央コントローラの計算により転送先の IP アドレス，ポート番号へ変換されて各ホストへ割り当てられる (図 4.11)．

データフロースクリプトの命令セットは，表 4.1 のようになっており，[11] で提案されている基本的な命令セットと，デバイスの通信プロトコルの隠蔽やスクリプトを分散実行するために CCDM で提案された命令セットが存在する．

基本的な命令セットは，表中の Data Type から Socket で，データの型の定義，四則演算，ビット演算，論理演算，比較，条件分岐などの命令が定義されている．CCDM で提案された命令セットは Distributed

```
<resource>
...
  <Node name="{Node name}" type="ALG">
    <parent_host>{Parent node name}</parent_host>
    <ethlist>
      <eth type="out">
        <ip>{IP address}</ip>
        <port>
          <begin>{port}</begin>
          <end>{port}</end>
        </port>
      </eth>
      <eth type="in">
        <ip>{IP address}</ip>
        <port>
          <begin>{port}</begin>
          <end>{port}</end>
        </port>
    </ethlist>
  </Node>
...
</resource>
```

図 4.9　ネットワーク構成スクリプトにおける ALG の詳細情報に関する記述

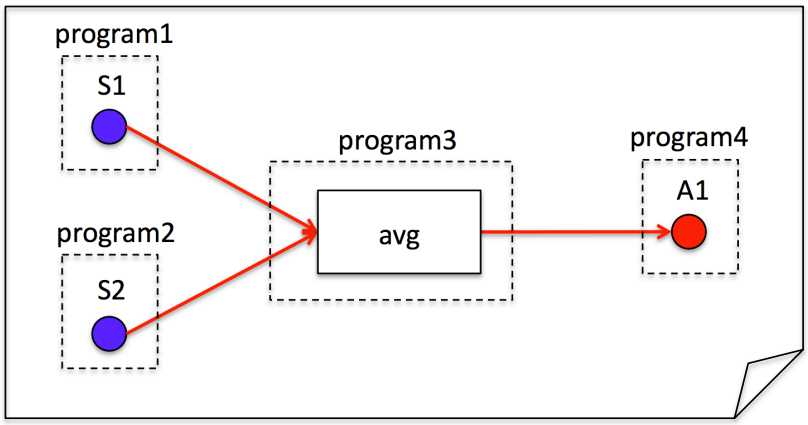

図 4.10　データフロースクリプトの概念図

Execution, Device Communication Abstraction であり，本研究でも用いる．本研究では，属性が定義されたデバイスに対して一括処理を行うための Device Attribution Communication の Category に属する INPUTmultichannel, OUTPUTmultichannel の命令を追加した．以下，CCDM で提案された命令セット (4.4.1, 4.4.2, 4.4.3) と本研究で提案する命令セット (4.4.4) について述べる．

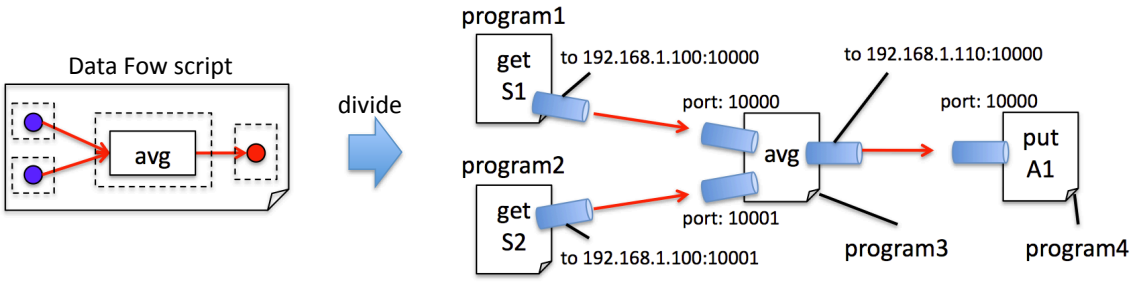

図 4.11　データフロースクリプトの分割

表 4.1　データフロースクリプトの命令セット

Category	Command
Data Type	byte int double text null true false
Operation	plus minus multiply divide mod
Bit Operation	bitand bitor bitxor bitlshift bitrshift biturshift
Logical Operation	and or not xor
Test	eq neq lt gt lteq gteq
Type Cast	castbyte cast int castdouble casttest
Branch	if
Loop	while
Procedure	progn
Variable	setq getq
Array	byteArray byteArrayAppend byteArrayGet byteArrayLength
Socket	dgSocket dgServerSocket dgSend dgRecv dgClose
Distributed Execution	chunk post catch
Device Communication Abstraction	INPUT OUTPUT
Device Attribution Communication	INPUTmultichannel OUTPUTmulticahnnel

4.4.1　chunk

chunk はデータフロースクリプトを構成するデバイスの制御かデータ処理について記述された分割不可能なプログラムの単位であり，データフロースクリプトはこの chunk の集合で構成され，図 4.10 内の破線で囲まれた 1 つ 1 つの program に当たる．以降，この分割不可分のプログラムを chunk と記述する．各ホストの動作を設定するための設定ファイルを生成する際には，図 4.11 のようにこの chunk の単位で分割される．chunk 同士のデータフローの定義は，4.4.2 の post, catch 命令を用いる．

4.4.2　post, catch

4.4.1 で述べた通り，chunk 同士で値を送受信をするときは post, catch 命令を用い，chunk から他の chunk にデータを送信する場合には post，受信する場合には catch 命令を用いる．なお，catch 命令で受信されるデータは到着した順番に処理され，データの処理順序は制御されない．post と catch の概念を表したものが図 4.12 であり，この図は chunk1 から post により値を出力し，chunk2 は catch により値を受信している

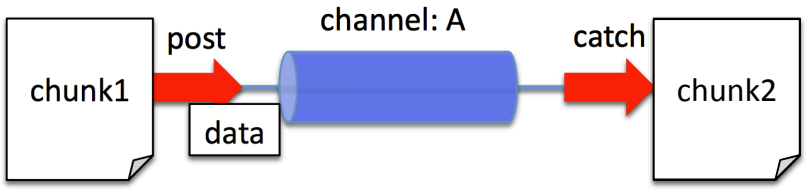

図 4.12　post と catch の概念

post, catch 命令には，共に name 属性と channel 属性があり，post 命令で送信されたデータは catch 命令で受信するという対の関係になっている．post 命令と catch 命令の対の関係はそれぞれの channel 属性で表現され，同じ channel 名を持つ post 命令，catch 命令でデータのやり取りが行われる．上で示した図 4.12 は基本的な post と catch 命令の概念となっており，post と catch の channel 属性が”A” となっていることを示している．この post, catch の channel 属性の指定により，一対多，多対一，多対多の chunk 間のデータ転送が可能である．

4.4.3　INPUT, OUTPUT

INPUT/OUTPUT 命令は，デバイスを制御する際に用いられる命令セットであり，デバイスの通信プロトコルの隠蔽し，デバイスの操作に対する共通のインタフェースを提供するものである．INPUT 命令では from 属性で図 4.6 にある，device タグの name 属性で定義されたデバイス名を設定することにより，設定されたデバイス名からデータを取得する．OUTPUT 命令では，to 属性でデバイス名を指定し，OUTPUT タグの子要素である ARG 要素で指定された変数を設定することで，デバイスへ値を出力する．デバイスからのデータの取得，データの出力は，ネットワーク構成スクリプトの device タグ内に定義されたデバイスのプロトコルに従い行われる．この命令セットの目的は，デバイスのプロトコルの隠蔽であり，CCDM で定義された命令セットなため，詳細な説明は省く．

図 4.13 は chunk, post, catch, INPUT, OUTPUT を用いたプログラム例である．このプログラムは，センサ S1 からデータを取得し，アクチュエータ A1 へデータを出力するというプログラムを示している．chunk2 つから構成され，前半の chunk はセンサ S1 からデータを取得し，その値を post を用いて channel 属性 A として出力し，後半の chunk は catch を用いて channel 属性 A としてデータを受信し，アクチュエータ A1 へ値を出力している．

また，図 4.10 のような動作を記述するデータフロースクリプトは付録 A へ添付する．

```
<progn>

  <chunk>
    <progn>
      <INPUT from="S1"/>  <!-- get value from S1 -->
      <post channel="A" name="INPUT"/>  <!-- send S1's value to channel A -->
    </progn>
  </chunk>

  <chunk>
    <progn>
      <catch channel="A" name="receive"/>  <!-- receive data -->
      <OUTPUT to="A1">  <!-- send received data to A1 -->
        <ARG>
          <getq name="receive"/>
        </ARG>
      </OUTPUT>
    </progn>
  </chunk>

</progn>
```

図 4.13　データフロースクリプトのプログラム例

4.4.4　INPUTmultichannel, OUPUTmultichannel

NAT, FW, ALG などのネットワーク機器が存在するネットワークにおいても，本研究の提案するアーキテクチャによりデバイス制御がアプリケーションレベルで可能となることを示すために，本研究で新たに定義した命令セットである．2.3 で述べたように，同じ属性が定義されたデバイスに対して同様の処理を行いたいという場合，デバイスの個数は予め可変であることを想定しなければならない．この命令セットは，デバイスの個数に依らずにデータ処理を記述することを可能にするためのインタフェースであり，デバイスの状態に応じてデバイスとデータ処理間のデータフローを自動的に定義し，データフロースクリプトを自動的に生成する．ネットワーク構成スクリプトのデバイスの情報が記述された device タグには，type 属性があり，この属性を INPUTmultichannel, OUTPUTmultichannel の type 属性で定義することで，デバイスをまとめて扱う．

INPUTmultichannel は，書式は図 4.14 のようになっており，このタグの type 属性の値と同じ type 属性が定義されたデバイスから interval 属性で指定された間隔で値を取得し，繰り返し処理を行う．INPUTmultichannel タグ内では，値を受信するごとに行う処理を記述を行うことができ，プログラムが可能なため，ユーザは type に該当するセンサから取得できた値の最大値や平均値を求める，ということが可能となる．

```
<chunk>
  ...
  <INPUTmultichannel type="{device type}" interval="{interval time}">
    <progn>
      {data process}
    </progn>
  </INPUTmultichannel>
</chunk>
```

図 4.14　INPUTmultichannel の書式

```
<chunk>
  ...
  <OUTPUTmultichannel type="{device type}" name="{variable name}"/>
</chunk>
```

図 4.15　OUTPUTmultichannel の書式

OUTPUTmultichannel の書式は図 4.15 のようになっており，このタグの type 属性で指定された type 属性を持つ全てのアクチュエータへ値を送信するための命令である．OUTPUTmultichannel タグの type 属性で指定された属性 (この場合は display) を持つ全てのアクチュエータへ name 属性で指定された値が送信される．

INPUTmultichannel, OUTPUTmultichannel 命令の動作については，第 5 章で述べる．

第 5 章

ネットワーク上へのデータフロー展開

第 4 章で述べたアーキテクチャで，中央コントローラは管理者が定義したデータフロースクリプトの通りにデバイス同士が協調動作するよう，各ホストとネットワーク機器の設定を行う．

この過程は，以下の手順にて行われる．

1. ネットワーク構成スクリプトの解析
2. データフロースクリプトの解析
3. 各ホストへの分割されたプログラムの割り当ての決定
4. データフローとネットワーク機器の設定の計算

以下，図 5.1 のようなネットワーク構成において，type 属性が temperature のセンサから取得した値の平均値を type 属性が display のアクチュエータへ出力するデータフロープログラムの展開を例に挙げながら各手順について詳細を述べる．この時，データフロースクリプトは INPUTmultichannel, OUTPUTmultichannel 命令を用いており，INPUTmultichannel 命令では type が temperature に指定されて，OUTPUTmultichannel では type が display に指定されている状態である．

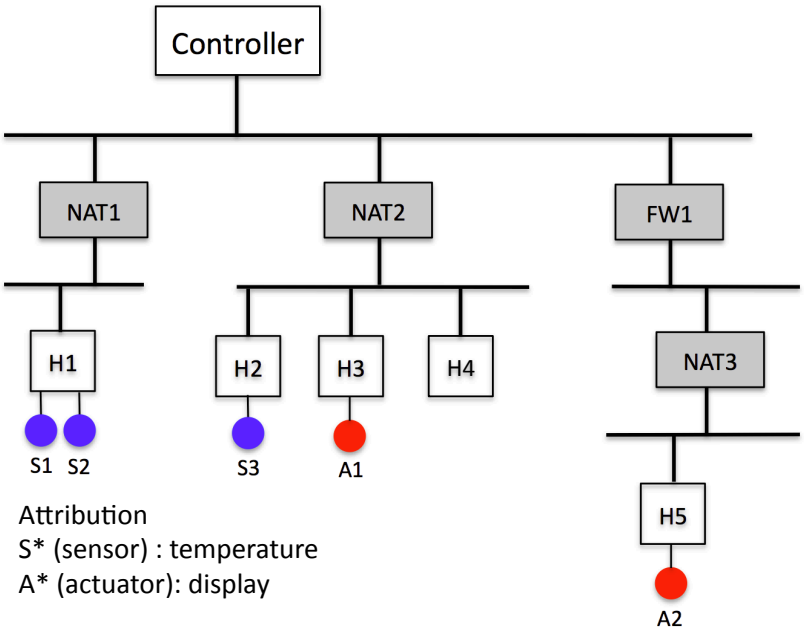

図 5.1　ネットワーク構成例

なお，図 5.1 中の S*は属性 temperature の温度センサ，A*は属性 display のディスプレイとする．以下，図中の Controller は中央コントローラ，NAT*は NAT， FW*は FW，ALG*は ALG，H*はホストを示しているものとする．

5.1　ネットワーク構成スクリプトの解析

まず中央コントローラはネットワーク構成スクリプトの解析をし，ネットワークトポロジとネットワーク構成要素の情報を取得する．ネットワークトポロジは，4.1 の前提条件で述べた通り，予めネットワーク構成スクリプトにトポロジ情報が含まれており，そのトポロジはツリー構造であることを仮定しており，任意のホスト間のパスは一意に定まる．

ネットワークの構成スクリプトの解析で収集される情報は，各ノードのタイプ (Host, NAT, FW, ALG など) とそのノードの詳細情報とネットワークトポロジ情報であり，ノードの詳細情報は，IP アドレス，使用可能なポート番号，また，Host に関しては保持しているデバイスとその属性とプロトコルの情報が収集される．デバイスの属性情報は，本研究で提案した INPUTmultichannel, OUTPUTmultichannel 命令に用いられるものであり，プロトコル情報は，INPUT, OUTPUT という抽象化された命令をそれぞれのデバイスに対応するプロトコルに置換する際に用いられるもので，これは CCDM の機能を用いている．

5.2　データフロースクリプトの解析

データフロースクリプトの解析では，デバイスの属性情報に基く INPUTmultichannel, OUTPUTmultichannel 命令の置換と，デバイス制御の chunk 内で用いられるデバイス情報の取得が行われる．

5.2.1　INPUTmultichannel, OUTPUTmultichannel 命令の展開

本研究では，定義された属性情報に基づいたデバイスに対する一括処理を定義できる INPUTmultichannel, OUTPUTmultichannel という命令を定義した．この命令は，デバイスの属性情報に基づいてそれぞれのデバイスの制御とデータ処理のデータフローが自動的に展開されるため，デバイスの属性情報に対するデータフローの定義を行えば，管理者はデバイス情報の追加，削除を行うのみで，データフロースクリプトを変更することなく適切にデータフローが展開される (図 5.2)．中央コントローラは，ネットワーク構成スクリプトの解析において取得したデバイスの属性情報を元に INPUTmultichannel と OUTPUTmultichannel について以下の手順でデータフローを設定する．

5.2.1.1　INPUTmultichannel

INPUTmultichannel の書式は 4.4.4 で述べた通り，図 4.14 のようになっている．

以下，INPUTmultichannel 命令が展開される手順である．

1. INPUTmultichannel タグの type 属性と interval 属性の値を取得する．
2. INPUTmultichannel タグが展開される際に用いる post，catch で用いる channel 名 (random_channel_name11) を他の channel 名と重複しないように決定する．
3. 1 で取得した type 属性のデバイス一覧を，ネットワーク構成スクリプトの解析で得た情報から取得する．

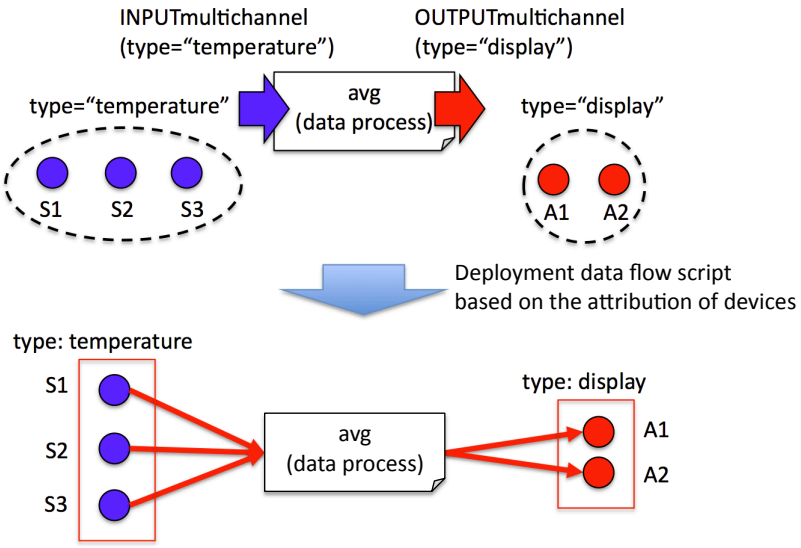

図 5.2　INPUTmultichannel, OUTPUTmultichannel の展開

4. 該当する個々のデバイスからデータを取得し，データ処理の chunk (図 5.2 中の avg という chunk) へデータを送信する chunk を自動生成する．この時，データ送信の際に使用される post の channel 属性は 2 で決定した random_channel_name1 である．データを送信する間隔は，INPUTmultichannel の interval 属性で設定された値であり，この間隔でデバイスから取得されたデータはデータ処理の chunk に送信される．この，デバイス制御を行う自動生成されるプログラムは図 5.3 のようになっており，これは interval の間隔で device name から取得したデータを random_channel_name1 へ出力するというプログラムである．
5. データ処理の chunk 内に記述された INPUTmultichannel タグの内部は，データを受信した際に行う繰り返し処理となり，値を受信して処理を行うことを該当する属性が定義されたデバイスの個数分繰り返すという chunk に展開し直す．図 4.14 は図 5.4 のように展開される．図 5.4 中の catch タグと繰り返しの処理の部分は，該当するデバイスの個数分繰り返される．これにより，デバイスから取得されたデータを繰り返し処理を行うことが可能である．

この例では INPUTmultichannel により，S1 から S3 のそれぞれについて，デバイスの値を取得して INPUTmultichannel 命令 chunk にデータを送信するという chunk が自動生成され，INPUTmultichannel 命令はデータを受信して data process で定義された処理を行う，ということを 3 回繰り返すプログラムへ展開される．

5.2.1.2　OUTPUTmultichannel

OUTPUTmultichannel 命令は，属性で指定されたデバイス全てに対して同じ値を出力する際に使用する．OUTPUTmultichannel は 4.4.4 で述べた通り，図 4.15 のような書式になっており，この命令は以下のように展開される．

1. OUTPUTmultichannel タグの type 属性の値を取得する．
2. OUTPUTmultichannel タグが展開される際に用いる post，catch で用いる channel 名 (random_channel_name2) を他の channel 名と重複しないようにランダムに決定する．

```
<chunk>
  <while>
      <eq>
        <int>1</int>
        <int>1</int>
      </eq>
      <progn>
        <sleep><int>{interval}</int></sleep>
        <INPUT from="{device name}"/>
        <post channel="{random_channel_name1}" name="INPUT"/>
      </progn>
  </while>
</chunk>
```

図 5.3　INPUTmultichannel で自動生成されるデバイス制御の chunk プログラム

```
<chunk>
    ...
  <catch channel="{random_channel_name1}" name="new_val"/>
    {data process}
  <catch channel="{random_channel_name1}" name="new_val"/>
    {data process}
    ...
</chunk>
```

図 5.4　INPUTmultichannel 命令が展開されるプログラム

3. 1 で取得した type 属性のデバイス一覧を，ネットワーク構成スクリプトの解析で得た情報から取得する．
4. type 属性に指定されたものに該当する個々のデバイスへデータを送信する chunk を自動生成する．自動生成されるプログラムは，図 5.5 のようになっており，これは random_channle_name2 の channel 属性を持つ post から出力されたデータを受信し，そのデータを device name のデバイスへ出力する chunk である．
5. OUTPUTmultichannel タグは，channel 属性が 2 で決定した random_channel_name2 で，name 属性が OUTPUTmultichannel の name 属性で設定された post タグに置換される．これにより，4 で自動的に生成されたデバイス制御の chunk 全てに，name 属性で指定された変数の値が送信される．

以上のようにして，INPUTmultichannel, OUTPUTmultichannel 命令はデバイスの個数に応じてデータフローが動的に展開される．

例においては，OUTPUTmultichannel 命令により，A1, A2 のデバイス制御を記述した chunk が自動生成され，OUTPUTmultichannel 命令はデータ処理による演算結果をそれらのデバイス制御の chunk

```
<chunk>
  <while>
    <eq>
      <int>1</int>
      <int>1</int>
    </eq>
    <progn>
      <catch channel="{channel name}" name="output_message" />
      <OUTPUT to="{device name}">
        <ARG>
          <getq name="output_message"/>
        </ARG>
      </OUTPUT>
    </progn>
  </while>
</chunk>
```

図 5.5　アクチュエータへのデータの送信に自動生成される chunk プログラム

へ送信する命令へと展開される．

以上のように，INPUTmultichannel, OUTPUTmultichannel は，その時点でシステム以下に存在するデバイスに応じて 5.2 のようにデータフロースクリプトが自動的に展開される．

5.2.2　デバイス制御の chunk のデバイス名抽出

デバイス制御の chunk はそのデバイスを管理しているホストへ割り当てる必要があるため，個々の chunk が制御するデバイスの情報を取得しておく．この情報は，5.3 で述べるように chunk の各ホストへの割り当ての決定の際に用いられる．

5.3　ホストのタスク割り当ての決定

中央コントローラは，システム管理者が記述したデータフロースクリプトをネットワーク構成スクリプトの解析が終わると，次にその情報に基づいてデータフロースクリプトを構成するそれぞれの chunk が割り当てられるホストを決定する．chunk は，割り当てるホストが一意に決定するものとしないものの 2 種類に分かれる．

5.3.1　割り当てるホストが一意に決定するもの

これは，デバイスから値を取得するもの，出力するものなどデバイス制御に関する chunk である．デバイスは 1 つのホストに管理されているために，デバイスの操作の chunk の割当先のホストは一意に定まる．図 5.2 の中でこのタスクに当てはまるものは，S1 から S3 のセンサ，A1 と A2 のアクチュエータのデバイス制御を行っている 5 つの chunk である．S1, S2 のデバイスを操作するそれぞれのプログラム

は，S1, S2 を管理している H1 へ割り当てられ，その他のタスクも同様にデバイスを管理しているホストへ割り当てられる．

5.3.2　割り当てるホストが一意に決定しないもの

デバイスの制御以外の，データ処理の chunk は，任意のホストで動作可能である．chunk が割り当てられたホスト間の物理的なホップ数を抑える，近いネットワークでデータフロー展開するなどにより無駄な通信経路を削減することなどが考えられるが，今回はランダムに割り当てられることとする．図 5.2 の INPUTmultichannel, OUTPUTmultichannel が展開された後の chunk の中でこれに該当するものは，平均値の計算 (avg) の chunk である．これらのプログラムは，システム下のホスト (この例では H1〜H5) の任意のホストにランダムで割り当てられる．今，この例において平均値を計算する chunk が H4 に割り当てられたとする．

5.4　データフローの計算

データフロースクリプト内の chunk 間のデータフローは post, catch 命令で定義されておりデータの転送先の IP アドレスやポート番号は定義されていない．そこで，chunk のホストへの割り当て先が決定した後に，各 chunk 間のデータフローを計算し，各データの転送先の IP アドレスとポート番号を計算して決定することで，post, catch 命令で記述された抽象的なデータフローを IP アドレス，ポート番号へ置き換える．

4.1 の前提条件で仮定している通り，中央コントローラはネットワークトポロジをツリー構造として扱うため，chunk の各ホストへの割り当てが決定すると chunk が割り当てられたホスト間のデータパスは一意に決定する．そして chunk 間のデータパス上に NAT, FW, ALG などのルータが存在する場合は，データが通過させるための設定を入れる必要がある．1 つの chunk 間のデータフローの計算手順は以下のようになっている．

1. chunk 間の物理的なパスを計算する．
2. パス中で設定が必要なネットワーク機器を選択する．
3. データを通過されるためのネットワーク機器の設定を決定する．
4. データフロースクリプトのデータのフローを表した抽象的な表現である post 命令，catch 命令を計算結果に基づいて IP アドレス，ポート番号へ置き換える．

以上の計算手順を，全ての chunk 間のデータフローに対して行う．以下，それぞれの詳細を avg から A2 へのデータフローを例に挙げて述べる．

5.4.1　chunk 間の物理的なパスの計算

chunk が割り当てられたホスト間の物理的なパスは，中央コントローラがネットワークトポロジをツリー構造で保持している仮定から自明である．avg の chunk が割り当てられたホスト H4 から A2 の chunk が割り当てられるホスト H5 までのパスは H4, NAT2, FW, NAT3, H5 である (図 5.6).

図 5.6　H4 から H5 の物理的なパス

5.4.2　パス中で設定が必要なネットワーク機器の選択

次に設定が必要なネットワーク機器を選択する．これは，FW，ALG に加え，データフローがグローバルアドレス空間からプライベートアドレス空間へ通過する NAT ルータである．今挙げている例では，H4, FW, NAT3, H5 となる (図 5.7).

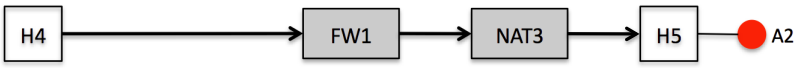

図 5.7　H4 から H5 の設定が必要なネットワーク機器のパス

NAT2 に関しては，データフローがプライベートアドレス空間からグローバルアドレス空間へと通過しているため，ネットワーク機器の設定は不要である．

5.4.3　データを通過させるためのネットワーク機器の設定の決定

設定が必要なネットワーク機器を選択した後，それぞれの機器の実際の設定を 1 つずつデータフローの方向に向かって計算する．ここで，説明のために以下の用語を導入する．設定を計算する対象となるネットワーク機器を NE (network equipment), データフロー中の設定の計算の際に考慮する必要がある前の要素を pre node, 後の要素を post node とし，pre node の IP アドレス，ポート番号を srcIP, srcPort, NE の pre node 側の IP アドレス，ポート番号を rcvIP, rcvPort, post node 側の IP アドレス，ポート番号を sndIP, sndPort, post node の IP アドレス，ポート番号を dstIP, dstPort とする (図 5.8).

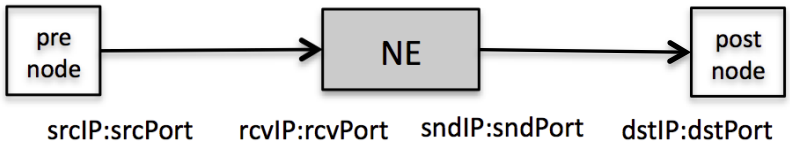

図 5.8　pre node, NE, post node

以下，それぞれのネットワーク機器の設定の計算方法を述べる．最初に NAT, ALG の設定を計算した後，FW の設定を計算するという手順になる．

5.4.3.1　NAT, ALG の場合

NAT に関しては，プライベートアドレス空間からグローバルアドレス空間へデータを送信する場合には特別な設定は必要ないため，グローバルアドレス空間からプライベートアドレス空間へデータを通す場合のみを考えれば良い．データフローがグローバルアドレス空間からプライベートアドレス空間の方向に向いている NAT を NAT(out → in) と表し，逆にデータフローがプライベートアドス空間からグローバ

ルアドレス空間へ向いている NAT は NAT(in → out) と表す．NAT, ALG の場合，データの送信元とネットワーク機器が受信したデータの転送先を考えればいいため，pre node = {Host, NAT(out → in), ALG}, post node = {Host, NAT(out → in), ALG} となる．

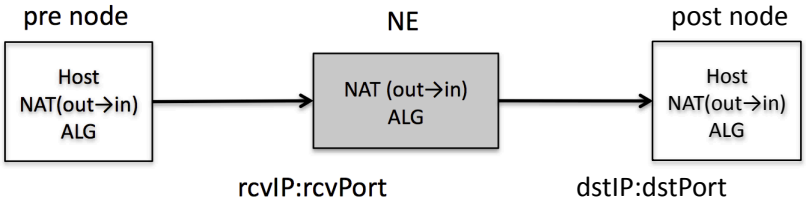

図 5.9　NAT, ALG

設定の計算は以下の手順ですべての NAT(out → in) と ALG に対して行われる．

1. NE の pre node 側の使用可能なポート番号 rcvPort を取得する
2. post node の使用可能なポート番号 sndPort を取得する
3. pre node の送信先を rcvIP, rcvPort に決定する
4. NE は，rcvPort で受信したデータを (dstIP, dstPort) へ転送するよう設定を行う．

これらの情報が決定した後，NAT に関しては図 5.10 の <> で囲まれた変数に適切な値を入力し，NAT の設定ファイルを作成する．なお，変数名は図 5.8 に対応しており，また，NAT の設定に必要な EthOut は rcvIP の IP アドレスを持つネットワークインタフェース名を，EthIn は sndIP の IP アドレスを持つネットワークインタフェース名を示している．

ALG に関しては，rcvPort で受信した値を (dstIP, dstPort) へ送信するプロキシを立てさせる設定ファイルを作成する．

例においては，NAT3 が設定の計算の対象となり，NE = NAT3, pre node = H4, post node = H5 となる．

5.4.3.2　FW の場合

NAT, ALG についての設定が決定した後，FW の設定の計算を行う．なお，本研究において FW に関しては，今回は送信元の IP アドレス，送信先の IP アドレス，ポート番号に対して制限を設ける．送信元の IP アドレスが設定されるのは Host, NAT(in → out), ALG なので，これらを pre node とし，送信先として該当するのが Host, NAT(out → in), ALG となるのでこれらを post node とする (図 5.11).

そして，図 5.11 のような，srcIP が送信元で，(dstIP, dstPort) が宛先のパケットのみを通す設定ファイルを作成する．

```
iptables -t nat -A PREROUTING -p udp -i <EthOut> --dport <rcvPort>\
 -j DNAT --to-destination <dstIP>:<dstPort>
iptables -A FORWARD -i <EthOut> -o <EthIn> -p udp --dport <dstPort> -j ACCEPT
iptables -t nat -A POSTROUTING -d <dstIP> -j SNAT --to <srcIP>
```

図 5.10　NAT に入れる設定のテンプレート

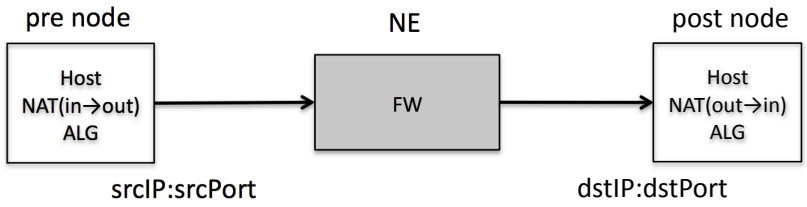

図 5.11　FW

```
iptables -A FORWARD -s <srcIP> -p udp -d <dstIP> --dport <dstPort> -j ACCEPT
```

図 5.12　FW に入れる設定のテンプレート

```
<setq name="socket">
  <dgSocket>
    <text>{IP address}</text>
    <int>{port}</int>
  </dgSocket>
</setq>
<dgSend>
  <getq name="socket"/>
  <getq name="INPUT"/>
</dgSend>
<dgClose>
  <getq name="socket"/>
</dgClose>
```

図 5.13　post コマンドが置き換えられるプログラム

以上のように NAT，FW，ALG の設定が決定すると，中央コントローラは各ネットワーク機器の設定ファイルを生成する．

5.4.4　データフロースクリプトの分割，置換

5.4.3 までで全てのデータフローの送信元，送信先の IP アドレス，ポート番号が決定する．chunk 内の post 命令に関しては，送信先の IP アドレスとポート番号が指定されたプログラムへ (図 5.13)，catch 命令に関してはデータの受信を行う IP アドレスとポート番号が指定されたプログラムへ (図 5.14) 置き換えられる．同時に，データフロースクリプトは chunk ごとに分割し，各ホストの設定ファイルを生成する．

なお，これらの post，catch 命令の置き換えに関しては CCDM の機能を用いている．

```
<setq name="server_socket">
  <dgServerSocket>
    <text>{IP address}</text>
    <int>{port}</int>
  </dgServerSocket>
</setq>
<setq name="response">
  <dgRecv>
    <getq name="server_socket"/>
  </dgRecv>
</setq>
<dgClose>
  <getq name="server_socket"/>
</dgClose>
```

図 5.14　catch コマンドが置き換えられるプログラム

5.5　データフロー展開

各ホスト，ネットワーク機器の設定ファイルが生成されると，中央コントローラは設定ファイルを該当する機器へ配布し，実行させることにより，データ処理を含んだセンサからアクチュエータへのデータフローが展開される．例で挙げているネットワーク構成において展開されるデータフローの様子は図 5.15 のようになっている．図中の get はデバイスからデータを取得し，次のホストへ転送することを，put は受信したデータをアクチュエータへ転送することを，avg は平均を計算している chunk のプログラムが起動していることを表し，S1 から S3 のセンサから取得した値の平均値 (avg) を A1, A2 のアクチュエータへ出力していることを示している．

5.6　ネットワーク構成の変更に伴うデータフローの再展開

システムを構成しているネットワークの構成が変更する場合にも，変更後のネットワーク構成でシステムを継続的に動作させる必要がある．ここで述べるネットワーク構成の変更とは，具板的にデバイス，ネットワーク機器，ホストが新しく追加されたり，取り除かれることを示し，特にデバイスに関してはあるネットワークセグメントから他のネットワークセグメントへ物理的に移動することも含む．デバイスに関して変更が生じるとデータフローの始点であるセンサ，終点であるアクチュエータが変化することになるためデータフローも変更する必要があり，ネットワーク機器とホストに関して変更が生じても，新しいネットワーク構成にデータフローを再度展開する必要があるため，これらの機器に再設定が必要となる．

上記で述べたようなネットワーク構成の変更は頻繁に生じると考えられる．デバイスは，デバイスを新たに購入して追加する場合や，故障により取り除く場合に構成が変わるため，特に BAS のようにデバイスが大量に存在する環境下では，このような変更は頻繁に発生すると考えられる．同様に，ネットワーク

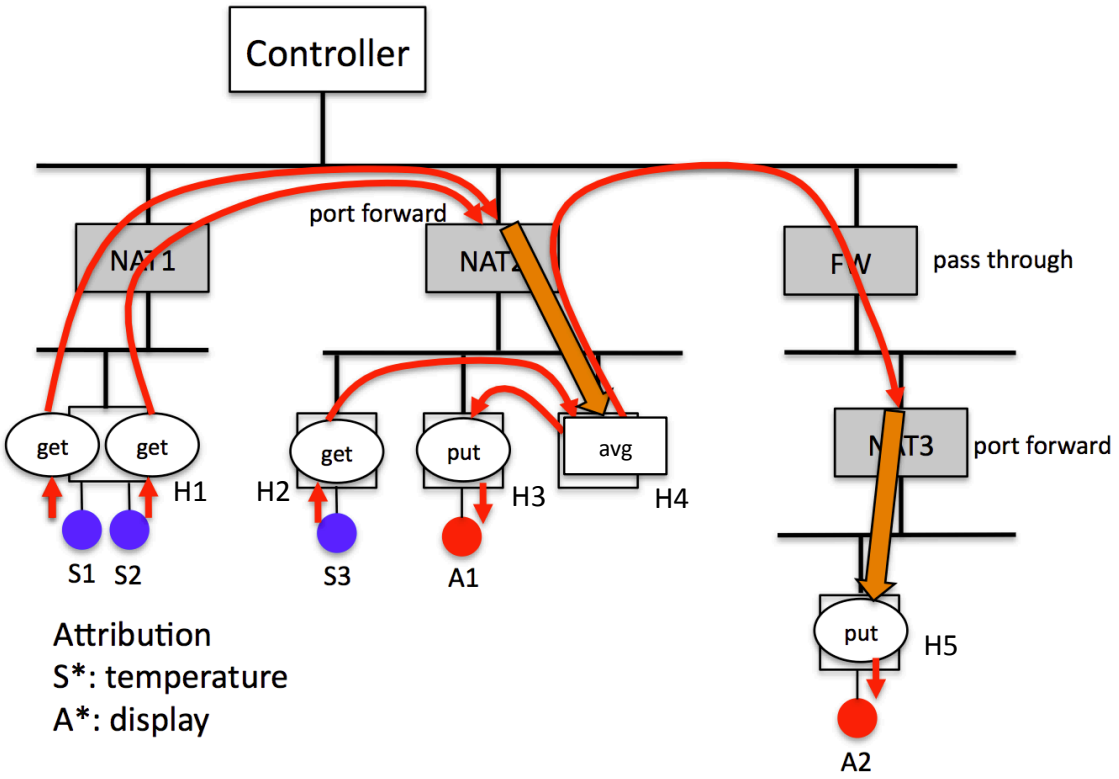

図 5.15　図 5.2 のデータフロースクリプトが図 5.1 のネットワーク構成へ展開されたときのデータフロー

機器やホストも故障により新しい機器へ交換することが考えられる．長期に渡り継続的に運用を行うためには，こうしたネットワーク構成要素の変更に対しても柔軟にデータフローを展開する必要がある．

ここで，デバイスの移動に伴うデータフローの再展開について，図 5.15 において，アクチュエータ A2 が H5 から H4 へ移動した場合を例に挙げて述べる．A2 が H5 から H4 へ移動した後に，H5 は A2 を制御していないこと，H4 が新たに A1 を制御するようになった情報を中央コントローラへアップロードする．なお，今回の実装ではホストから明示的に中央コントローラへ情報をアップロードしている．中央コントローラはネットワーク構成要素が変更したという情報を受信すると，現在ホスト上で実行しているアプリケーションの動作を止めネットワーク機器の設定を初期化し，図 5.16 のようにデータフローの再展開を行う．

なお，データフロースクリプトはネットワークトポロジの変更の影響は受けず，デバイスが移動してもデータフロースクリプトに変更の必要性は生じない．

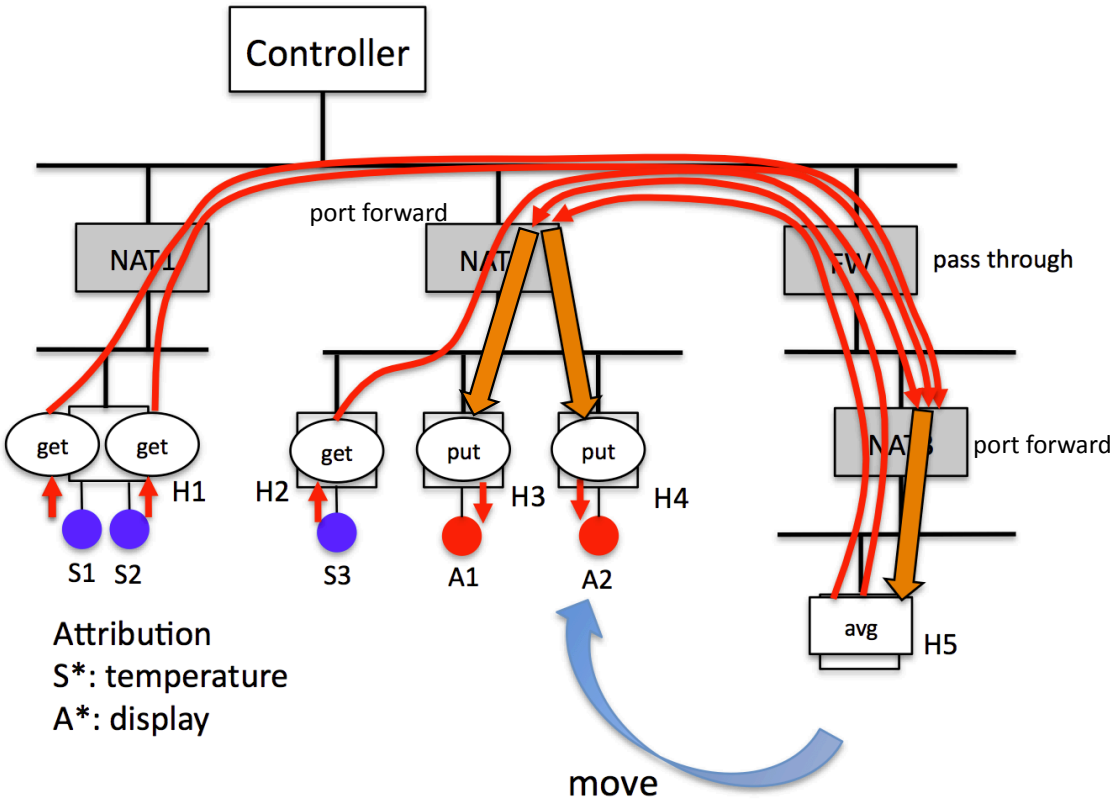

図 5.16　デバイスの移動に伴うデータフローの再展開

第 6 章

動作検証

第 4 章で提案したアーキテクチャの動作検証を行った．実験内容は，個々のデバイスのデータフロー展開を行うもの (6.2) と，デバイスに設定された属性を用いた一括処理を行うもの (6.3) がある．6.1 で動作検証に使用した実験器具を述べた後，動作検証に行った実験についてそれぞれ述べる．

6.1 使用した実験機器

実験は，デスクトップ PC5 台と laptop PC3 台を用いた実機で行ったものと，VirtualBox[26] による仮想マシンを用いて行ったものがある．中央コントローラ，ネットワーク機器，ホスト全てに Ubuntu 10.04 Server Edition をインストールして実験を行った．

実機を用いた実験では Arduino[27] に温度センサ，トグルスイッチ，LED を搭載したもの，またはディスプレイを搭載した実機デバイスをセンサ・アクチュエータデバイスとして用いており，仮想マシンを用いて行った実験では，ランダムに値を出力する仮想温度センサと受信した値を表示する仮想ディスプレイの仮想デバイスを用いている．デバイスの仕様は以下のようになっている．

- 実機デバイス
  温度センサ，トグルスイッチ，LED を搭載したデバイスに関しては，ホストからデバイスに UDP による 1 オクテットの命令を送信すると，温度センサ，トグルスイッチの値の返答，または LED の点灯，消灯を行う．[0x64] の UDP による命令を送信すると温度センサの値を返し，[0x67] の場合はトグルスイッチの値を返す．また，[0x75] の命令を送信すると LED を点灯，[0x6E] の UDP を送信すると消灯する仕様に Arduino にプログラムを組み込んでいる．
  ディスプレイを搭載したデバイスは，受信した UDP の値の ASCII コードに対応する ASCII 文字を表示する．
- 仮想デバイス
  仮想デバイスは，仮想温度センサ，仮想ディスプレイを実装した．仮想温度センサに関しては，ホストから [0x64] の UDP による命令を送信するとそのホストに 15 から 30 のランダムな値を返す．
  仮想ディスプレイは，受信した UDP の 1 オクテット目の値を表示する．

なお，以下に述べる実験で使用したデバイスは，実験 4 では仮想マシン，仮想デバイスを用い，それ以外の実験は全て実機で行った．また，中央コントローラからネットワーク機器と，ホストで実行されるアプリケーションの設定については，Java のソケットを用いたプログラムを実装し，中央コントローラと各機器がコネクションを貼ることで行っている．

6.2　個々のデバイス制御とデータ処理のデータフロー展開

この節では，管理者が個々のデバイス制御とデータ処理をデータフロースクリプトに明示的に記述し，様々なネットワーク構成下でもデータフロースクリプトの記述通りにデータフローが展開できることを確認する．

6.2.1　実験 1: 単純なフロー

この実験は，ネットワーク機器が存在しないネットワーク上で，温度センサの値をそのままディスプレイへ出力するという単純なデータフローを展開するものである．データフロースクリプトはセンサの値をアクチュエータへ出力するという図 6.1 のようなものである．

図 6.1　単純なフローのデータフロースクリプト

図 6.2 はデータフローが展開されている様子であり，温度センサ S1 を制御しているホスト H1 が S1 から温度の値を取得し，ディスプレイ A1 を制御する H2 へ値を転送している．H2 は受信した値を A1 へ転送し，A1 が値を出力している．

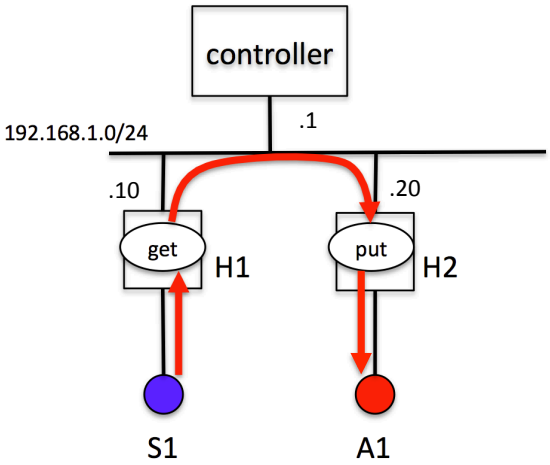

図 6.2　ネットワーク機器が存在しない環境下で展開される単純なデータフロー

図のようにネットワーク機器が存在しない環境下において温度センサ S1 からディスプレイ A1 へデータフローが展開されていることを確認した．

6.2.2　実験 2: データ処理を含むフロー

この実験では実験 1 と同様に，ネットワーク中にネットワーク機器が存在しない環境下で，センサからアクチュエータへのデータフロー中にデータ処理を含むデータフロープログラムをネットワーク上へ展開できることを確認するもので，データフロープログラムは図 6.3 のように，温度センサ S1，S2 の値の平均値を計算し (avg) ディスプレイ A1 へ出力するというものである．

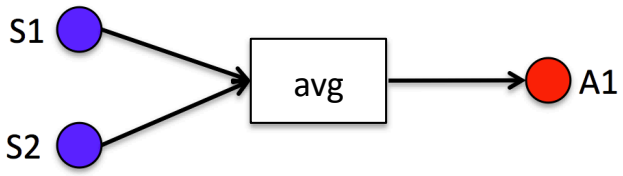

図 6.3　データ処理を含むデータフロースクリプト

図 6.4 がネットワーク機器が存在しないネットワークにおいて，センサからアクチュエータへデータフローが展開されている様子で，平均値を計算している avg の chunk が H3 へ割り当てられている場合を示している．

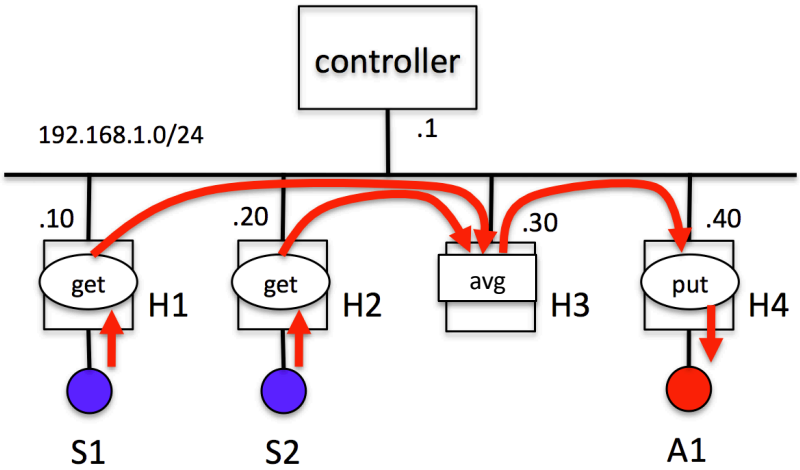

図 6.4　ネットワーク機器が存在しない環境下でデータ処理を含むデータフローが展開されている様子

ネットワーク機器が存在しない環境下で，センサからアクチュエータへのデータ処理を含むデータフローが展開されていることを確認した．

6.2.3　実験 3: データ処理を含み NAT がある

この実験では，ネットワーク上に NAT ルータが存在し，デバイスがプライベートアドレス空間に存在する場合にもデータフローを展開できることを確認するためのものである．データフロースクリプトは実験 2 と同様，図 6.3 のものを用い，図 6.5 がデータフローが展開されている様子である．

図中の NAT2, NAT3 のホストへの矢印はポートフォワードが設定されていることを示しており，適切にポートフォワードの設定がされていることが確認できた．NAT2, NAT3 で実行されたポートフォワードの設定はそれぞれ図 6.6, 図 6.7 のようになっている．なお，NAT2 のグローバル IP アドレス側の NIC 名は eth0, プライベート IP アドレス側の NIC 名は eth1, NAT3 のグローバル IP アドレス側の NIC 名は eth0, プライベートアドレス側の NIC 名は eth3 となっている．

そして，各ホストに割り当てられたアプリケーションの内容は以下のようになっている．

- H1: 温度センサ S1 から取得した値を 192.168.1.20:10000 (NAT2) へ転送する．
- H2: 温度センサ S2 から取得した値を 192.168.1.20:10001 (NAT2) へ転送する．
- H3: 192.168.10.30:10000 で値が受信されるのを待機しており，受信した値の平均値を計算し，そ

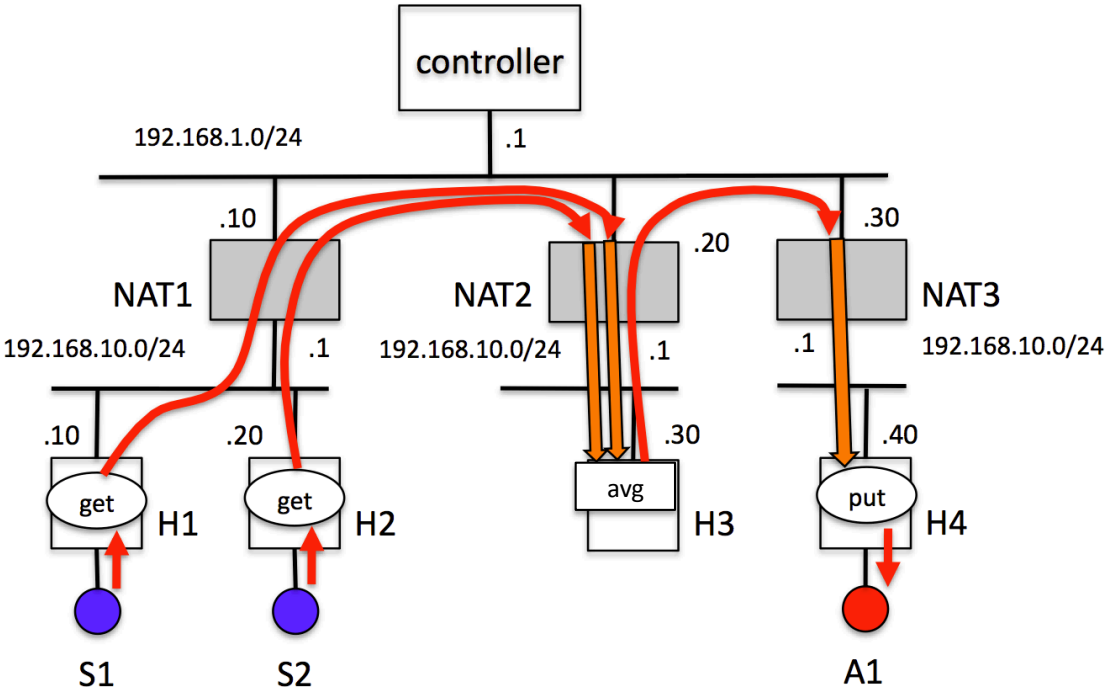

図 6.5　NAT ルータがあるネットワーク上でデータ処理を含むデータフローが展開される様子

```
iptables -t nat -A PREROUTING -p udp -i eth0 --dport 10000\
 -j DNAT --to-destination 192.168.10.30:10000
iptables -A FORWARD -i eth0 -o eth1 -p udp --dport 10000 -j ACCEPT
iptables -t nat -A PREROUTING -p udp -i eth0 --dport 10001\
 -j DNAT --to-destination 192.168.10.30:10000
iptables -A FORWARD -i eth0 -o eth1 -p udp --dport 10000 -j ACCEPT
```

図 6.6　NAT2 で実行されたポートフォワードの設定

```
iptables -t nat -A PREROUTING -p udp -i eth0 --dport 10000\
 -j DNAT --to-destination 192.168.10.40:10000
iptables -A FORWARD -i eth0 -o eth3 -p udp --dport 10000 -j ACCEPT
```

図 6.7　NAT3 で実行されたポートフォワードの設定

の結果を 192.168.1.30:10000 (NAT3) へ転送する．

- H4: 192.168.10.40:10000 で値が受信されるのを待機しており，値を受信したらその値をディスプレイ A1 へ出力する．

NAT2, NAT3 において受信されたデータは，それぞれ H3, H4 へポートフォワードされ，温度センサ S1 と S2 の平均値が A1 へ出力されることを確認した．

6.2.4　実験 4: データ処理を含み多段 NAT がある

これは，NAT ルータによるプライベートアドレス空間の中に，さらに NAT ルータが存在し，その NAT ルータが形成するプライベートアドレス空間内にアクチュエータが存在する場合を想定している．データフロースクリプトは図 6.3 であり，図 6.8 がデータフローが展開されている様子である．

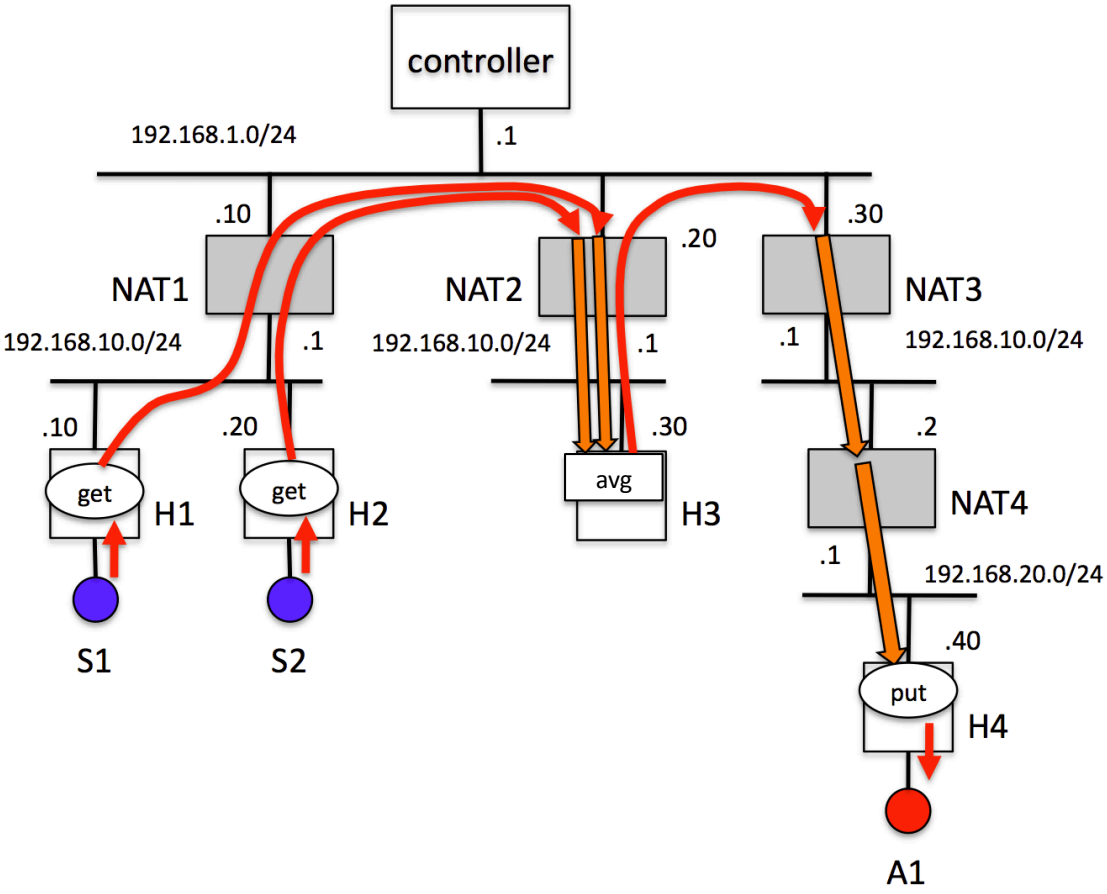

図 6.8　多段 NAT があるネットワーク上でデータ処理を含むデータフローが展開される様子

NAT にも適切にポートフォワードの設定が行われ，データフローが展開された．NAT で実行されたポートフォワードのためのコマンドは省略する．これにより，NAT が多段に存在してもセンサからアクチュエータへのデータフローが適切に展開されることが示された．なお，NAT ルータで実行されたポートフォワードの設定のコマンドは省略する．

6.2.5　実験 5: データ処理を含み ALG, FW, NAT がある

これは，ネットワーク機器として，NAT，FW，ALG が存在する場合を想定している．なお，本研究において ALG に行う設定は，あるポートで受信した値を別のホストへ転送する，というプロセスを立てるのみであり，アプリケーションレベルでのデータの操作は行わない．データフロースクリプトは図 6.3 であり，図 6.9 がデータフローが展開されている様子である．

ALG に行われた設定は，

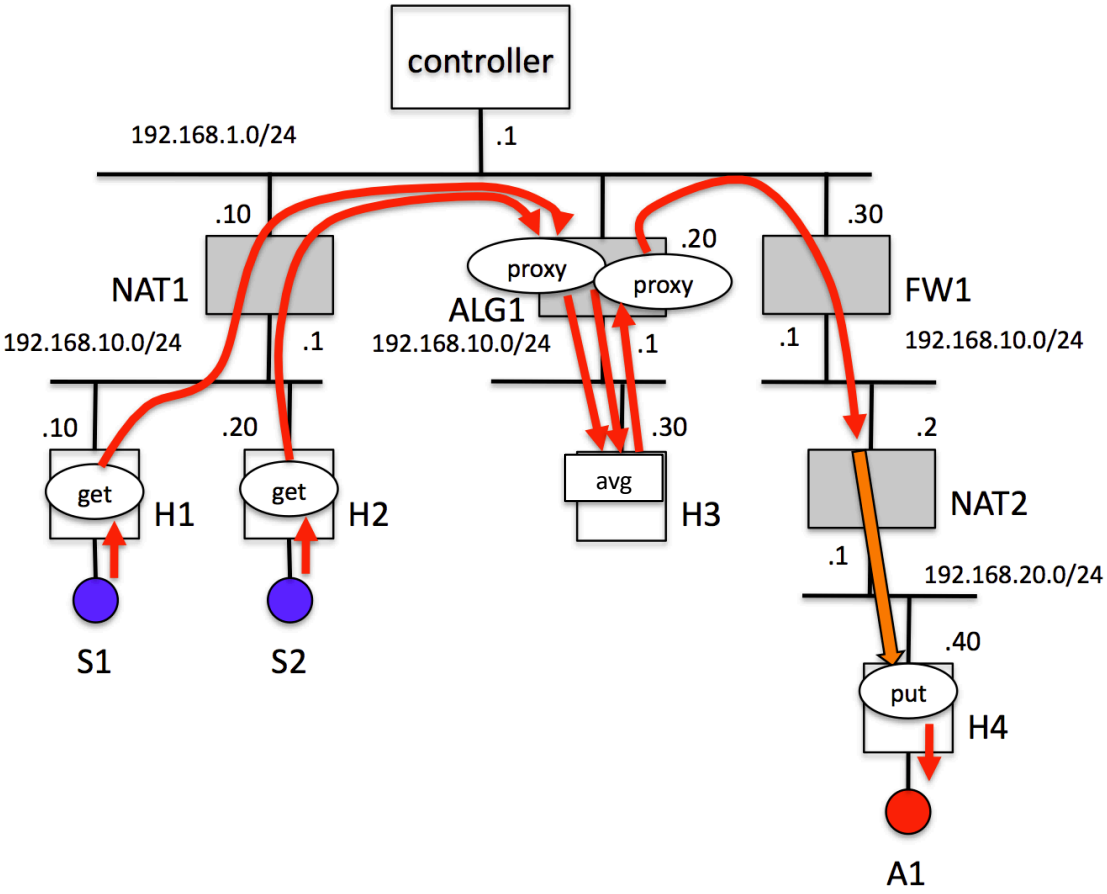

図 6.9　NAT, FW, ALG ルータが存在するネットワーク上でデータ処理を含むデータフローが展開される様子

```
iptables -A FORWARD -s 192.168.10.1 -p udp -d 192.168.10.2 --dport 10000 -j ACCEPT
```

図 6.10　FW1 で実行されたデータを通過させるための設定

```
sudo iptables -t nat -A PREROUTING -p udp -i eth6 --dport 10000\
 -j DNAT --to-destination 192.168.20.40:10000
sudo iptables -A FORWARD -i eth6 -o eth7 -p udp --dport 10000 -j ACCEPT
```

図 6.11　NAT2 で実行されたポートフォワードの設定

- 192.168.1.20:10000 (ALG1) で受信した値を 192.168.10.30:10000 (H3) へ転送する.
- 192.168.1.20:10001 (ALG1) で受信した値を 192.168.10.30:10000 (H3) へ転送する.
- 192.168.10.1:10000 (ALG1) で受信した値を 192.168.10.2:10000 (NAT2) へ転送する.

というプロセスを立ち上げるものである.

また，FW1, NAT2 で実行されたコマンドはそれぞれ図 6.10, 図 6.11 のようである.

そして，各ホストに割り当てられたアプリケーションの内容は以下のようになっている.

- H1: 温度センサ S1 から取得した値を 192.168.1.20:10000 (ALG1) へ転送する.

- H2: 温度センサ S2 から取得した値を 192.168.1.20:10001 (ALG1) へ転送する．
- H3: 192.168.10.30:10000 で値が受信されるのを待機しており，値を受信したらそれらの平均値を計算し，その結果を 192.168.10.2:10000 (NAT2) へ転送する．
- H4: 192.168.20.40:10000 で値が受信されるのを待機しており，値を受信したらその値をディスプレイ A1 へ出力する．

このようにして NAT，FW，ALG が存在するネットワーク環境下で，センサからアクチュエータへのデータフローが展開されることを確認した．

6.2.6　実験 6: 計算モジュールを含み ALG，FW，NAT があり，デバイスが移動

この実験は，デバイスが移動したときに，適切にデータフローが再展開されることを示すためのものである．図 6.9 のようにデータフローが展開されている状況で，A1 のデバイスを H4 から H3 の制御下へと移動させた．本研究の実装においては，ネットワークの構成要素の変更は変更を行った構成要素から中央コントローラへ明示的に示しているため，H4 は A1 を管理していないという情報を中央コントローラへ，H3 は A1 を管理しているという情報を中央コントローラへアップロードする．中央コントローラはその情報を受信し，図 6.12 のようにデータフローを再展開する．

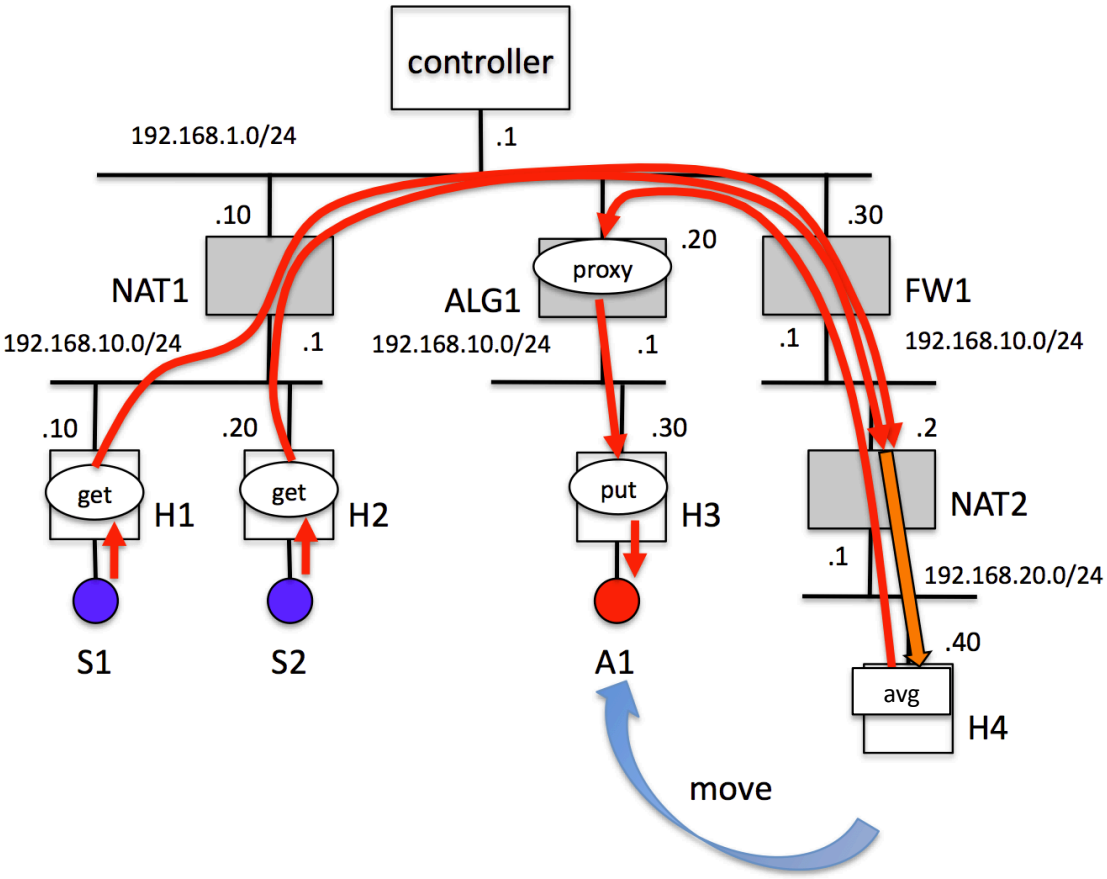

図 6.12　デバイスが移動した後にデータフローが再展開される様子

このようにデバイスが移動したときにもデータフローが適切に展開されることを確認した．

6.2.7　実験 7: 複雑な計算モジュールを含み，ALG，FW，NAT がある

これまでに述べた実験は，センサからアクチュエータへデータを送信する，センサの平均値をアクチュエータへ送信するという比較的単純なものであった．この実験では，センサとアクチュエータのデータフロー中に複数のデータ処理がある場合も適切に動作することを示すためのものである．図 6.13 のようなネットワークにおいて，図 6.14 のようなデータフローを展開することを考える．

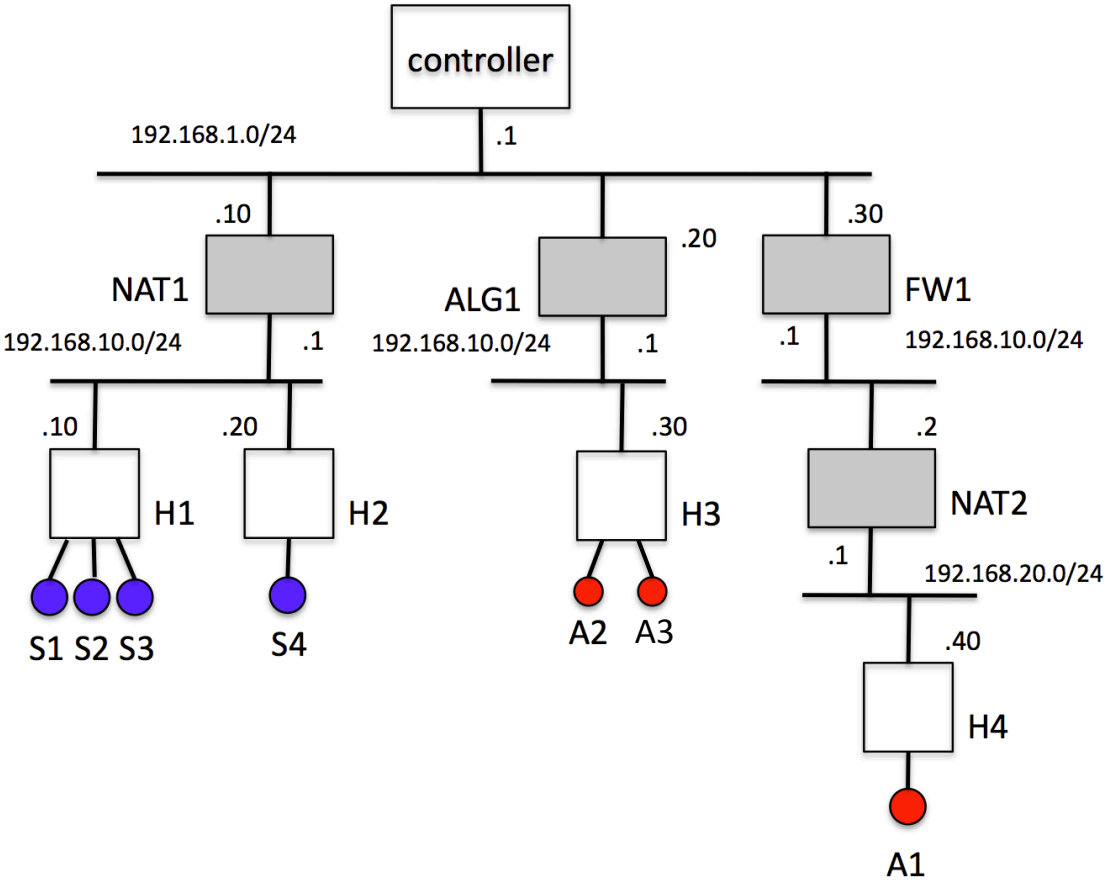

図 6.13　実験 7 のネットワーク構成図

センサは S1 から S4 の温度センサ 4 つ，アクチュエータは A1 から A3 のディスプレイ 3 つが存在し，S1, S2, S3 の平均値 (avg(S1, S2, S3) と示す) と S4 の値を比較して，avg(S1, S2, S3) と S4 の値が等しい場合は 0, avg(S1, S2, S3) が S4 より小さい値の場合は 1, 大きい場合は 2 をディスプレイ A1 へ出力し，A2 へは avg(S1, S2, S3) を，A3 へは S4 の値を出力するというデータフローを示すプログラムである．なお，各ネットワーク機器で実行されたコマンドは省略する．このようにデータ処理が複数含まれる複雑なデータフローも適切に展開できることを確認した．

6.3　デバイスの属性に基くデータフロー展開

6.2 で行った実験は，管理者が個々のデバイスとデータ処理をデータフロープログラムによって明示したものである．この実験では，属性が設定されたデバイスに対して，デバイスが移動，追加されたときも適切に一括処理ができることを示すための実験である．

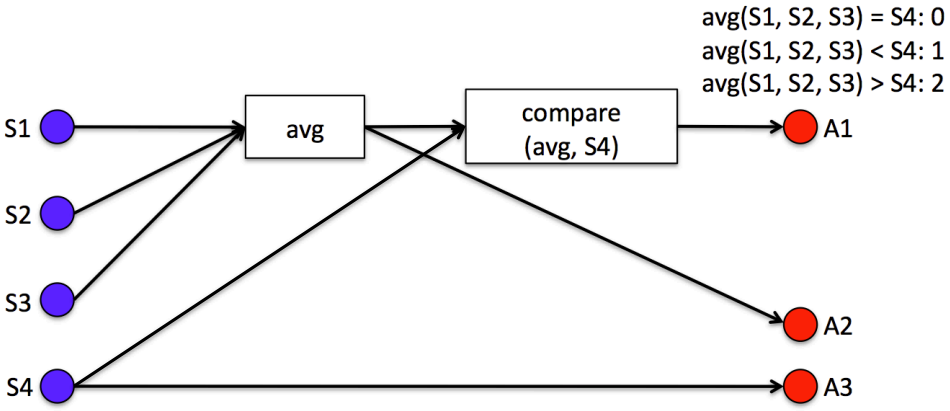

図 6.14　実験 7 で用いるデータフロースクリプトの概要

本実験で記述するデータフローは，属性 switch が設定されたスイッチ (センサ) が全て ON となっている場合に属性 light が設定された LED(アクチュエータ) が点灯し，属性 switch が設定されたスイッチの中で一つでも OFF となっていた場合には属性 light が設定された LED が消灯するというものである．4 種類の実験を行い，それぞれ 1. 最初にデータフロー展開されること，2. デバイスが移動した場合にデータフローが再展開すること，3. アクチュエータが追加されたときの動作，4. センサが追加されたときの動作を確認するために行った．

6.3.1　実験 9: 最初のデータフロー展開

これは，INPUTmultichannel, OUTPUTmultichannel 命令を用いて属性に対する一括処理が記述されたデータフロースクリプトを用いて，システムに存在するデバイスに応じたデータフローを展開できることを示すための実験である．図 6.15 のようなネットワーク構成を考える．

まず，中央コントローラは，属性 switch が設定されているスイッチが S1, S2 の 2 つ，属性 light が設定されている LED が A1 の 1 つ存在していることをネットワーク構成スクリプトから把握する．次に，その情報から属性に対する一括処理が書かれたデータフロースクリプトを，個々のデバイスの制御とデータ処理のデータフロースクリプトへと展開する．このとき，データ処理部は，スイッチの個数が 2 個なので，2 つ値を受信して共に ON であれば A1 の LED を点灯させる，そうでなければ消灯する，というデータ処理プログラムを展開する．図 6.16 が INPUTmultichannle, OUTPUTmultichannel 命令が個々のデバイスの制御とデータ処理のプログラムへ展開されたデータフロースクリプトの概念図である．実際に記述したスクリプトは付録 B.1 へ添付する．実験 9 から実験 12 は，全てこのデータフロースクリプトを使用している．図中の count という部分は，スイッチが全て ON かそうでないかを判断しているデータ処理プログラムである．

デバイスの状態に応じて，属性に対する一括処理を定義していたデータフロースクリプトが図 6.16 のようなスクリプトへ展開されると，中央コントローラは各ホスト，ネットワーク機器の動作を決定し設定することで図 6.17 のようなデータフローを展開する．

実験により，S1, S2 のスイッチが共に ON の場合のみ A1 の LED が点灯し，そうでない場合は消灯することを確認することで，データフローが適切に展開できることが確認できた．

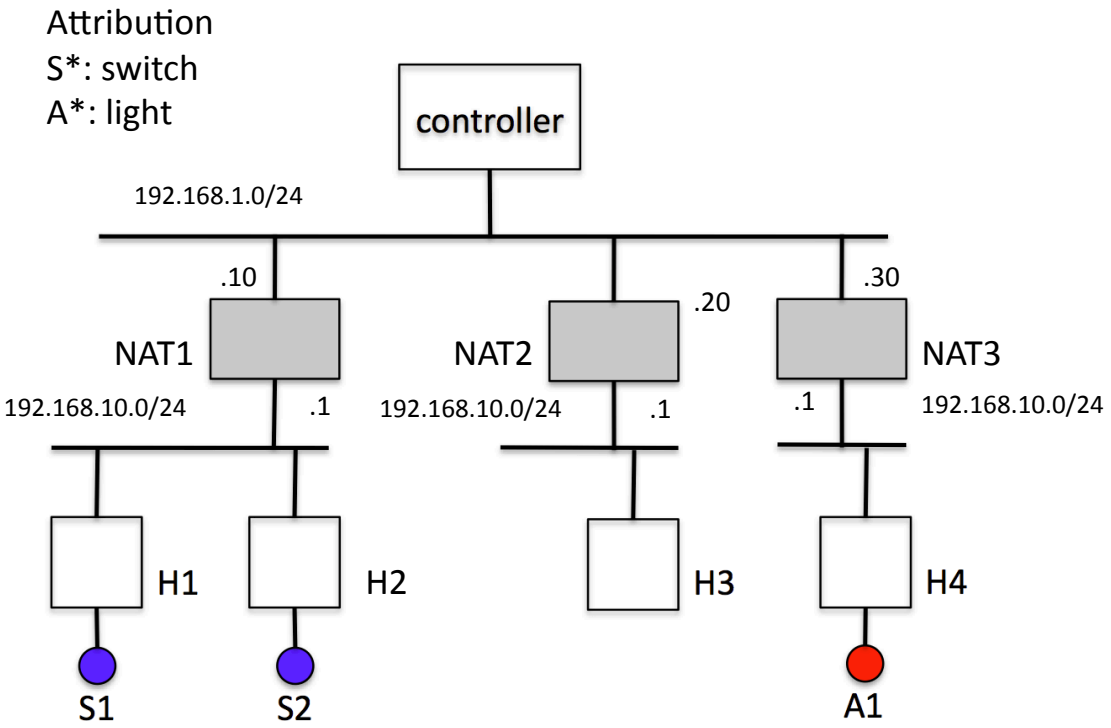

図 6.15　実験 9 のネットワーク構成

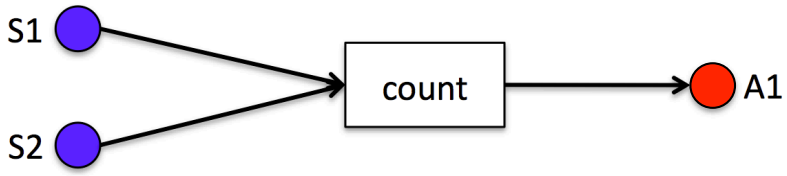

図 6.16　展開されるデータフロースクリプト

6.3.2　実験 10: デバイスの移動に対するデータフローの再展開

この実験は，A1 がホスト H4 からホスト H3 へ移動した場合に，デバイスの移動に合わせてデータフローが再展開されることを確認するためのものである．図 6.17 のネットワーク構成から，A1 をホスト H4 からホスト H3 へ移動させる．このとき H4 から中央コントローラへは，H4 は A1 を制御していないという情報を，H3 から中央コントローラへは，H3 が A1 を制御しているという情報を送信する．中央コントローラはネットワーク構成が変更したという情報を受信すると，新しいネットワーク構成から再度経路計算を行い，データフローの再展開を行う．図 6.18 がアクチュエータ A1 が移動した後にデータフローが再展開されている様子である．

count の計算結果の出力先が，A1 の移動に合わせて変更され，同時にデータフロー上にある NAT のポートフォワードの設定がされることで，移動した先の A1 へ結果が出力される．A1 が移動した後も，実験 9 と同様に全てのスイッチが ON となっているときのみ A1 の LED が点灯していることを確認することで，デバイスの移動後もデータフローが適切に展開されていることが確認できた．

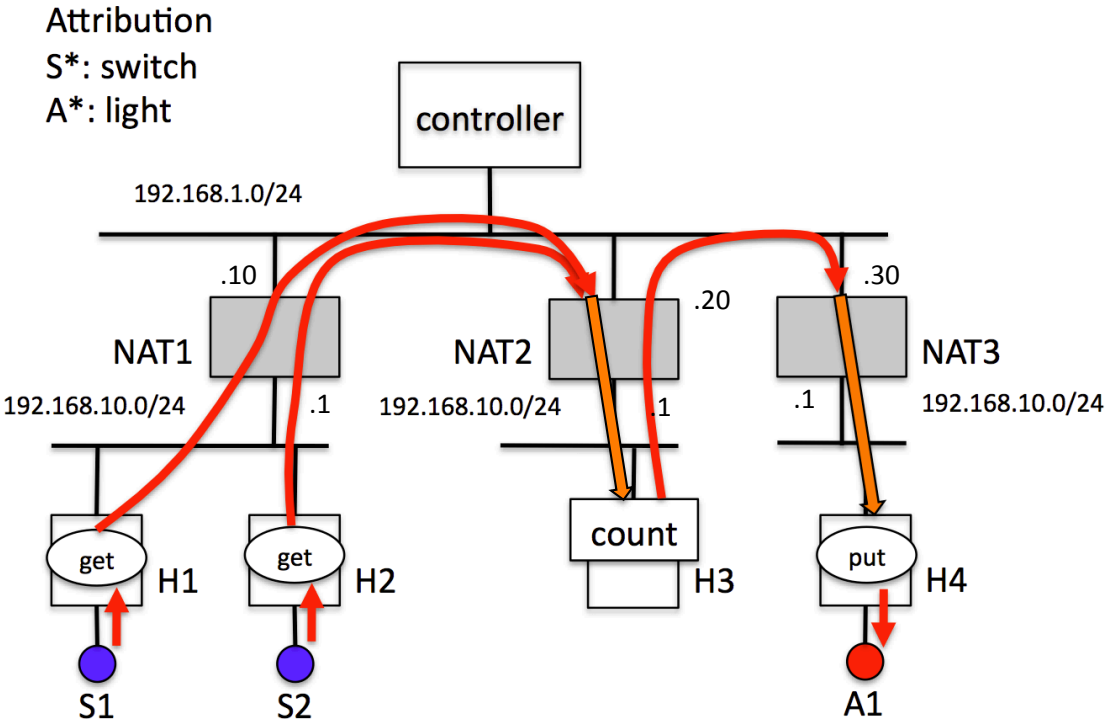

図 6.17　データフローが展開されている様子

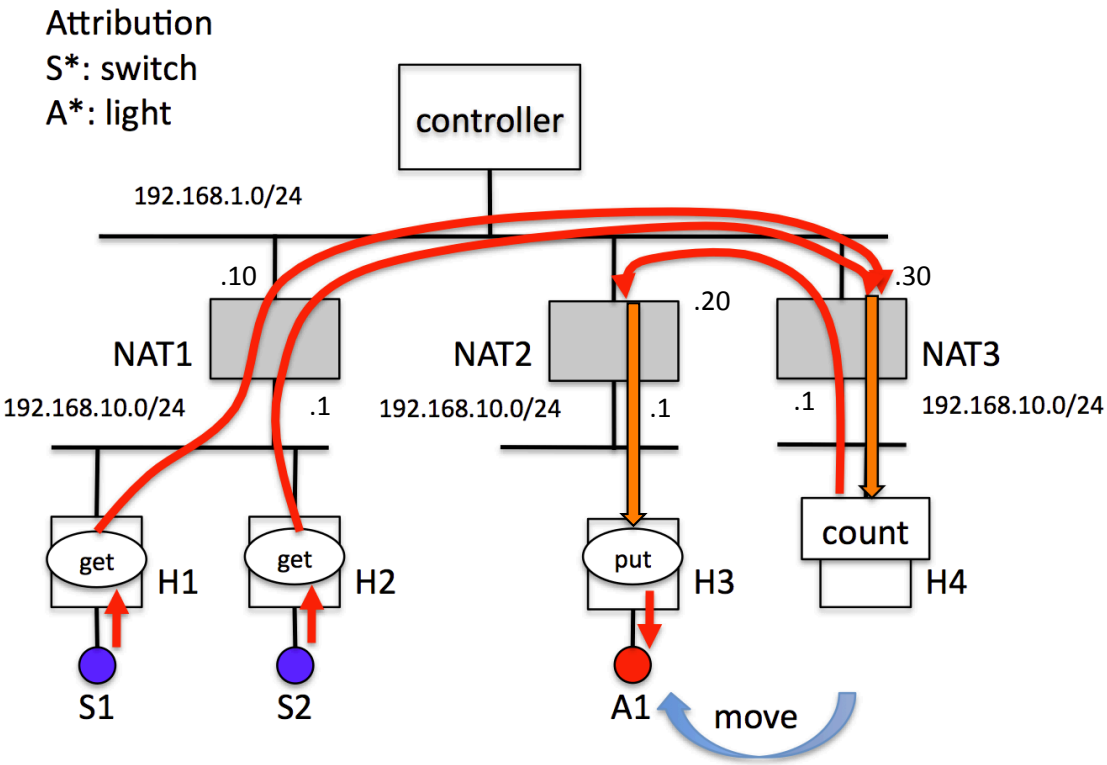

図 6.18　アクチュエータ A1 が移動した際のデータフローの再展開

6.3.3　実験 11: 属性が設定されているアクチュエータの追加

次に，属性 light が設定された A2 という LED をホスト H4 へと追加し，A2 へも count の結果が出力されることで，属性設定されたアクチュエータ全てに出力が行われることを示す．このとき，属性 switch

が設定されているスイッチは S1, S2 が，属性 light が設定されている LED は A1, A2 が存在していることから中央コントローラは一度図 6.19 のようなデータフロースクリプトを展開する．

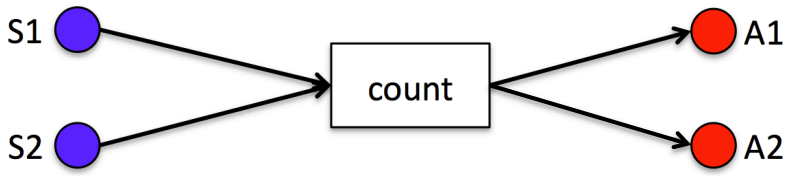

図 6.19　アクチュエータ A2 が追加されたときに展開されるデータフロースクリプト

そして中央コントローラは，このデータフロースクリプトの動作をネットワーク上へ展開する．A2 が追加され，そのアクチュエータに対しても count の結果が出力されたときのデータフローが展開されたときの様子は図 6.20 のよう．

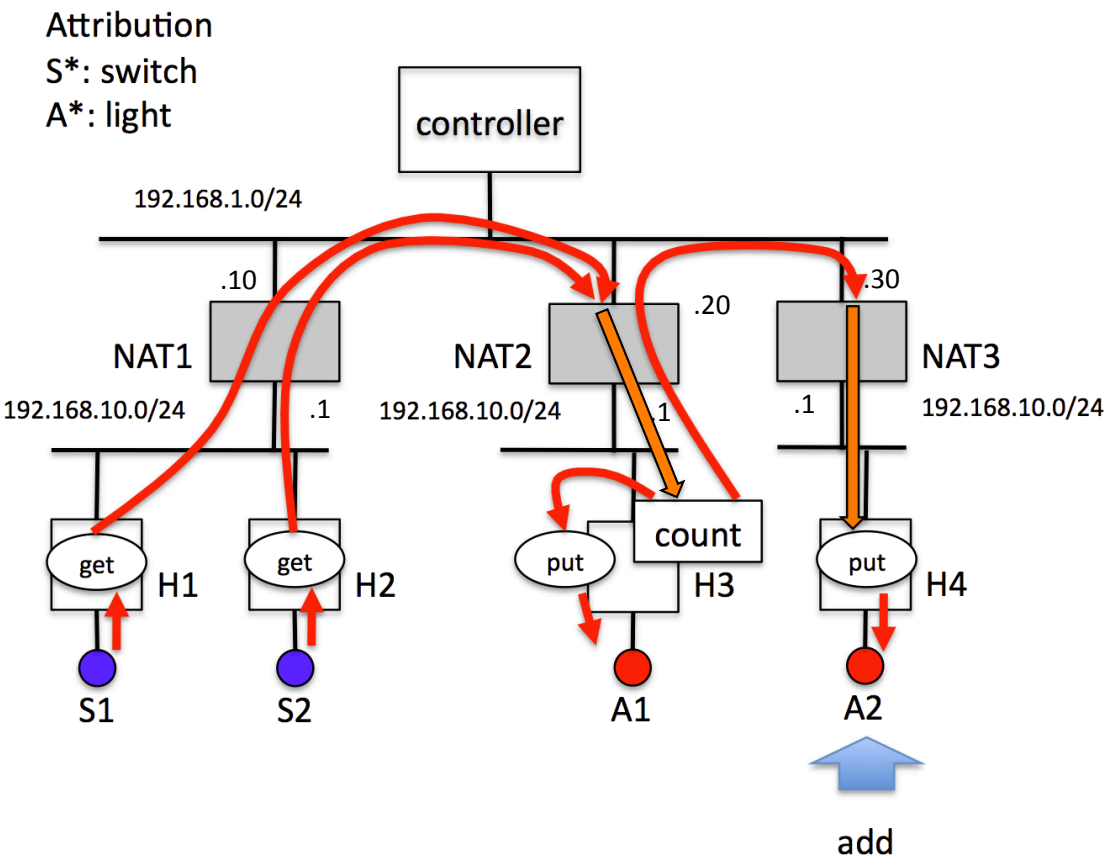

図 6.20　属性設定されたアクチュエータ A2 が追加された時のデータフロー展開

属性 light のアクチュエータが追加されたことにより count の計算結果の出力先が A1 と A2 となり，同時に NAT のポートフォワードの設定がされることで，A1, A2 共に結果が出力される．S1, S2 のスイッチが ON の時のみ A1 と A2 の LED が点灯し，それ以外は消灯することを確認することで，追加されたアクチュエータへもデータフローが展開されることを確認した．

6.3.4　実験 12: 属性が設定されているセンサの追加

次に，属性 switch が設定されているスイッチ S3, S4 が追加されたときに，これらのデバイスから取得された値もデータ処理の対象となることを示す実験を行った．この場合の動作は，属性 switch が設定されている S1 から S4 のスイッチ全てが ON になっている場合のみ A1 と A2 の LED が点灯するというものであり，センサの個数が変化するため，中央コントローラはデータ処理を行う count のデータ処理プログラムもそれに合わせて展開する．属性 switch が設定されているスイッチ S3, S4 が追加されたとき，中央コントローラが展開するデータフロースクリプトの概念図は図 6.21 のよう．

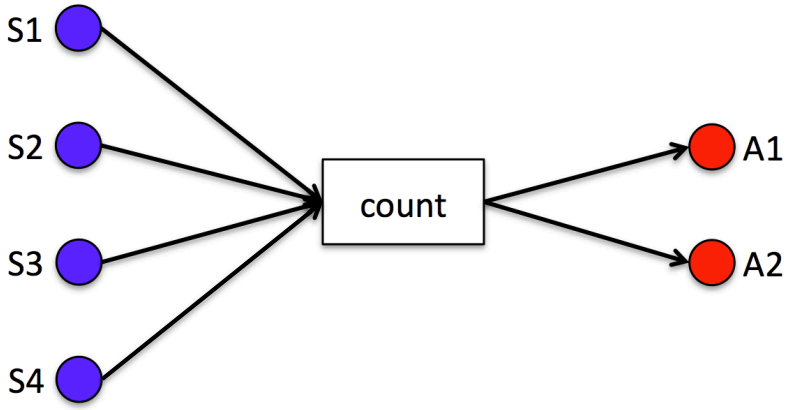

図 6.21　センサ S3, S4 が追加されたときに展開されるデータフロースクリプト

実際に展開されたデータフロースクリプトは付録 B.2 へ添付する．S3, S4 から取得される値もデータ処理の対象となり，count の部分もセンサの個数分の 4 つの値を受信して結果を出力するような判定となっており，中央コントローラは，このデータフロースクリプトの動作をネットワーク上へ展開する．図 6.22 がデータフローが展開されたときの様子である．

スイッチ S3，S4 の値もデータ処理の対象となり，S1 から S4 のスイッチ全てが ON になっているときのみ A1 と A2 の LED が点灯し，それ以外の場合は消灯することで，データフローが適切に展開できていることを確認した．

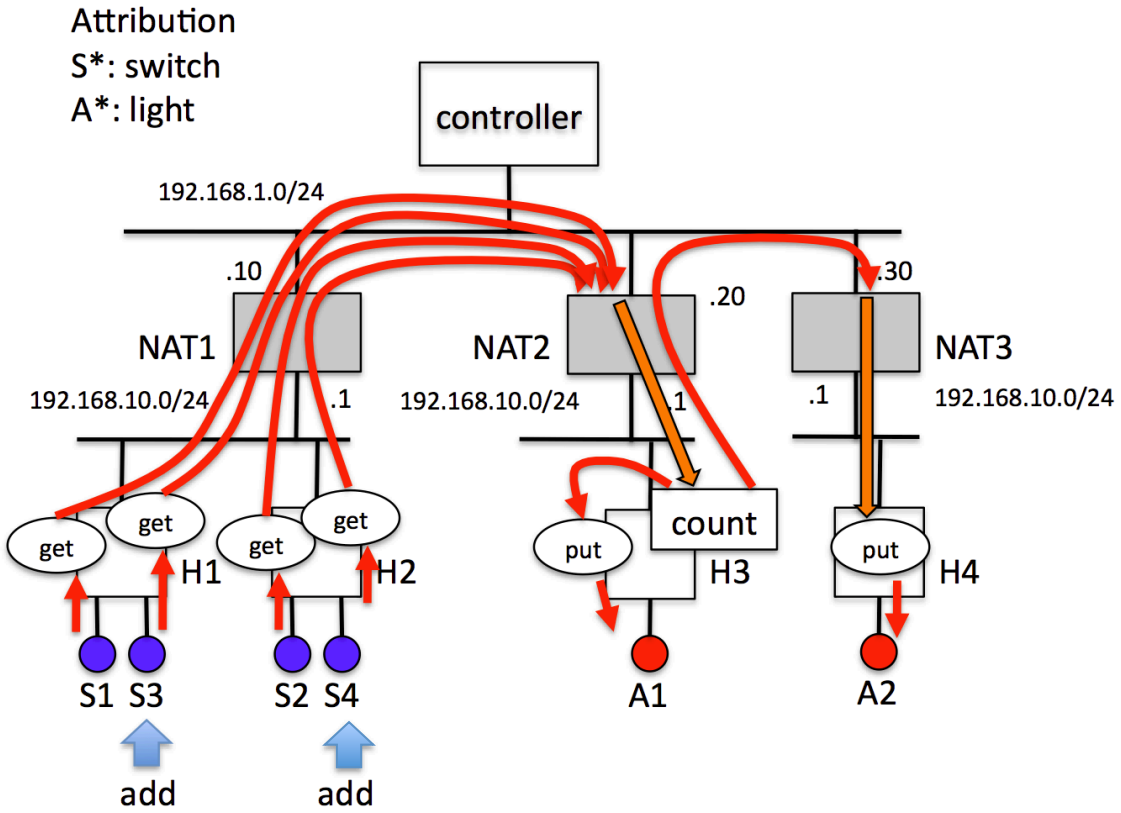

図 6.22　属性設定されたセンサ S3, S4 が追加された時のデータフロー展開

第 7 章

考察

7.1　ネットワーク機器が存在する環境下でのデータフロー展開について

実験 2 から実験 5 についてはネットワーク構成が異なるものの，管理者が記述するデータフロースクリプトは図 6.3 のもので共通である．これは，アプリケーション間のデータの転送のためのトランスポート層以下の設定と，データフロー中にあるネットワーク機器の設定がデータフローに合わせて自動的に行われることで，アプリケーションの分散展開の際に必要なトランスポート層以下の全ての設定が中央コントローラによってなされることにより可能となる (図 7.1).

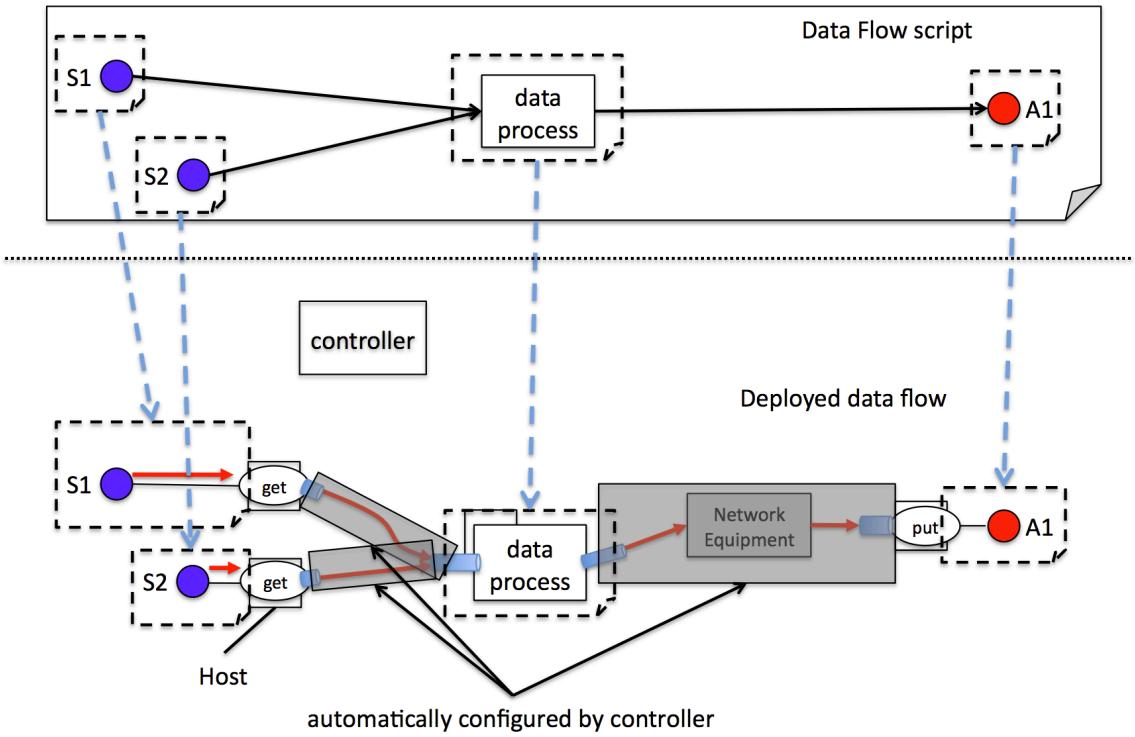

図 7.1　データフロースクリプトが分散展開される際の，アプリケーション間のデータフローとネットワーク機器の中央コントローラによる自動設定

よって管理者はデバイス制御とデータ処理のアプリケーション層のみを考慮してデータフロースクリプトを記述することができ，さらにこのスクリプトはネットワークトポロジに依らない．特に，実験 3 から実験 5 においては，NAT，FW，ALG などのネットワーク機器がネットワーク中に含まれているが，こ

れらの機器も中央コントローラによりデータフローに合わせた設定を行うことでデータフローが展開されていることが示されている．また，実験 7 で用いた図 6.14 のような複雑なデータフロースクリプトも展開できることが確認でき，管理者が柔軟なセンサ・アクチュエータデバイスの制御アプリケーションを展開できることがわかった．

7.2　デバイスの状態変化に対するデータフロー展開

7.1 で述べた通り，データフロー展開の際のトランスポート層以下の設定は全て中央コントローラによって自動的に設定されるため，管理者はネットワークトポロジに依らないデータフロースクリプトを記述できる．これは，ネットワークトポロジに依らずにデバイスを名前などの識別子で扱えることが可能となり，デバイス操作の抽象化が可能となっていることを示す．

これにより，実験 6, 実験 10 で示された通り，デバイスを移動させた際にもデータフロースクリプトを変更することなく，中央コントローラにデバイスが移動したというネットワークトポロジの変更情報をアップロードするだけで，その変更に応じたデータフローを展開できる．また，デバイスを属性名という識別子で一括で処理を行うことにより，実験 11, 実験 12 のようデバイス同士を属性設定に基づいて協調動作をさせることも可能となる．

7.3　アプリケーションの分散実行におけるデータフロー展開

実験により，提案したアーキテクチャでセンサ・アクチュエータ間のデータ処理を含むデータフローを展開する際に必要な，NAT，FW，ALG といったネットワーク機器の設定が自動的に行われ，ネットワーク機器が存在する環境下でも良好なデータフロー展開が可能となったことが確認された．これは，データフロー展開におけるトランスポート層以下の設定を全て自動化することで，アプリケーションレベルのデータフロー定義でデータフローシステムの展開が可能となることを示している．本研究ではセンサ・アクチュエータデバイスを用いた BAS 展開を対象としており，デバイスの制御とデータ処理のアプリケーションを分散展開することでセンサ・アクチュエータ間のデータフローを展開することを想定しているが，本研究が対象としているネットワーク機器の設定は，トランスポート層以下の設定であり，アプリケーションには依存しないため，他のアプリケーションの分散展開への適用も可能と考えられる．

第 8 章

まとめ

本研究では，ビルディングオートメーションシステムにおいて，データ処理を含んだセンサからアクチュエータへのデータフローを展開する状況を想定し，データフロー上の NAT，ファイアウォール，Application Level Gateway ルータなどのネットワーク機器と，エンドホスト上で実行されるデバイス制御とデータ処理のアプリケーションを制御することでデータフローを柔軟に展開するための手法を提案した．本研究の特徴は，エンドホストで分散実行されているアプリケーションとネットワーク機器を統合的に扱い，自動的に設定を行うことで，ネットワーク機器が存在する環境下でもアプリケーション間のデータフロー展開を実現可能としていることである．

提案手法は中央管理方式のアーキテクチャであり，管理者が記述したデータフロー定義を実現するために，中央コントローラがデータフロー展開に必要なネットワーク機器とエンドホストのアプリケーションの設定を計算し，制御するというものである．提案手法を実装し実機を用いて動作検証を行うことで提案手法のアーキテクチャにより，ネットワーク機器が存在する環境下でもセンサからアクチュエータのデータフローが良好に展開できることを確認した．また，ネットワーク機器がデータフローに合わせて自動的に設定されることでより柔軟なセンサネットワークアプリケーションを展開できる事を示すために，センサ・アクチュエータデバイスを移動，追加，削除したときの動作検証も行った．

謝辞

本論文を執筆するに当たり，大変多くの方からご指導，ご協力を頂きました．ここに心より感謝の意を表します．まず, 研究を進めるにあたり広い見地と深い知識により的確なご指導, ご助言をいただいた江崎浩教授に深く感謝致します. 研究と運用，後輩への指導や教育の在り方についてもたくさんのご指導を頂いた東京大学大学院工学系研究科国際工学教育推進機構特任講師，土本康生博士および東京大学生産技術研究所電子計算機室助教，山本成一博士に深く感謝いたします.

研究を進めるにあたり, 温かい激励のお言葉やご指摘を頂いた Thomas Silverston 博士, 藤田祥博士, 土井裕介博士, 金海好彦氏, 白井俊宏氏，浅井大史氏, 肥村洋輔氏, 趙越氏, 下忠健一氏に深く感謝致します.

本論文を執筆するにあたり, 研究の方向性のみならず論文の書き方やプレゼンテーションの方法など幅広くご指導頂いた落合秀也博士に深く感謝申し上げます. 落合氏には議論や質問に長い時間お付き合い頂きました.

研究のアドバイスや相談をはじめ, 学生生活や研究生活のサポートをして下さった杉田毅博氏, 川上雄也氏, 唐明シン氏, Sathita Kaveevivitchai 氏, Leela-amornsin Lertluck 氏に深く感謝致します.

研究室の同期として苦楽を共にした呉和賢氏, 本館拓也氏に感謝致します. Luciano Aparicio 氏, David Jageberg 氏, Jonas Johansson 氏，Romain Fontugne 氏, Badertscher Stefan Jurg 氏, Trinh Minh Tri 氏に感謝致します.

研究に関する議論やサポート，研究室の仕事などに積極的に取り組んで大きな力になってくれた後輩の，石田渉君，石橋尚武君，大津恭平君，園田大剛君，小坂良太君, 正原竜太君, 李聖年君, 林東權君，木下僚君， 東浦成良君，美嶋勇太朗君，朴成軍君に感謝いたします.

諸事務を通じて研究生活を支援してくださった江崎研究室秘書の高橋富美さん，田坂佳苗さん，岩井愛映子さんには大変感謝しております.

最後に，6 年間に渡る長い学生生活を支えてくれた家族や友人，お世話になった皆様にこの場をお借りして御礼申し上げます.

2012 年 2 月 8 日

川口 紘典

参考文献

[1] 平成 22 年度版 情報通信白書. http://www.soumu.go.jp/johotsusintokei/whitepaper/h22.html.

[2] Robin D. BACnet today. *Supplement to ASHRAE Journal*, Vol. 53, No. 11, pp. B4, B6–11, B12–17, nov 2011.

[3] グリーン東大工学部プロジェクト技術発展による地球環境問題への新たな取り組みの可能性. http://www.gutp.jp/.

[4] Ipv6 によるインターネットの利用高度化に関する研究会第三次報告書 〜ipv4 アドレス枯渇を迎えた課題解決先進国「日本」〜. http://www.soumu.go.jp/main_content/000138333.pdf.

[5] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, Vol. 38, pp. 69–74, March 2008.

[6] Hideya Ochiai Akihiro Sugiyama and Hiroshi Esaki. CCDM: central controller-based device management architecture and method to split management scripts. *In IEEE SAINT*, 2009.

[7] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). RFC 3489 (Proposed Standard), March 2003. Obsoleted by RFC 5389.

[8] R. Mahy, P. Matthews, and J. Rosenberg. Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). RFC 5766 (Proposed Standard), April 2010.

[9] UPnP FORUM. http://www.upnp.org/.

[10] G. Tsirtsis and P. Srisuresh. Network Address Translation - Protocol Translation (NAT-PT). RFC 2766 (Historic), February 2000. Obsoleted by RFC 4966, updated by RFC 3152.

[11] 秀也落合, 浩江崎. 高級言語によるネットワーク汎用 i/o 制御とプロトコル翻訳機構. 情報処理学会論文誌, Vol. 49, No. 10, pp. 3451–3461, oct 2008.

[12] Hyo sang Lim and Kyu young Whang. Continuous query processing in data streams using duality of data and queries. In *In Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 313–324, 2006.

[13] Don Carney, Ugur Cetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Michael Stonebraker, Nesime Tatbl, and Stan Zdonik. Monitoring stream - a new class of data management applications. *Proceedings of the 28th VLDB conference*, 2002.

[14] Daniel Abadi, Yanif Ahmad, Hari Balakrishnan, Magdalena Balazinska, Mitch Cherniack, Jeong hyon Hwang, Samuel Madden, Anurag Maskey, Er Rasin, Mike Stonebraker, Nesime Tatbul,

and Ying Xing. The aurora and borealis stream processing engines.

[15] Stan Zdonik Sbz, Stan Zdonik, Michael Stonebraker, Mitch Cherniack, Ugur C Etintemel, Magdalena Balazinska, and Hari Balakrishnan. The aurora and medusa projects. *IEEE Data Engineering Bulletin*, 2003.

[16] Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Keith Ito, Itaru Nishizawa, Justin Rosenstein, and Jennifer Widom. Stream: The stanford stream data manager. *IEEE Data Engineering Bulletin*, Vol. 26, No. 1, 2003.

[17] Jianjun Chen, David J. Dewitt, Feng Tian, and Yuan Wang. Niagaracq: A scalable continuous query system for internet databases. In *In SIGMOD*, pp. 379–390, 2000.

[18] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Sam Madden, Vijayshankar Raman, Fred Reiss, and Mehul Shah. Telegraphcq: Continuous dataflow processing for an uncertan world, 2003.

[19] Shinichi Yamada, Yousuke Watanabe, Hiroyuki Kitagawa, and Toshiyuki Amagasa. Location-based information delivery using stream processing engine. *Mobile Data Management, IEEE International Conference on*, Vol. 0, p. 57, 2006.

[20] 稲守孝之, 渡辺陽介, 北川博之, 天笠俊之. 分散ストリーム処理環境のための運用管理システムの提案. *DEWS2007*, 2 2007.

[21] J.D. Case, M. Fedor, M.L. Schoffstall, and J. Davin. Simple Network Management Protocol (SNMP). RFC 1098, April 1989. Obsoleted by RFC 1157.

[22] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Introduction to version 2 of the Internet-standard Network Management Framework. RFC 1441 (Historic), April 1993.

[23] J. Moy. OSPF version 2. IETF RFC 2328, April 1998.

[24] R. Enns. NETCONF Configuration Protocol. RFC 4741 (Proposed Standard), December 2006. Obsoleted by RFC 6241.

[25] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman. Network Configuration Protocol (NETCONF). RFC 6241 (Proposed Standard), June 2011.

[26] Virtualbox. `http://www.virtualbox.org/`.

[27] Arduino. `http://www.arduino.cc/`.

[28] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems (TODS), Volume 30 Issue 1*, pp. 122–173, 2005.

[29] S. Bushby. BACnet A Standard Communication Infrastructure for Intelligent Buildings. *National Institute of Standards and Technology*, pp. 5–6, 1997.

[30] D. Fisher. BACnet and LonWorks: a white paper. *AIRAH JOURNAL, 54, 2*, pp. 16–21, 2000.

[31] 落合秀也, 松浦知史, 砂原秀樹, 中山雅哉, 江崎浩. 広域センサネットワークの運用構造と多属性検索. 電子情報通信学会論文誌 B, J91-B, 10, pp. 1160–1170, 2008.

[32] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking Control of the Enterprise. *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications, Kyoto, Japan, ACM*, pp. 1–12, 2007.

対外発表

1. 川口紘典，落合秀也，江崎浩，「データストリーム処理における LAN の構成に基づくデータフローの決定」 IEICE IA 研究会 (2010.12)
2. 川口紘典「センサ・アクチュエータ間のデータフロー展開」 WIDE 研究会ポスター発表 (2011.12)
3. 川口紘典，落合秀也，江崎浩，「センサ・アクチュエータ間のデータフロー展開」 信学総合大会 (2012.3) (発表予定)

付録

A　データフロースクリプトのサンプルコード

```
<?xml version="1.0" encoding="UTF-8"?>
<progn>
  <chunk>
    <while>
      <eq>
        <int>1</int>
        <int>1</int>
      </eq>
      <progn>
        <sleep><int>1000</int></sleep>
        <INPUT from="S1"/>
        <post channel="A" name="INPUT"/>
      </progn>
    </while>
  </chunk>

  <chunk>
    <while>
      <eq>
        <int>1</int>
        <int>1</int>
      </eq>
      <progn>
        <sleep><int>1000</int></sleep>
        <INPUT from="S2"/>
        <post channel="A" name="INPUT"/>
      </progn>
    </while>
  </chunk>

  <chunk>
    <while>
      <eq>
        <int>1</int>
        <int>1</int>
      </eq>
      <progn>
        <catch channel="A" name="response" />
        <catch channel="A" name="response2"/>
        <setq name="temp1">
          <byteArrayGet>
            <getq name="response"/>
            <int>0</int>
          </byteArrayGet>
        </setq>
        <setq name="temp2">
          <byteArrayGet>
            <getq name="response2"/>
            <int>0</int>
          </byteArrayGet>
        </setq>

        <setq name="average_temp">
```

```
          <divide>
            <plus>
              <getq name="temp1"/>
              <getq name="temp2"/>
            </plus>
            <int>2</int>
          </divide>
        </setq>

        <setq name="avg_array">
          <byteArrayAppend>
            <byteArray/>
            <getq name="average_temp"/>
          </byteArrayAppend>
        </setq>

        <post channel="B" name="avg_array"/>
      </progn>
    </while>
  </chunk>

  <chunk>
      <while>
        <eq>
        <int>1</int>
        <int>1</int>
      </eq>
        <progn>
          <catch channel="B" name="avg_array"/>
          <OUTPUT to="A1">
            <ARG>
              <getq name="avg_array"/>
            </ARG>
          </OUTPUT>
        </progn>
    </while>
  </chunk>
</progn>
```

B 属性設定に基づくデータフロー展開の実験で用いたデータフロースクリプト

B.1 実験 9 から実験 12 で用いたデータフロースクリプト

```
<?xml version="1.0" encoding="UTF-8"?>
<progn>
  <chunk>
    <while>
      <eq>
        <int>1</int>
        <int>1</int>
      </eq>
      <progn>
        <setq name="on_sum">
          <int>0</int>
        </setq>
        <setq name="count">
          <int>0</int>
        </setq>
        <INPUTmultichannel type="switch" interval="1000">
          <progn>
            <setq name="on_sum">
              <plus>
                <byteArrayGet>
                  <getq name="new_val"/>
                  <int>0</int>
                </byteArrayGet>
                <getq name="on_sum"/>
              </plus>
            </setq>
            <setq name="count">
              <plus>
                <getq name="count"/>
                <int>1</int>
              </plus>
            </setq>
          </progn>
        </INPUTmultichannel>

        <if>
          <eq>
            <getq name="on_sum"/>
            <getq name="count"/>
          </eq>
          <progn>
            <setq name="output">
              <byteArrayAppend>
                <byteArray/>
                <byte>1</byte>
              </byteArrayAppend>
            </setq>
            <OUTPUTmultichannel type="light" name="output"/>
          </progn>
        </if>

        <if>
          <neq>
            <getq name="on_sum"/>
            <getq name="count"/>
          </neq>
          <progn>
            <setq name="output">
              <byteArrayAppend>
                <byteArray/>
                <byte>0</byte>
              </byteArrayAppend>
```

```
            </setq>
            <OUTPUTmultichannel type="light" name="output"/>
          </progn>
        </if>
      </progn>
    </while>
  </chunk>
</progn>
```

B.2 実験 12 で展開されたデータフロースクリプト

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<progn>
  <chunk>
    <while>
      <eq>
        <int>1</int>
        <int>1</int>
      </eq>
      <progn>
        <sleep>
          <int>1000</int>
        </sleep>
        <INPUT from="S1"/>
        <post channel="0" name="INPUT"/>
      </progn>
    </while>
  </chunk>

  <chunk>
    <while>
      <eq>
        <int>1</int>
        <int>1</int>
      </eq>
      <progn>
        <sleep>
          <int>1000</int>
        </sleep>
        <INPUT from="S2"/>
        <post channel="0" name="INPUT"/>
      </progn>
    </while>
  </chunk>

  <chunk>
    <while>
      <eq>
        <int>1</int>
        <int>1</int>
      </eq>
      <progn>
        <sleep>
          <int>1000</int>
        </sleep>
        <INPUT from="S3"/>
        <post channel="0" name="INPUT"/>
      </progn>
    </while>
  </chunk>

  <chunk>
    <while>
      <eq>
        <int>1</int>
        <int>1</int>
      </eq>
      <progn>
        <sleep>
          <int>1000</int>
        </sleep>
```

```
        <INPUT from="S4"/>
        <post channel="0" name="INPUT"/>
      </progn>
    </while>
  </chunk>

  <chunk>
    <while>
      <eq>
        <int>1</int>
        <int>1</int>
      </eq>
      <progn>
        <setq name="on_sum">
          <int>0</int>
        </setq>
        <setq name="count">
          <int>0</int>
        </setq>
        <progn>
          <catch channel="0" name="new_val"/>
          <progn>
            <setq name="on_sum">
              <plus>
                <byteArrayGet>
                  <getq name="new_val"/>
                  <int>0</int>
                </byteArrayGet>
                <getq name="on_sum"/>
              </plus>
            </setq>
            <setq name="count">
              <plus>
                <getq name="count"/>
                <int>1</int>
              </plus>
            </setq>
          </progn>
          <catch channel="0" name="new_val"/>
          <progn>
            <setq name="on_sum">
              <plus>
                <byteArrayGet>
                  <getq name="new_val"/>
                  <int>0</int>
                </byteArrayGet>
                <getq name="on_sum"/>
              </plus>
            </setq>
            <setq name="count">
              <plus>
                <getq name="count"/>
                <int>1</int>
              </plus>
            </setq>
          </progn>
          <catch channel="0" name="new_val"/>
          <progn>
            <setq name="on_sum">
              <plus>
                <byteArrayGet>
                  <getq name="new_val"/>
                  <int>0</int>
                </byteArrayGet>
                <getq name="on_sum"/>
              </plus>
            </setq>
            <setq name="count">
              <plus>
                <getq name="count"/>
                <int>1</int>
              </plus>
```

```
          </setq>
        </progn>
        <catch channel="0" name="new_val"/>
        <progn>
          <setq name="on_sum">
            <plus>
              <byteArrayGet>
                <getq name="new_val"/>
                <int>0</int>
              </byteArrayGet>
              <getq name="on_sum"/>
            </plus>
          </setq>
          <setq name="count">
            <plus>
              <getq name="count"/>
              <int>1</int>
            </plus>
          </setq>
        </progn>
      </progn>

      <if>
        <eq>
          <getq name="on_sum"/>
          <getq name="count"/>
        </eq>
        <progn>
          <setq name="output">
            <byteArrayAppend>
              <byteArray/>
              <byte>1</byte>
            </byteArrayAppend>
          </setq>
          <post channel="1" name="output"/>
        </progn>
      </if>

      <if>
        <neq>
          <getq name="on_sum"/>
          <getq name="count"/>
        </neq>
        <progn>
          <setq name="output">
            <byteArrayAppend>
              <byteArray/>
              <byte>0</byte>
            </byteArrayAppend>
          </setq>
          <post channel="2" name="output"/>
        </progn>
      </if>
    </progn>
  </while>
</chunk>

<chunk>
  <while>
    <eq>
      <int>1</int>
      <int>1</int>
    </eq>
    <progn>
      <catch channel="1" name="response"/>
      <OUTPUT to="A2">
        <ARG>
          <getq name="response"/>
        </ARG>
      </OUTPUT>
    </progn>
  </while>
```

```
  </chunk>

  <chunk>
    <while>
      <eq>
        <int>1</int>
        <int>1</int>
      </eq>
      <progn>
        <catch channel="1" name="response"/>
        <OUTPUT to="A1">
          <ARG>
            <getq name="response"/>
          </ARG>
        </OUTPUT>
      </progn>
    </while>
  </chunk>

  <chunk>
    <while>
      <eq>
        <int>1</int>
        <int>1</int>
      </eq>
      <progn>
        <catch channel="2" name="response"/>
        <OUTPUT to="A1">
          <ARG>
            <getq name="response"/>
          </ARG>
        </OUTPUT>
      </progn>
    </while>
  </chunk>

  <chunk>
    <while>
      <eq>
        <int>1</int>
        <int>1</int>
      </eq>
      <progn>
        <catch channel="2" name="response"/>
        <OUTPUT to="A2">
          <ARG>
            <getq name="response"/>
          </ARG>
        </OUTPUT>
      </progn>
    </while>
  </chunk>
</progn>
```