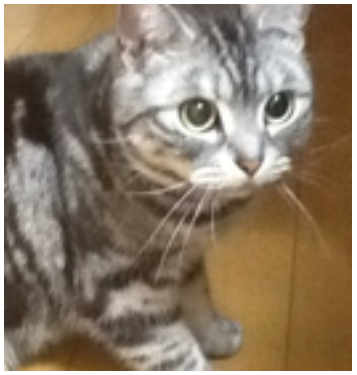
オープンソースによる OpenFlow1.3環境の構築

- Group Tableの活用事例 -

2013.5.27

@ttsubo

自己紹介



- 数年前は、通信事業者向けネットワークエンジニアでした。
- 最近は、データセンタ系ネットワークエンジニアに軸足のシフトを試みてます。IaaS管理基盤技術として、OpenStack等を勉強中。
- さらに「これからの時代、ネットワーク屋も、プログラミング必要だよね。」という風潮に感化されて、OpenFlowプログラミングも勉強中。

動機付け

効率的なトラフィックフロー制御を可能とするOpenFlow新技術が規定されているOpenFlow1.3がリリースされてから、１年が経過しました。

でも、その新技術の活用方法などを、OpenFlowコミュニティで共有できる場がありません。
OpenFlow新技術が活用されていない状況は、もったいないと感じています。

そこで、OpenFlow新技術の活用事例を、みなさんと共有したくて、OpenFlow1.3環境を構築してみました。

効率的なトラフィックフロー制御とは

OpenFlow1.3では、OpenFlow1.0課題克服を念頭にした
プロアクティブベースのフローエントリ機構が強化されております。
OpenFlowは、C-Plane/D-Plane分離技術と言われますが、D-Planeの効率化を図るためには、プロアクティブなフローエントリ機構の適用が必須だと思います。（さらに、リアクティブな特性を活用したC-Planeの自動化/自律化も必要ですね）

リアクティブベース

- OpenFlowスイッチ側での転送処理が完結せず、Packet-In動作によるコントローラへの問い合わせ処理がオーバヘッドとなり、NW全体のパフォーマンスが劣化する（＝スケールしない）
- 既存のネットワークとの相互接続時において、NWアドレス学習などC-Plane連携の活用事例が顕在化されつつある（ARP, BGP, OSPF, IGMP...）

プロアクティブベース

- OpenFlowスイッチ側での転送処理が完結するため、NW全体のパフォーマンス向上が期待できる
- Multi Table、Group Tableなどの新技術が活用することにより、Flow数の肥大化を抑止することも可能となる

OpenFlow環境構築にあたって

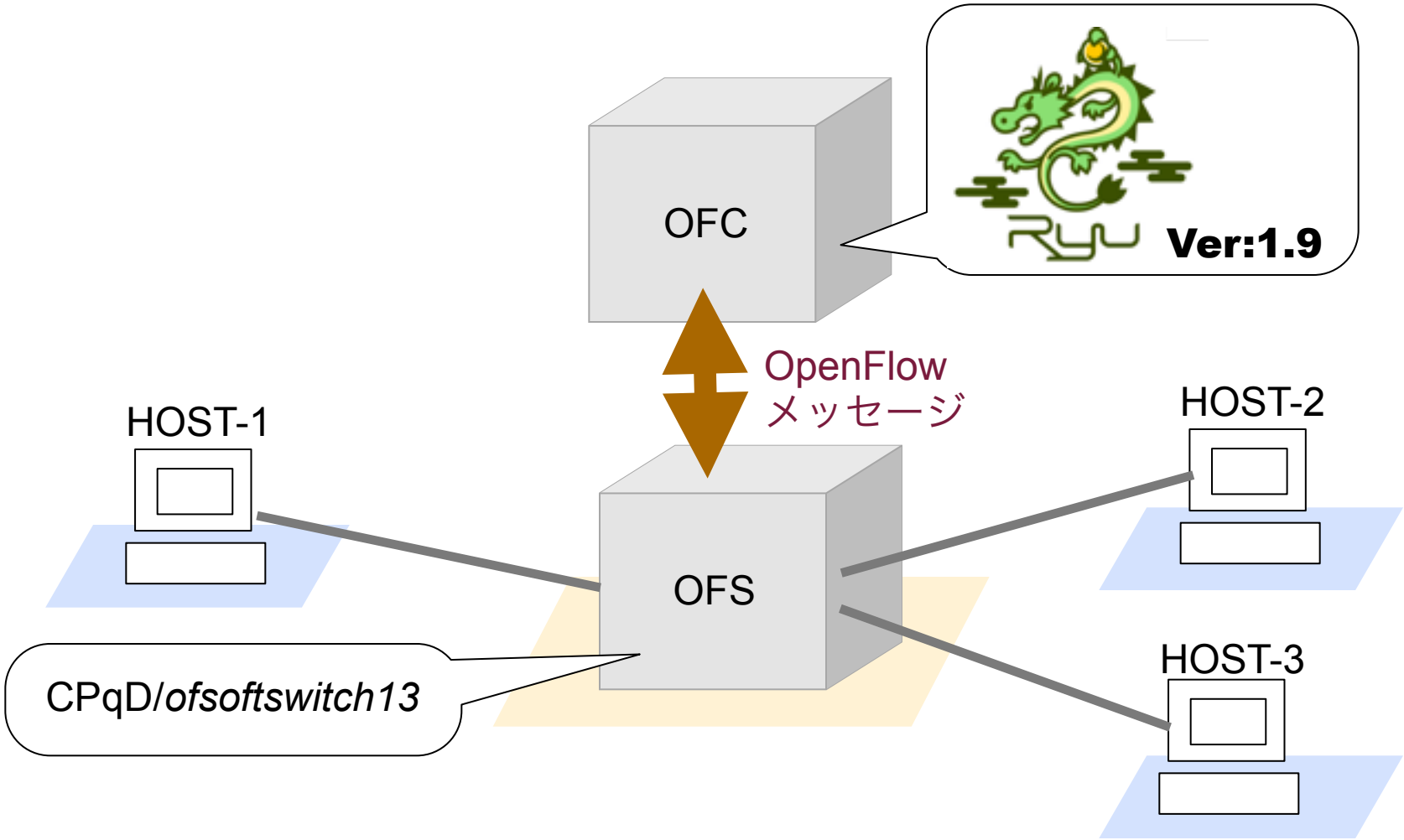
- オープンソース活用を大前提にTryしてみる！
- 最新のOpenFlow1.3で試してみる！
- OpenFlowらしい特性を確認してみる！

OpenFlow新技術として、Multi Table、Group Table、Meter、Multi Controller...
などが挙げられますが、

今回は、OpenFlowプロアクティブなフロー制御を実現する「GroupTable」を中心に、動作確認してみました。

OpenFlow1.3環境を作ってみました..

オープンソースな最新OpenFlow実装ソフトウェアって、まだまだ少ないですね...



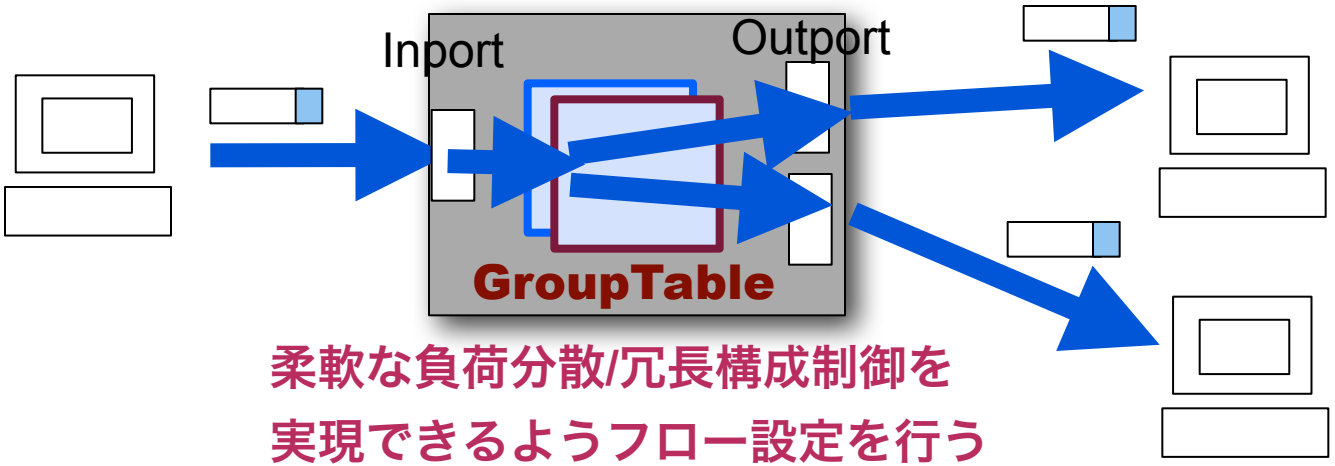
OpenFlowによる負荷分散/冗長構成

従来ベースでの負荷分散/冗長構成

ECMP,リンクアグリゲーションなど通信方式による冗長構成（負荷分散, ACT/SBY構成）の構築下において、 NW機器自らによるネットワーク稼働状況に応じた通信経路制御が実施されてきました。

OpenFlowベースでの負荷分散/冗長構成

OpenFlowでも、柔軟なトラフィックフロー制御によるマルチパス環境構築が可能です。OpenFlowコントローラが、プロアクティブに負荷分散/冗長構成のフロー定義をGroupTableに設定することにより、ネットワーク稼働状況に応じた通信経路制御が可能となります。

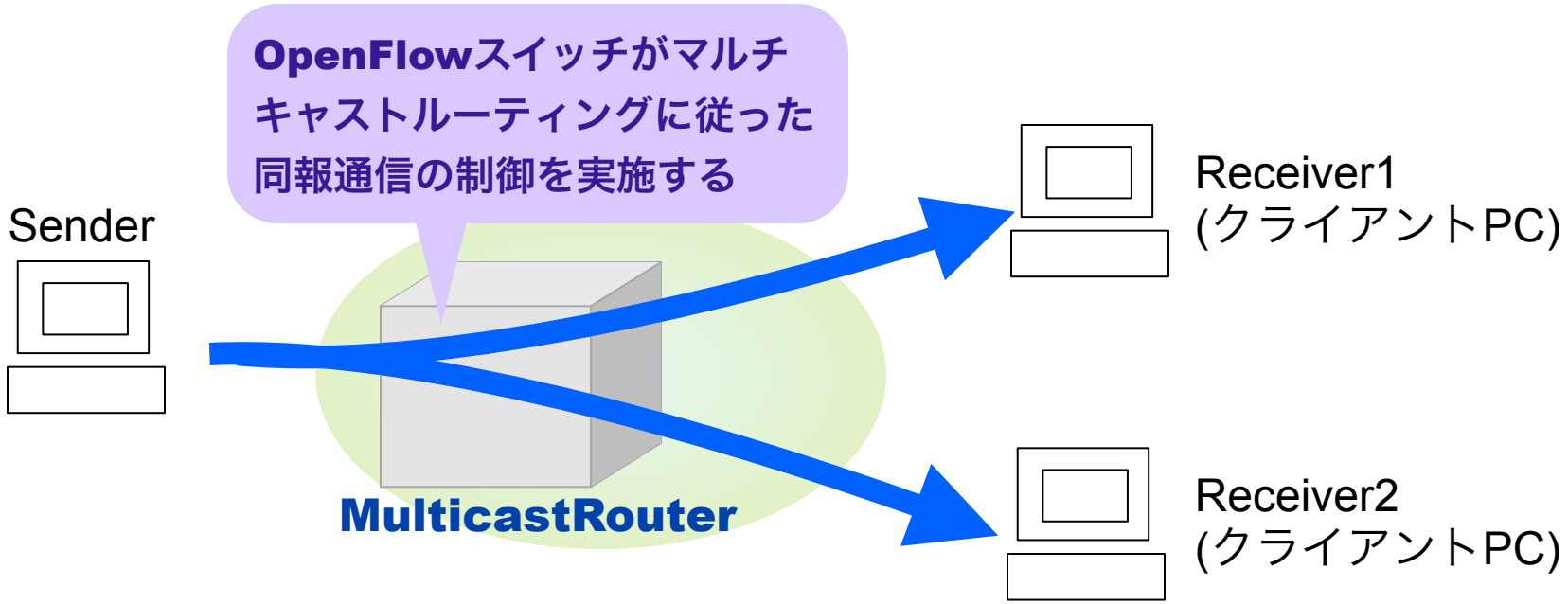


柔軟な負荷分散/冗長構成制御を
実現できるようフロー設定を行う

GroupTable活用事例1

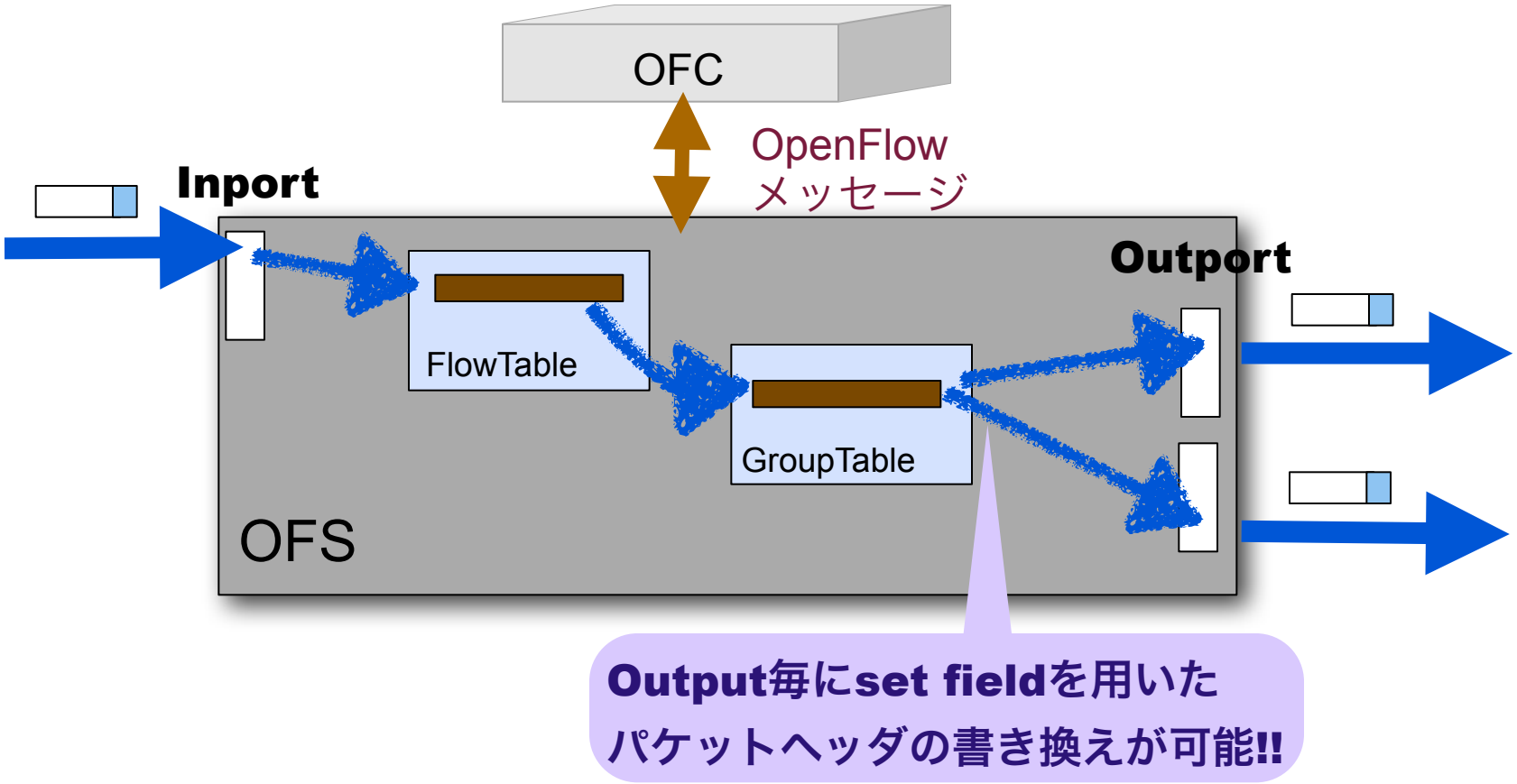
Multicastな同報通信

Sender(映像配信サーバ）よりReceiver(ライアントPC)へのMulticastルーティングを構築するモデル。同報通信なパケットコピー制御はOpenFlowのGroupタイプ(All)でトラフィクフローを制御する。



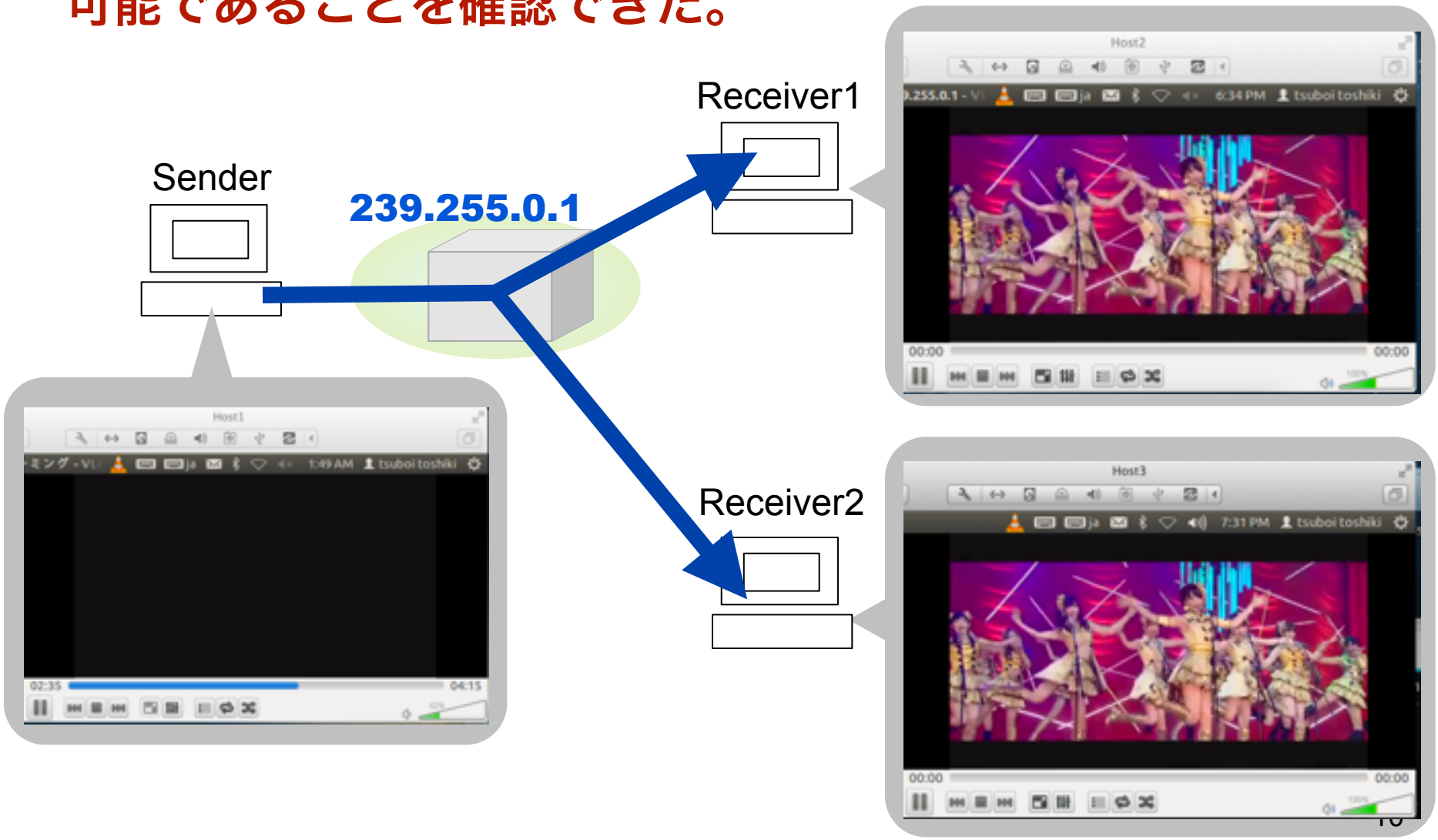
Groupタイプ(All)を用いたフロー制御

OFC側からプロアクティブなFlowTable/GroupTableを設定することにより、OFSのInportで受信した通信パケットを、複数Outportに同報送信するフロー制御を実現する仕組み



Groupタイプ(All)の動作確認

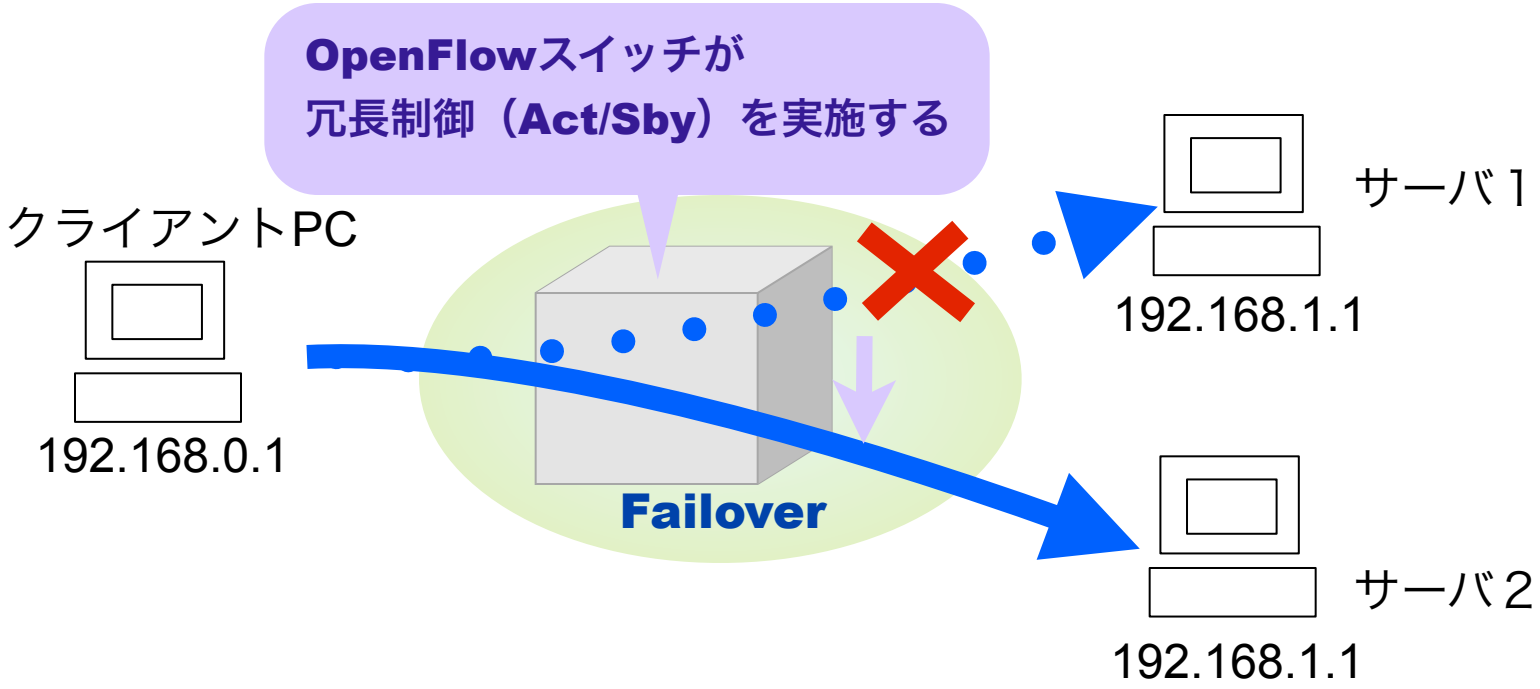
Groupタイプ(All)を活用することにより、Senderから2台のReceiverに対して、Multicastな同報通信が動作可能であることを確認できた。



GroupTable活用事例2

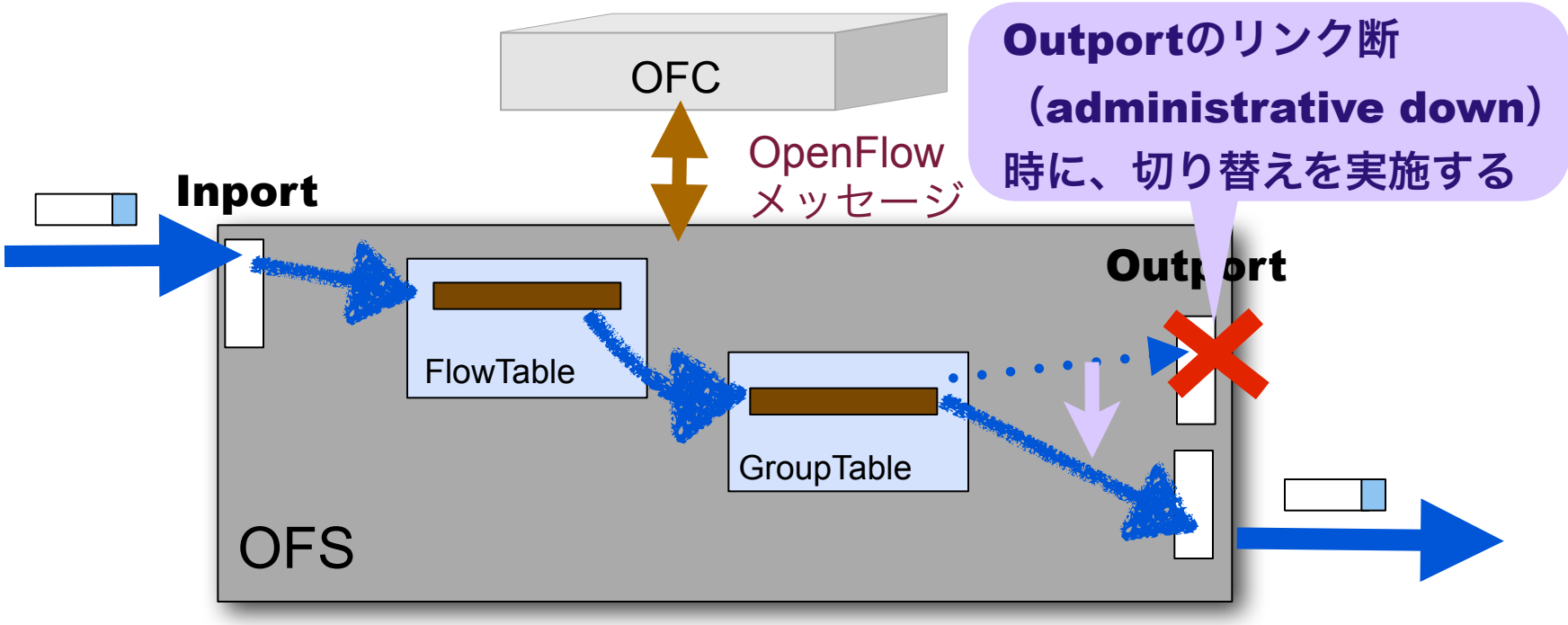
サーバ冗長構成（フェイルオーバ）

クライアントPCからサーバへのアクセス形態として、サーバ側でフェールオーバな冗長構成を構築するモデル。Act系からSby系への通信経路の振替え処理は、OpenFlowのGroupタイプ(FastFailover)でトラフィクフローを制御する。



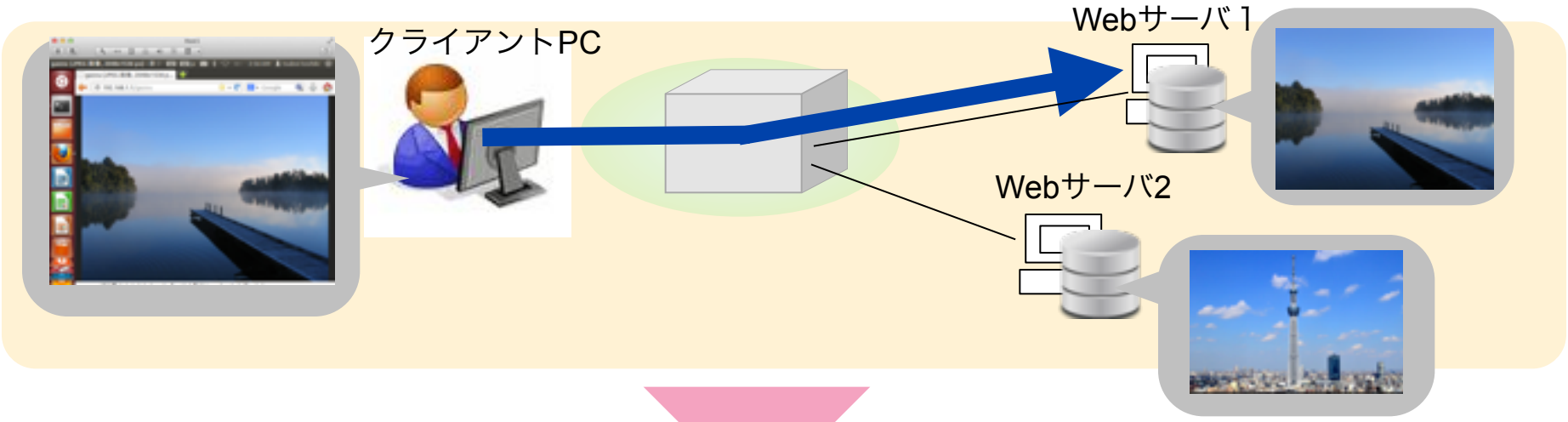
Groupタイプ(FastFailover)を用いたフロー制御

OFC側からプロアクティブなFlowTable/GroupTableを設定することにより、OFSのInportで受信した通信パケットを、動作可能なOutportに送信するフロー制御を実現する仕組み（OutPortが動作不可能と判断できるよう、故障検出もOFS側で行う）

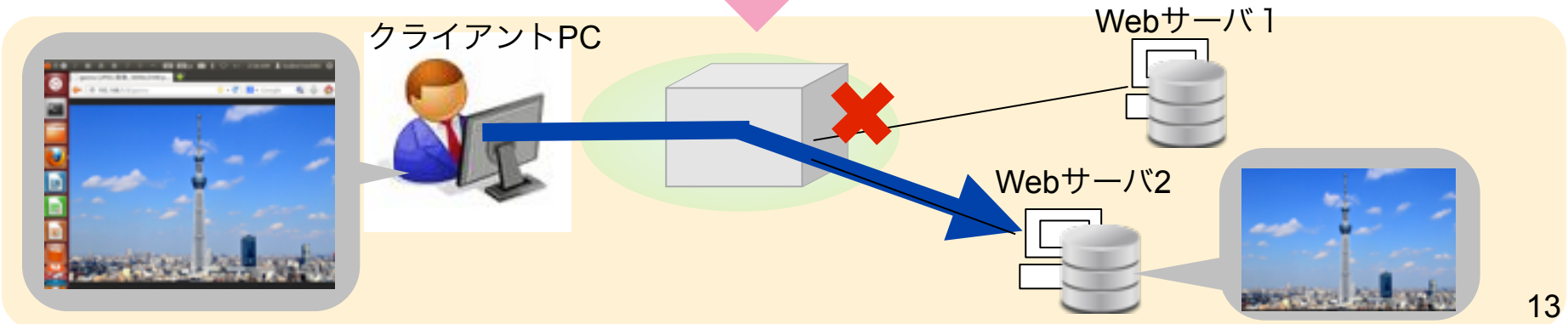


Groupタイプ(FastFailover)の動作確認

Webサーバ1向けアクセス通信経路にて故障が発生した場合には、接続先をWebサーバ1からWebサーバ2に振替えることにより、クライアント～Webサーバ間の通信処理が再開可能であることが確認できた。



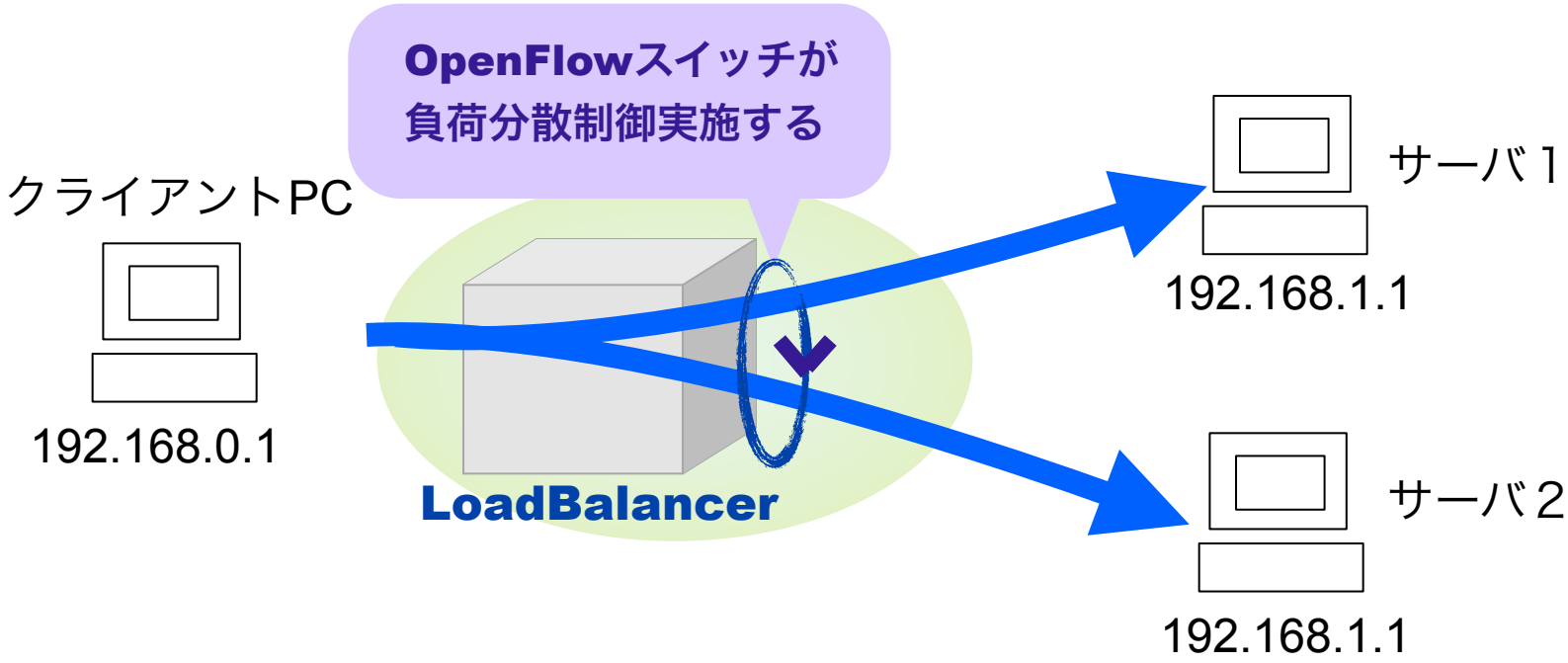
Webサーバ１向けアクセスの通信故障



GroupTable活用事例3

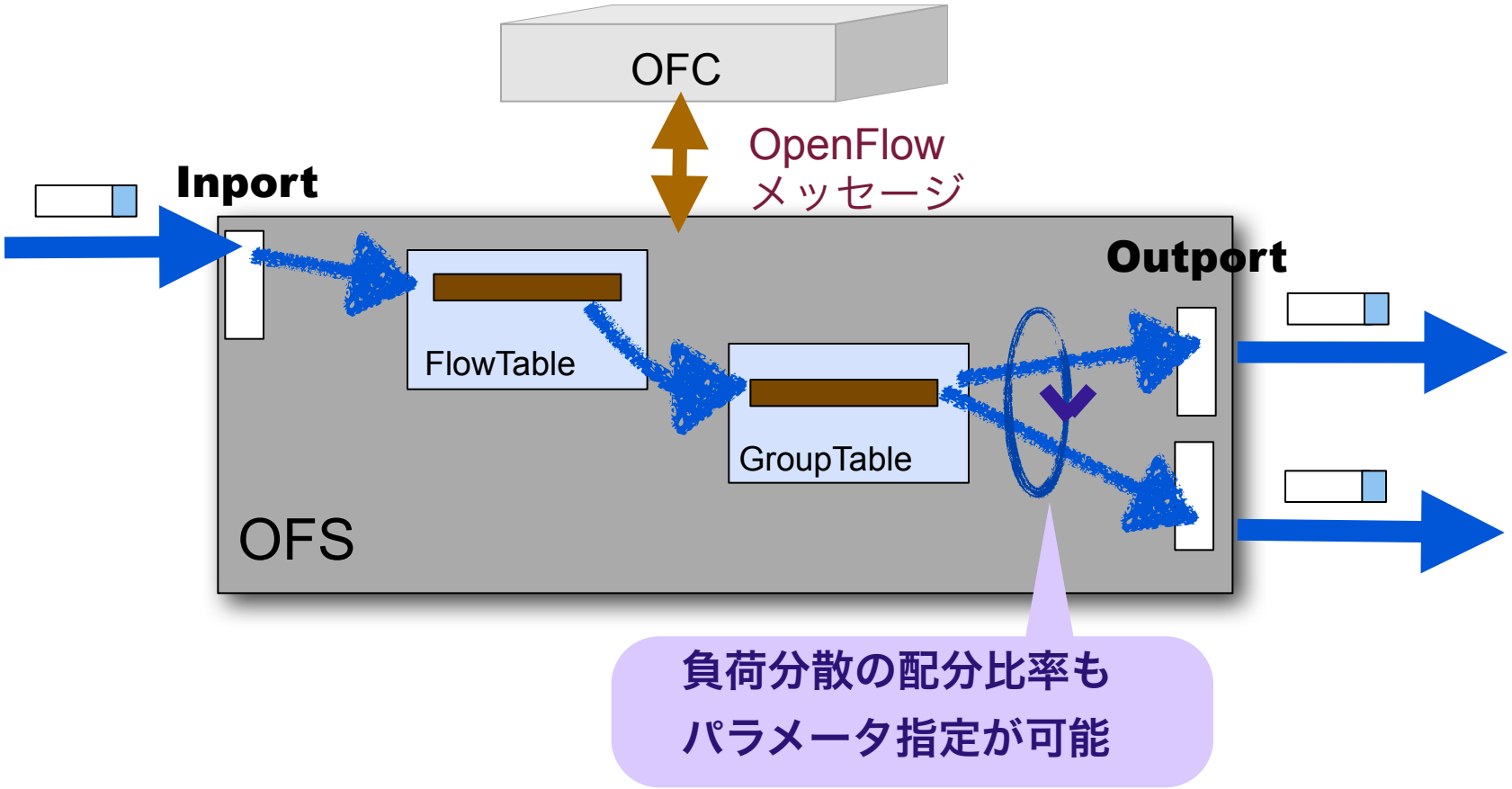
NWロードバランサーによるサーバ負荷分散

クライアントPCからサーバへのアクセス形態として、複数サーバ配備による負荷分散な環境を構築するモデル。サーバへの負荷分散なアクセス制御は、OpenFlowのGroupタイプ(Select)でトラフィクフローを制御する。



Groupタイプ(Select)を用いたフロー制御

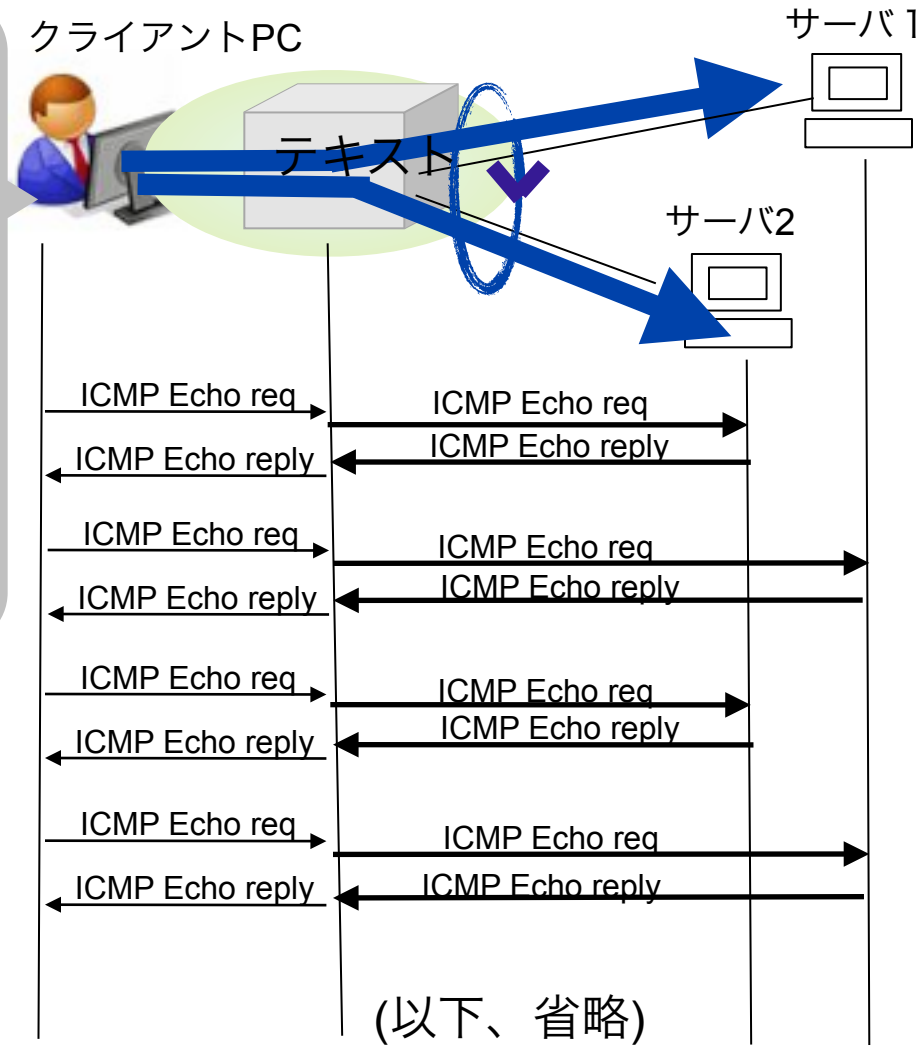
OFC側からプロアクティブなFlowTable/GroupTableを設定することにより、OFSのInportで受信した通信パケットを、複数Outportに負荷分散的に送信するフロー制御を実現する仕組み（負荷分散は、ラウンドロビン方式）



Groupタイプ(Select) の動作確認（ICMPの場合）

２台のサーバでICMPパケットを交互に受信できた。

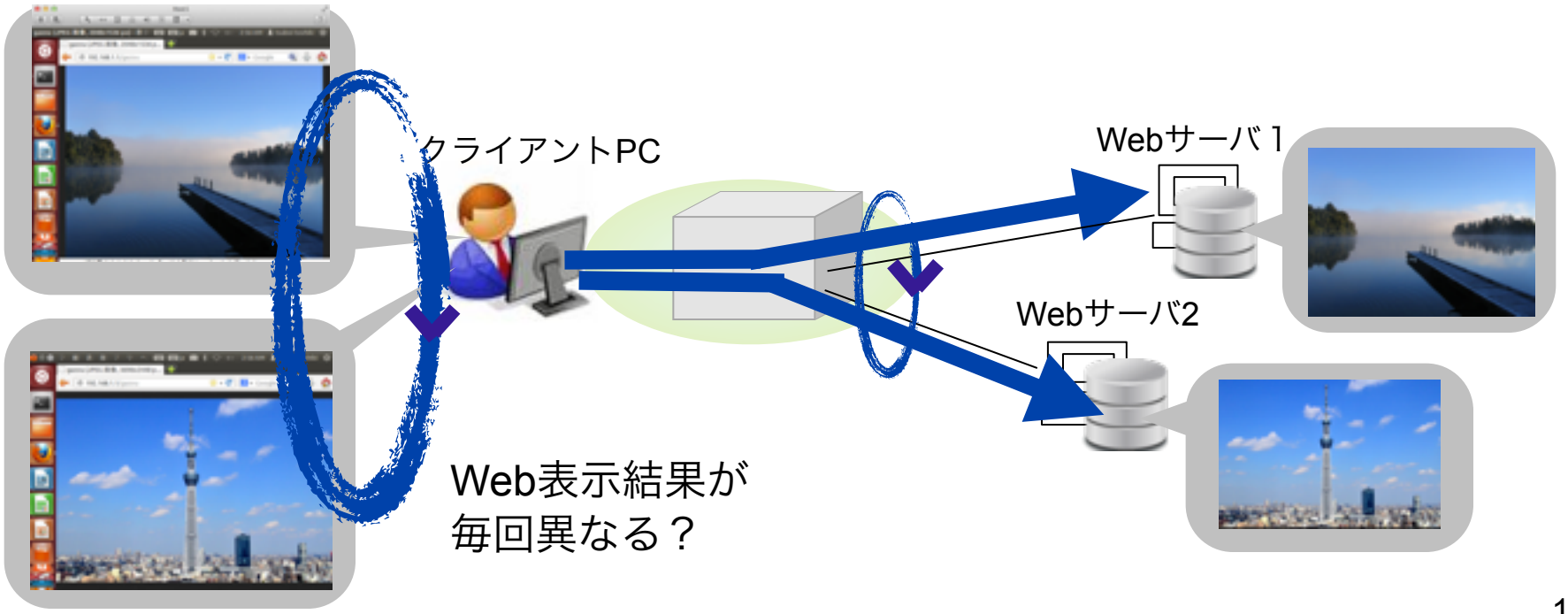
```
ttsubo@ubuntu:~$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_req=1 ttl=64 time=2.00 ms
64 bytes from 192.168.1.1: icmp_req=2 ttl=64 time=1.50 ms
64 bytes from 192.168.1.1: icmp_req=3 ttl=64 time=1.41 ms
64 bytes from 192.168.1.1: icmp_req=4 ttl=64 time=1.66 ms
64 bytes from 192.168.1.1: icmp_req=5 ttl=64 time=1.91 ms
64 bytes from 192.168.1.1: icmp_req=6 ttl=64 time=1.56 ms
64 bytes from 192.168.1.1: icmp_req=7 ttl=64 time=3.17 ms
64 bytes from 192.168.1.1: icmp_req=8 ttl=64 time=1.49 ms
64 bytes from 192.168.1.1: icmp_req=9 ttl=64 time=1.50 ms
64 bytes from 192.168.1.1: icmp_req=10 ttl=64 time=1.75 ms
64 bytes from 192.168.1.1: icmp_req=11 ttl=64 time=1.53 ms
64 bytes from 192.168.1.1: icmp_req=12 ttl=64 time=1.55 ms
64 bytes from 192.168.1.1: icmp_req=13 ttl=64 time=1.45 ms
64 bytes from 192.168.1.1: icmp_req=14 ttl=64 time=2.76 ms
64 bytes from 192.168.1.1: icmp_req=15 ttl=64 time=1.68 ms
```



Groupタイプ(Select) の動作確認（Webアクセスの場合）

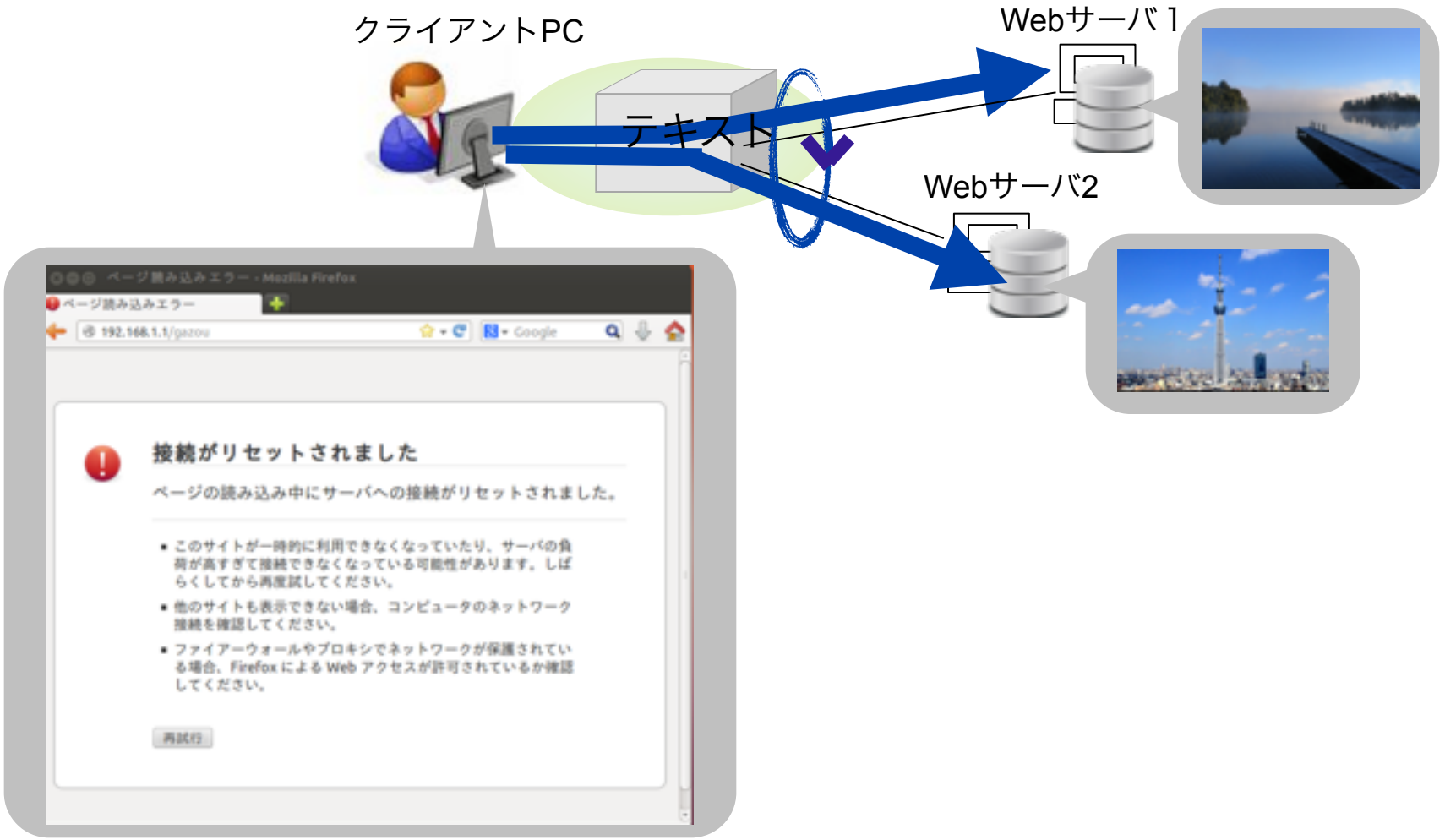
動作確認前の仮説

「OFS側では、クライアントPCからのWeb閲覧のアクセス毎に、サーバ負荷分散が図れるように接続先Webサーバが交互に切り替わってトラフィックフローを制御する」ような動作を想定しておりました。



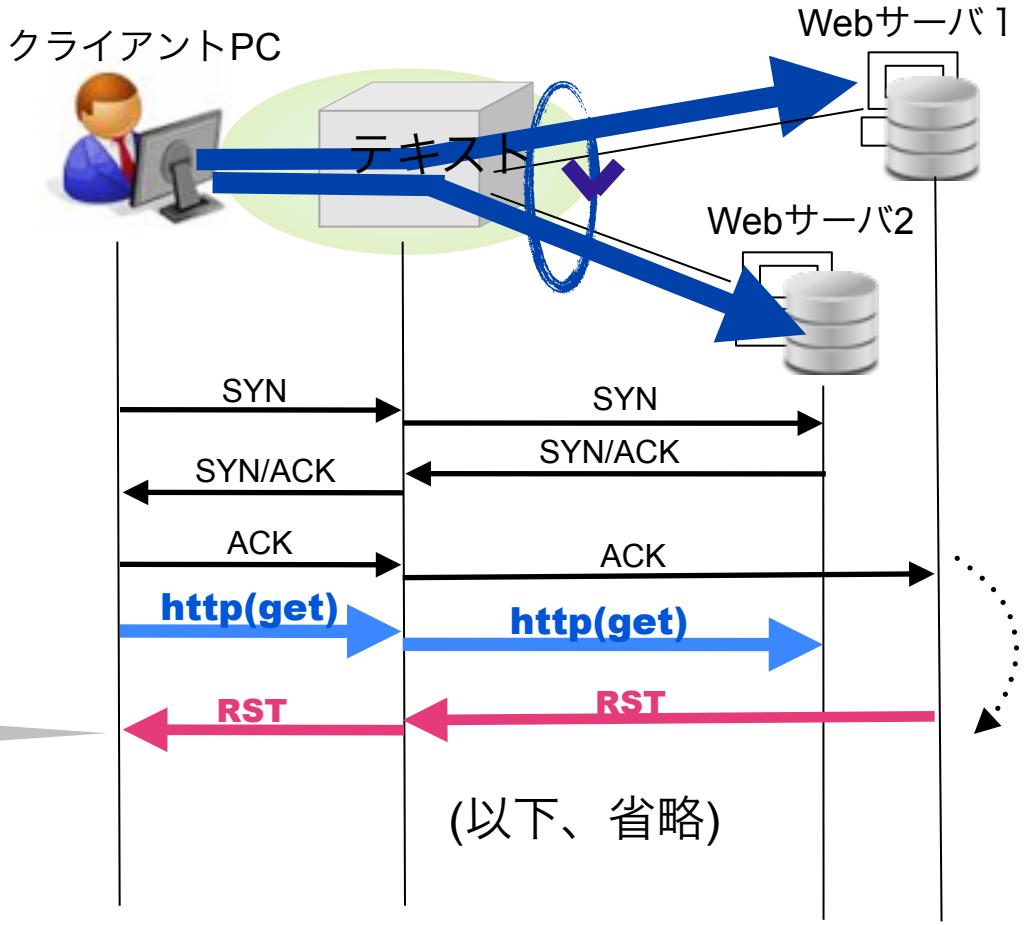
Groupタイプ(Select) の動作確認（Webアクセスの場合）

しかしながら、実際、クライアントPCからWeb閲覧を行うと、「接続がリセットされました」というエラーメッセージが表示されてしまいました。



「接続がリセットされました」の発生原因

OFS側では、 TCPセッション管理を想定せず、単純なラウンドロビンによる負荷分散を実施しているのみであるため、Web閲覧に関わるhttp通信シーケンスが、プロトコル違反と判定されてしまった…



OpenFlowの技術課題（私見）

OpenFlowといえば、ネットワーク挙動をプログラマブルに記述できるので、従来の通信技術では実現が難しかった柔軟なトラフィックフロー制御が可能となる技術と言われますよね。

さらに、最近のSDN/OpenFlow業界でのWeb/雑誌記事では、「ネットワーク仮想アプライアンス的な技術領域への適用」も期待される旨の文面をよく拝見します。

でも、現在のOpenFlow規格では、TCPセッション管理を踏まえたトラフィックフロー制御まで言及しておりません。

よって、NWロードバランサーやDPI(Deep Packet Inspection)のようなネットワーク仮想アプライアンス実現には、OpenFlowでは対応が難しいのではないでしょうか。

おわりに...

一般に、OpenFlow技術ネタを取り上げたセミナ講演資料/Web記事を最近よく見掛けるようになりました。でも、そこでの記載内容を鵜呑みにしてもよいのでしょうか？
(技術コンセプトと実装レベルに乖離があるかもしれません)

そこで、OpenFlow技術の目利き力を養うためにも、自分自身でOpenFlowを動かしてみると、いろいろな発見に遭遇できると思います.....