ネットワークコンピューティング
第8回

中山　雅哉 (m.nakayama@m.cnl.t.u-tokyo.ac.jp)
関谷　勇司 (sekiya@nc.u-tokyo.ac.jp)

本日のテーマ

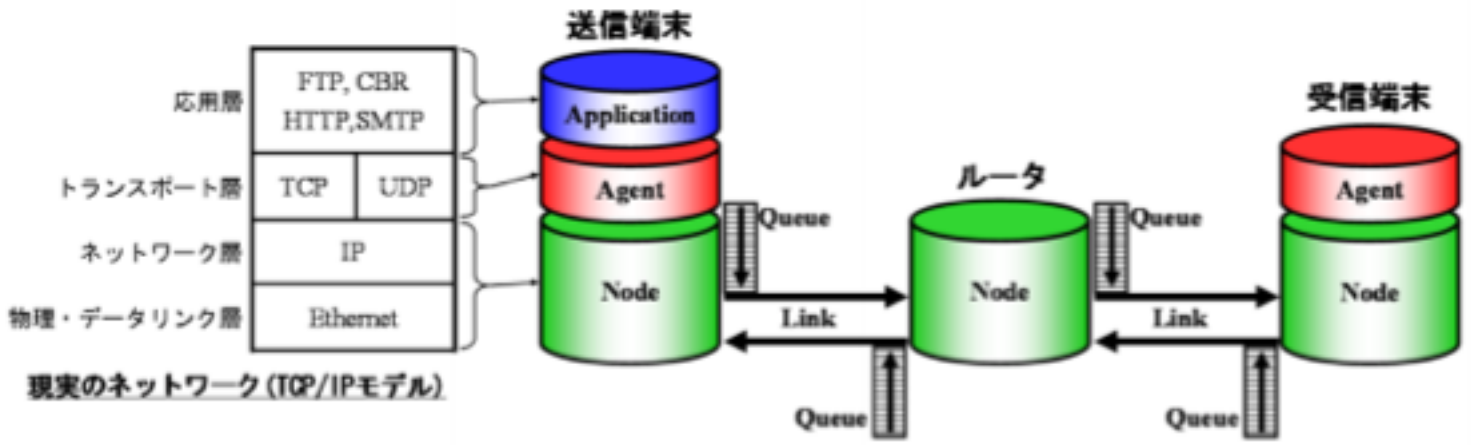
- ネットワークシミュレータ
- SDN
- SDN の必要性
- NFV

シミュレーション

- 現実の環境で行なうのが難しい場合に、現実の環境を模した場を作成し、実験をしてみること
- 例えば
  - Web サーバ 1台と Web のクライアントが 10,000台あり、同時にアクセスがあった場合の Web サーバの挙動
  - P2P ソフトウェアが同時に10,000台のマシン上で動いており、それらが連携してファイル交換する場合のあるファイルのダウンロード速度
- シミュレーションソフトウェア
  - ns2 / ns3
  - OPNET
  - mininet

ns2

- 非常に有名なネットワークシミュレータ
  - 多くの研究や論文がこのシミュレータを使って行われている
- 「シナリオファイル」と呼ばれるシミュレーション手順ファイルを作成する
  - 言語は TCL

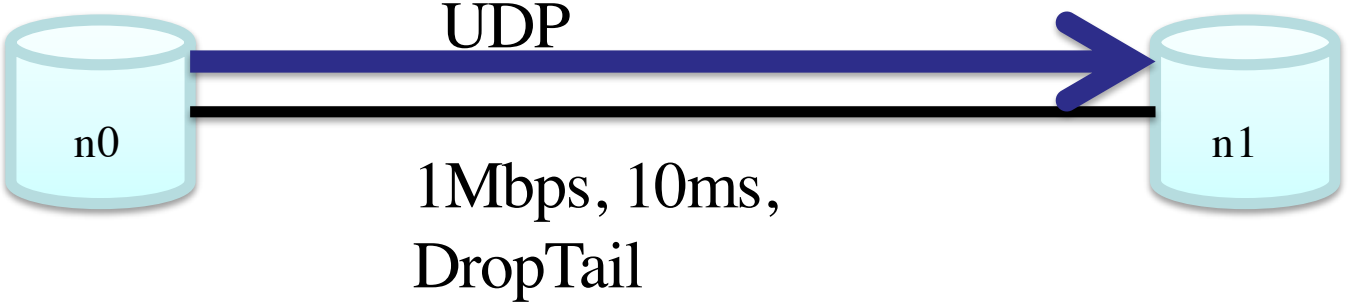


出展：http://www.it.ecei.tohoku.ac.jp/~bigtree/lecture/manual.pdf

ns2 の利用方法

- MacOS
  - MacOS の開発環境 (Xcode) を入れておけばそのまま動作可能
  - http://www.isi.edu/nsnam/ns/ から取得し、手順に従いインストール
- Windows
  - Cygwin の環境が必要 (TCL を動かすため)
  - Cygwin 環境が整えば、基本的なインストールは同様
- Linux
  - http://www.isi.edu/nsnam/ns/ から取得し、手順に従いインストール

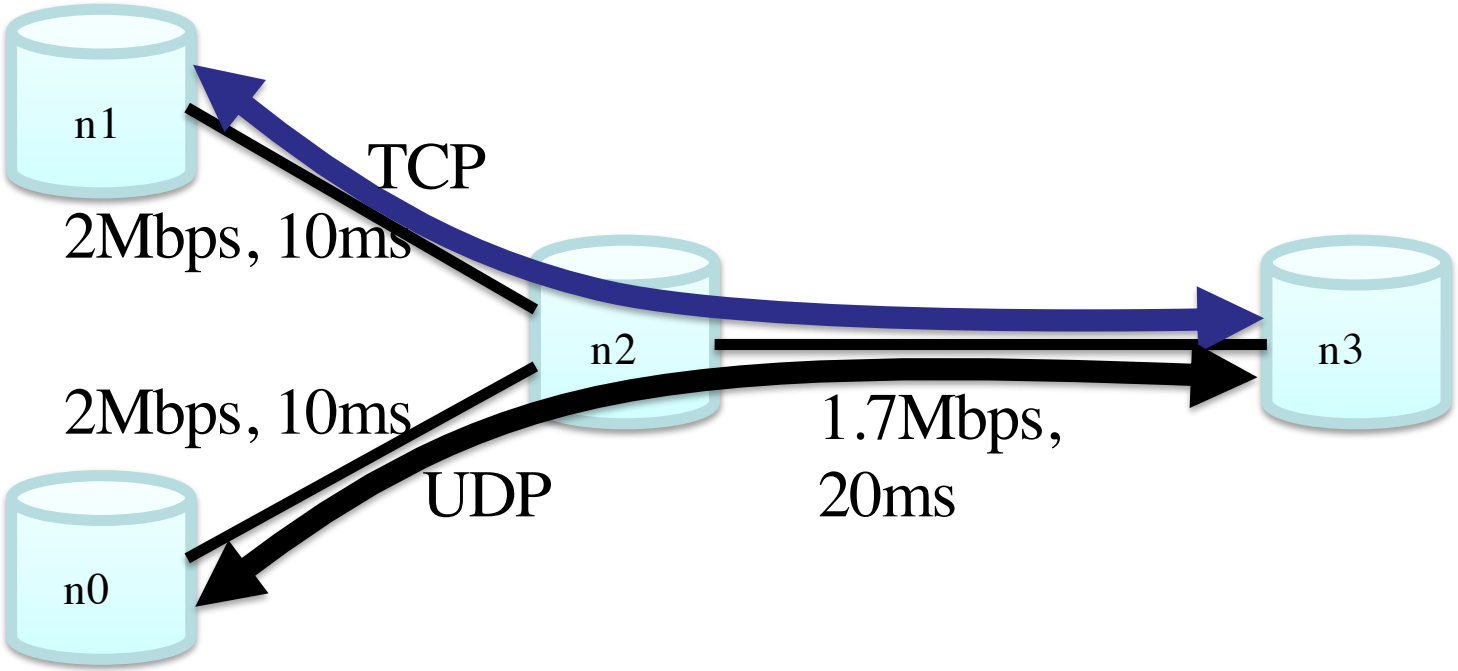
サンプルトポロジ (1)



シナリオファイル (1)

```
set ns [new Simulator]

set nf [open out.nam w]
$ns namtrace-all $nf

proc finish {} {
        global ns nf
        $ns flush-trace
        close $nf
        exec nam out.nam &
        exit 0
}

set n0 [$ns node]
set n1 [$ns node]

$ns duplex-link $n0 $n1 1Mb 10ms DropTail

set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0

set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

set null0 [new Agent/Null]
$ns attach-agent $n1 $null0

$ns connect $udp0 $null0

$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"

$ns at 5.0 "finish"
$ns run
```

サンプルトポロジ (2)



シナリオファイル (2)

```
#Create a simulator object
set ns [new Simulator]

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red

#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Open the Trace file
set tf [open out.tr w]
$ns trace-all $tf

#Define a 'finish' procedure
proc finish {} {
        global ns nf tf
        $ns flush-trace
        #Close the NAM trace file
        close $nf
        #Close the Trace file
        close $tf
        #Execute NAM on the trace file
        #exec nam out.nam &
        exit 0
}
```

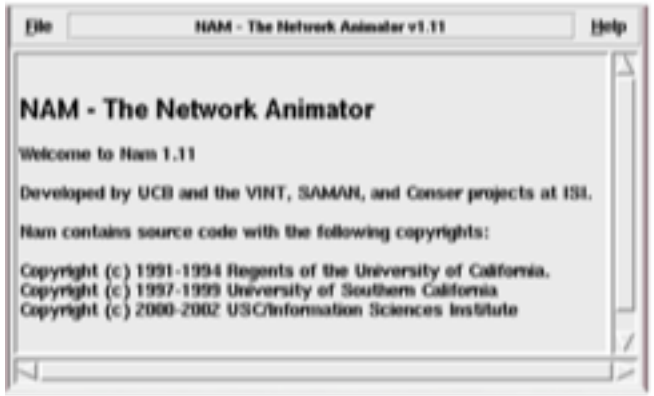
```
#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail

#Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10

#Give node position (NAM)
$ns duplex-link-op $n1 $n2 orient right-down
$ns duplex-link-op $n0 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

#Monitor the queue for link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5
```

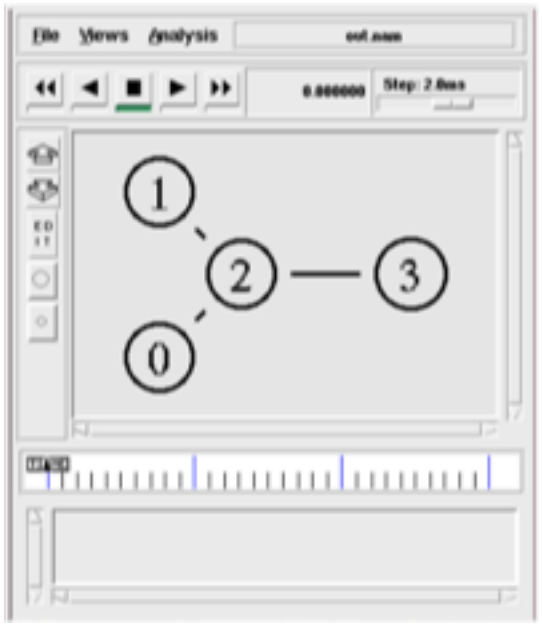
シナリオファイル (2) – cont.

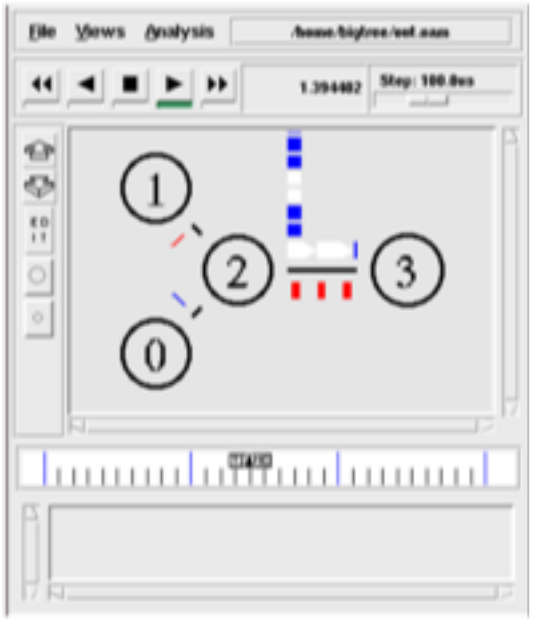
```
#Setup a TCP connection
set tcp [new Agent/TCP]
$ns attach-agent $n1 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

#Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP

#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n0 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null
$udp set fid_ 2
```

```
#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false

#Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"

#Call the finish procedure after 5 seconds of simulation
time
$ns at 5.0 "finish"

#Print CBR packet size and interval
puts "CBR packet size = [$cbr set packet_size_]"
puts "CBR interval = [$cbr set interval_]"

#Run the simulation
$ns run
```

実行画面例


File
NAM - The Network Animator v1.11
Help
NAM - The Network Animator
Welcome to Nam 1.11
Developed by UCB and the VINT, SAMAN, and Conser projects at ISI.
Nam contains source code with the following copyrights:
Copyright (c) 1991-1994 Regents of the University of California.
Copyright (c) 1997-1999 University of Southern California
Copyright (c) 2000-2002 USC/Information Sciences Institute


File
Views
Analysis
Step: 2.0ms
1
2
3
0


File
Views
Analysis
Step: 100.0ms
1
2
3
0

実行結果例（トレースファイル）

```
event time  snode tnode pkttype  pktsize      flag      fid  saddr  daddr  seqn pkid

+ 0.100 1 2 cbr 1000 ------- 2 1.0 3.1 0 0
- 0.100 1 2 cbr 1000 ------- 2 1.0 3.1 0 0
+ 0.108 1 2 cbr 1000 ------- 2 1.0 3.1 1 1
- 0.108 1 2 cbr 1000 ------- 2 1.0 3.1 1 1
r 0.114 1 2 cbr 1000 ------- 2 1.0 3.1 0 0
+ 0.114 2 3 cbr 1000 ------- 2 1.0 3.1 0 0
- 0.114 2 3 cbr 1000 ------- 2 1.0 3.1 0 0
+ 0.116 1 2 cbr 1000 ------- 2 1.0 3.1 2 2
```

結果の読み方

- イベントの種類
  - キューに入る(+)
  - キューから出る (–)
  - 受信 (r)
  - 廃棄 (d)
- イベント発生時間
- イベント発生場所 (始点ノード番号、終点ノード番号)
- パケット種類
- パケットサイズ
- パケットフラグ

SDN とは

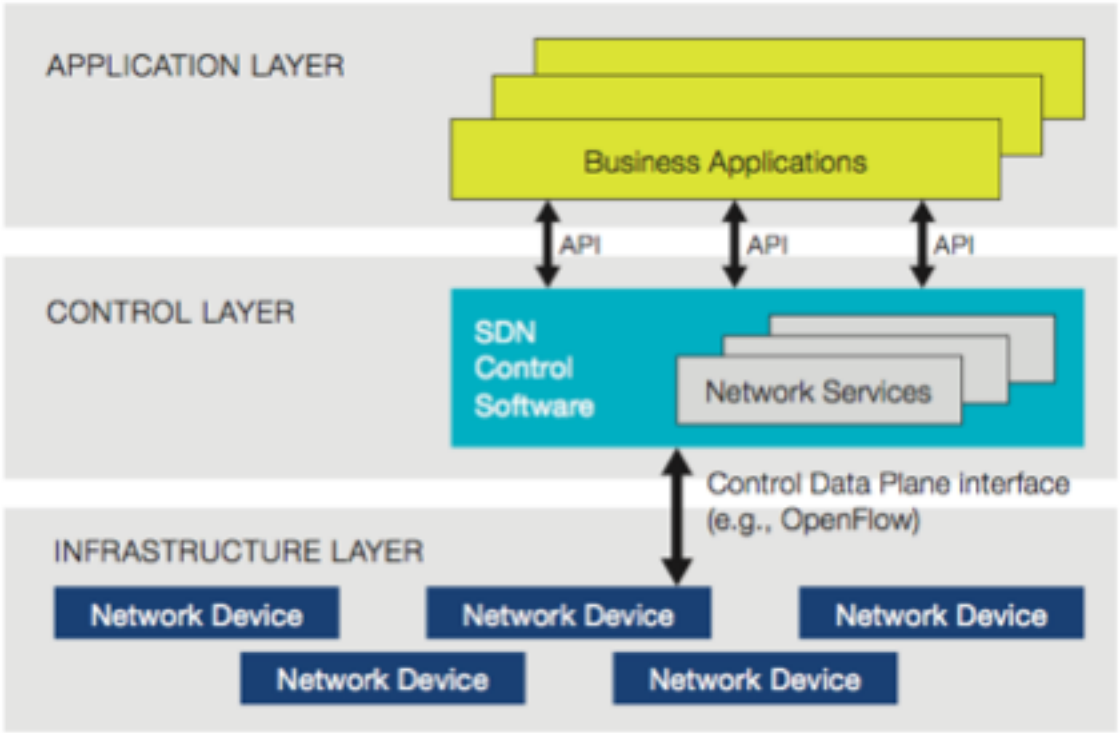
SDN と OpenFlow

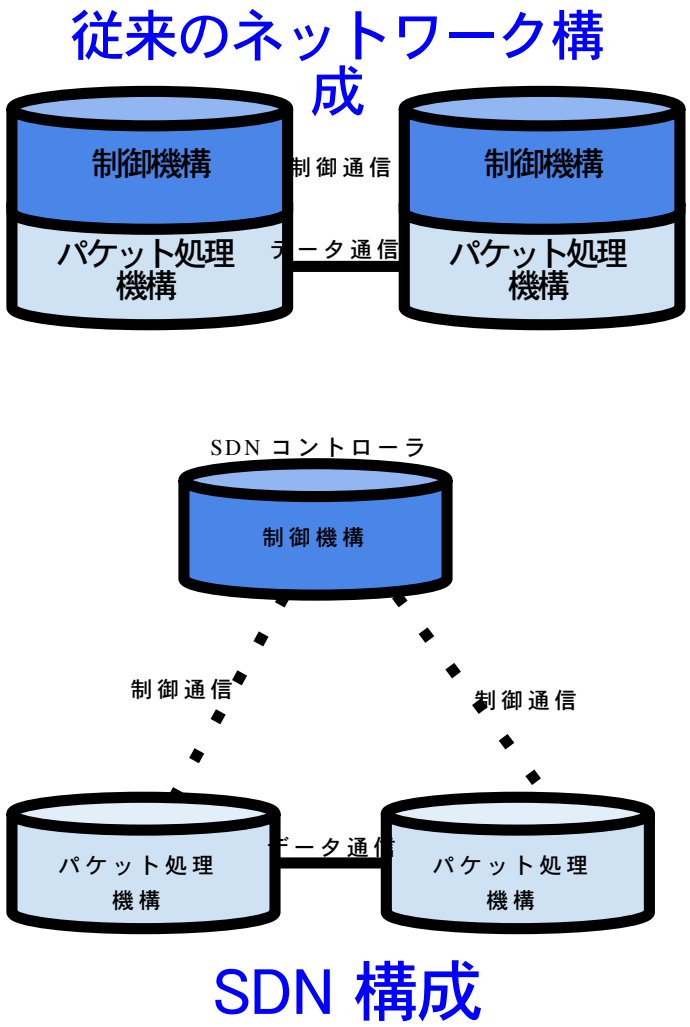
- SDN (Software Defined Networking)
  - ソフトウェアにて構成できるネットワーク
  - 代表的なものとして「OpenFlow」
- OpenFlow
  - Stanford 大学によって考案され、開発された
  - 現在は ONF (Open Networking Foundation) により標準化
  - 「ソフトウェア」によってネットワークを自由に変更する
- よく言われる定義
  - コントロールプレーンとデータプレーンの分離
  - 柔軟なネットワーク

SDN の定義 by ONF

- 複数ベンダーのネットワーク機器を一元管理
- 共通の API を用いて、オーケストレーションに大して下位層のネットワークの詳細を抽象化し、優れた自動化と管理を実現
- 最新のネットワーク昨日やサービスを導入し、迅速にイノベーションを実現
- 共通のプログラミング環境を使うことで運用者・企業・ソフトウェアベンダー・ユーザもプログラミングが可能に
- 一元化されたネットワークのステート情報をアプリケーションが活用することで、ユーザのニーズにシームレスに対応

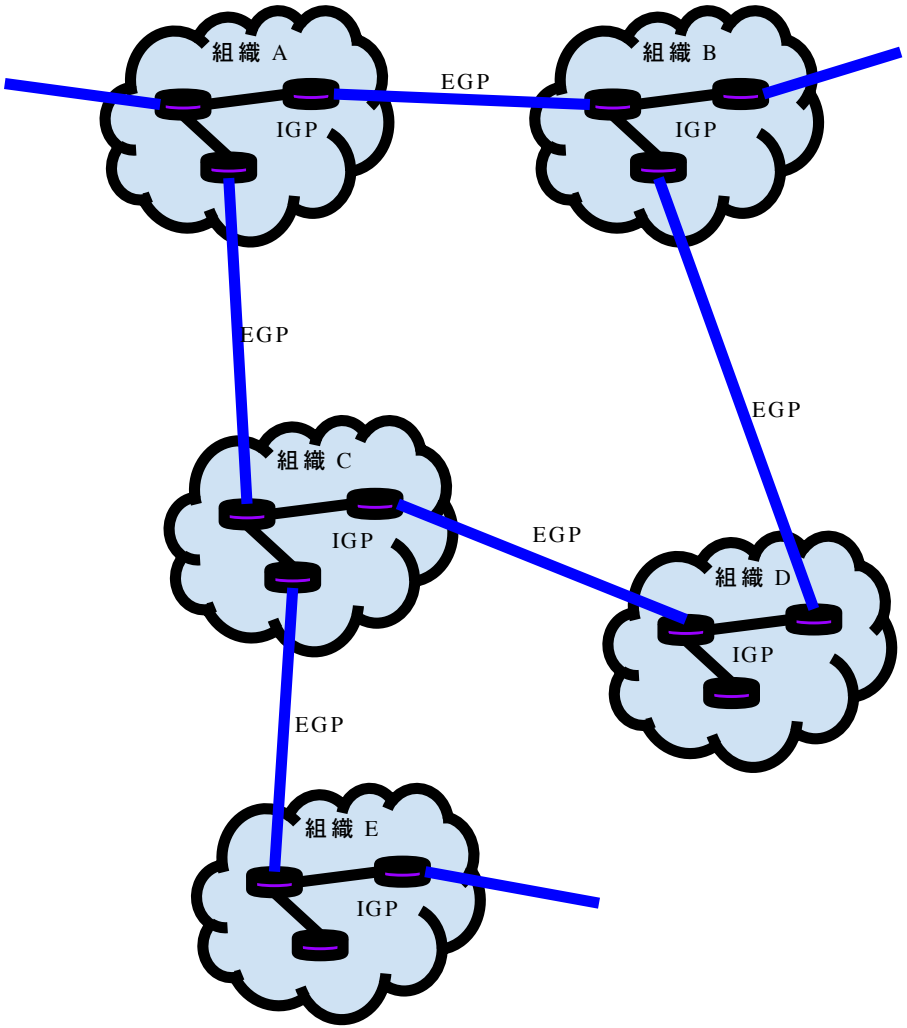
結局 SDN とは何なのか





従来のネットワーク（1）

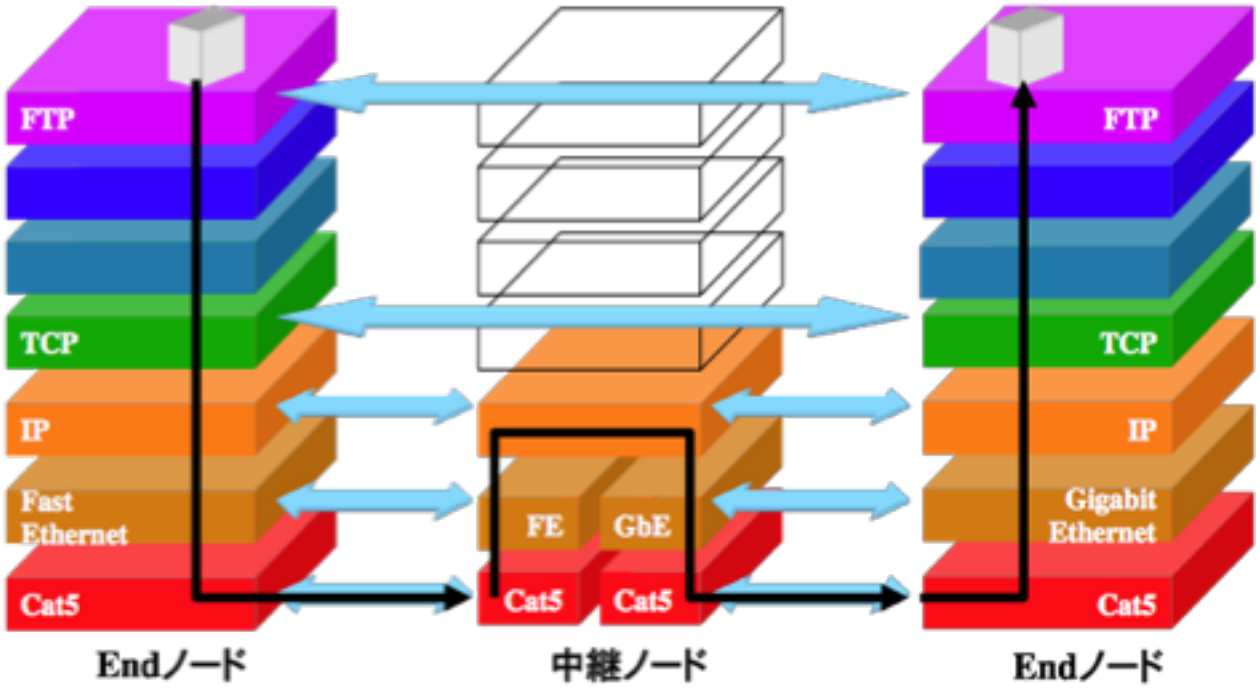
- 自律分散型のネットワーク
  - The Internet
  - 複数の管理ドメインが協調して機能する
- 挙動は自分で決める
  - 隣接組織とは情報を交換する
  - ネットワーク機器それぞれが、ルールに従い機能する
- 全体としてネットワークが成立



従来のネットワーク (2)

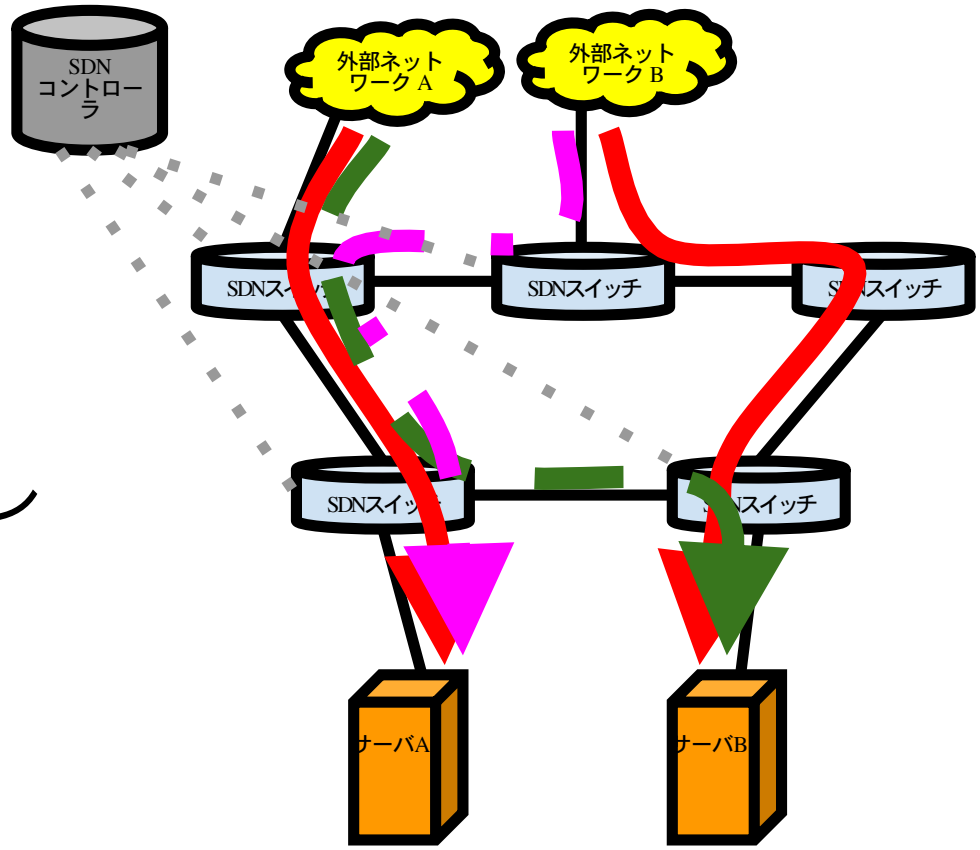
- IP アドレスがすべての識別子
  - OSI 参照モデル
- IP アドレスに基づいた経路制御

- IP アドレスが
  - locator
  - Identifier



OpenFlow では？

- すべてがコントローラ制御
  - 「集中管理」モデル
  - 設計者・管理者がすべてを見通し、制御する
- いわば
  「セルフサービス」
  ネットワーク
- 「パケットを隣に渡す」というレベルから制御が必要



OpenFlow の仕様

- OpenFlow 1.0 と 1.3 が存在
- 基本的には「rule」と「action」によって定義される
- rule の例
  1. dst mac = b8:f6:b1:14:bd:35 and VLAN ID = 104
  2. physical ingress port = 0/24
- action の例
  1. Push-Tag : VLAN ID = 200
  2. Drop
  3. Output : physical port = 0/48
  4. Output : Controller

既存の L2 スイッチの挙動を OpenFlow で記述すると？

- パケットを受信する
  - 受信したパケットの送信元 MAC アドレスを調査
    - パケットが入ってきた物理 or 論理ポートと MAC アドレスを記録
  - 受信したパケットの宛先 MAC アドレスを調査
    - 宛先となる MAC アドレスが記録されているか？
    - 記録されていない、もしくは特殊な MAC アドレスの場合
  - 宛先に向けてパケットを送出
- VLAN ID が付加されていた場合は？
  - MAC アドレスと VLAN ID を対にして管理
  - VLAN ID 毎に記録するテーブルが異なる
- 記録した MAC アドレスの保持時間

既存の L3 ルータの挙動を OpenFlow で記述すると？

- パケットを受信
  - 自身宛 MAC アドレスのパケットであるか確認
    - 自身宛であれば宛先 IP アドレスを経路表から検索
    - 宛先アドレスが経路表に存在する？
  - 経路表にしたがってパケットを送出
    - 送出する対向側インタフェースの MAC アドレスは記録サれている？
      - arp / ndp パケットを送出し対向インタフェースの MAC アドレスを調査
    - 宛先 MAC アドレスは対向のルータの MAC アドレスとしパケットを送出
- フィルタリングルールの存在？

既存のルールにとらわれない柔軟な動作

- SDN (OpenFlow) ならばこれらの「ルール」を変更できる
  - 宛先ポート番号「だけ」を見てパケット送出先を決定
  - 送信元 IP アドレスをみてパケット送出先を決定
  - パケットの「中身」を見て送出先を決定
- 全てを自分で定義しなければならない
  - コントローラにて「プログラム」
  - そのプログラムに従い、パケットは制御される

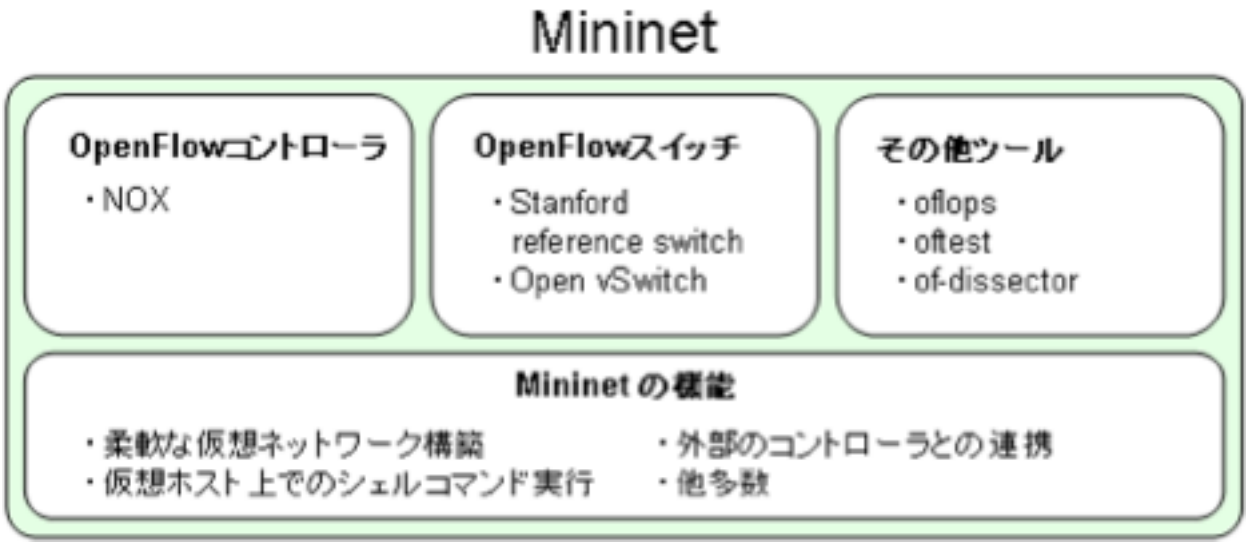
OpenFlow のシミュレーション

- Open vSwitch は OpenFlow をサポート
  - mininet であれば、スイッチを OpenFlow スイッチとしたシミュレーションも可能
  - OpenFlow コントローラと接続し、ルールを適用する
- 詳しく知りたい方は
  - http://www.hogetan.net/note/network/openflow_tutorial.html
  - http://tech-sketch.jp/2012/07/openflowmininet.html
  - http://archive.openflow.org/wk/index.php/OpenFlow_Tutorial

  等多くの情報が存在します

mininet

- 最近多く利用されるシミュレーション環境構築ソフトウェア
  - スタンフォード大学にて開発



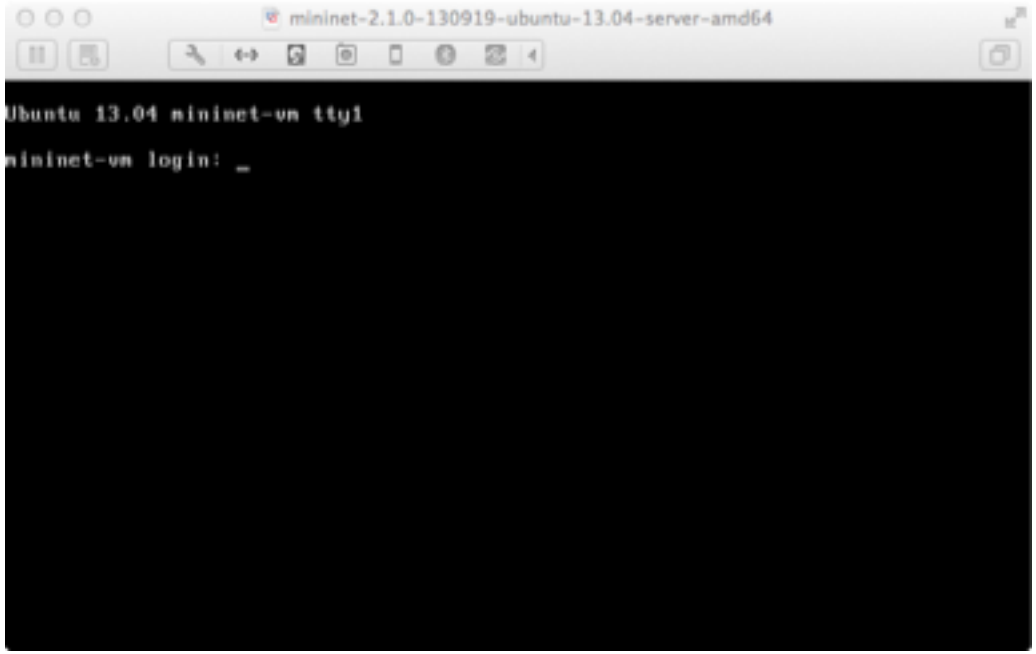
- OpenFlow のシミュレーションに利用されることが多い
  - OpenFlow 以外のシミュレーションも可能

mininet とは

- Linux 上でのみ動作するシミュレータ
  - Linux 特有の機能を使っているため
    - Linux Container
    - Open vSwitch
- MacOS や Windows で利用するためには
  - Vmware や VirtualBox などの仮想マシン (VM) 技術を利用できるソフトウェアを使う
- 仮想マシン (VM) とは？
  - 擬似的に作成されたコンピュータ
  - 実在する機器ではなく、ソフトウェア的に実現されたコンピュータ

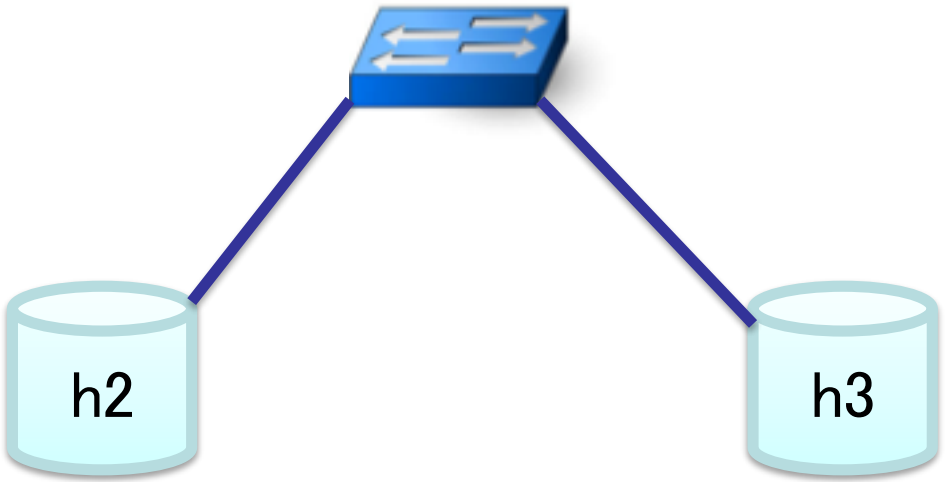
mininet のインストール

- ソースからビルド (Linux 上で)
- OVA 形式の VM イメージをインストール
  - http://yuba.stanford.edu/foswiki/bin/view/OpenFlow/Mininet
  - mininet がインストールされた Linux がまるごと入る



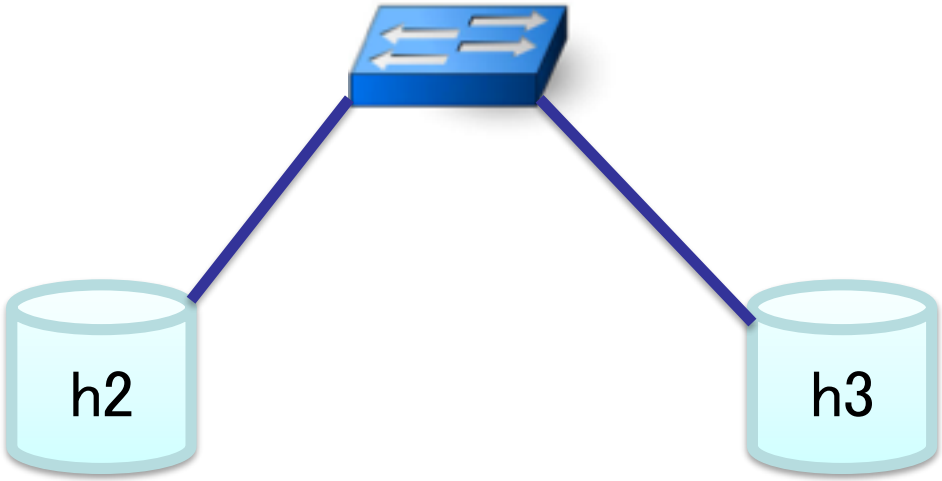
mininet の実行 (1)

```
$ sudo mn
*** Loading openvswitch_mod
*** Adding controller
*** Creating network
*** Adding hosts:
h2 h3
*** Adding switches:
s1
*** Adding links:
(s1, h2) (s1, h3)
*** Configuring hosts
h2 h3
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
```



mininet の実行 (2)

```
mininet> pingall
*** Ping: testing ping reachability
h2 -> h3
h3 -> h2
*** Results: 0% dropped (0/2 lost)
```




```
mininet> h2 python -m SimpleHTTPServer &
```

```
mininet> h3 wget -O - http://10.0.0.2:8000/
--2014-07-02 10:15:22--  http://10.0.0.2:8000/
Connecting to 10.0.0.2:8000... connected.
HTTP request sent, awaiting response... 200 OK
```

mininet でできること

- Linux でできることはほぼ全てできる
  - Linux を L3 ルータにしたり
  - 経路制御プロトコルを動作させたり
- スイッチは Open vSwitch というソフトウェアスイッチが用いられている
  - Open vSwitch でできることは全てできる
  - OpenFlow も可能
- 実験用のネットワークトポロジは Python にて記入

```
leftHost = 1
leftSwitch = 2
rightSwitch = 3
rightHost = 4

self.add_node( leftSwitch, Node( is_switch=True ) )
self.add_node( rightSwitch, Node( is_switch=True ) )
self.add_node( leftHost, Node( is_switch=False ) )
self.add_node( rightHost, Node( is_switch=False ) )

self.add_edge( leftHost, leftSwitch )
self.add_edge( leftSwitch, rightSwitch )
self.add_edge( rightSwitch, rightHost )

self.enable_all()
```

なぜ SDN が必要とされるのか

通信方向の変化と SDN の登場

- クラウド
  - 計算処理の分散化
  - サービス提供のソフトウェア化
  - **仮想化の進化と普及**

- ビッグデータ
  - データの有機結合
  - データセンター内**折り返し通信の増大**

従来の通信フロー・制御手法とは異なった通信が生まれる

ネットワークのコスト

"The cost of networking is escalating relative to the cost of all other equipment. It is Anti-Moore. All of our gear is going down in cost, and we are dropping prices, and networking is going the wrong way. That is a super-big problem, and I like to look out a few years, and I am seeing that the size of the networking problem is getting worse constantly. At the same time that networking is going Anti-Moore, the ratio of networking to compute is going up."

http://www.enterprisetech.com/2014/11/14/rare-peek-massive-scale-aws/

何が増えているのか？

- HTTP トラフィック
  - 特定サイトへのトラフィック集中
  - 動画サイト
- Cisco VNI の予想
  - 全世界のモバイルトラフィックは 2012 -> 2017 で 13倍に
  - 4G スマートフォンの普及
- sandvine による調査

https://www.sandvine.com/downloads/general/global-internet-phenomena/2014/2h-2014-global-internet-phenomena-report.pdf

	Upstream		Downstream		Aggregate	
Rank	Application	Share	Application	Share	Application	Share
1	BitTorrent	25.49%	Netflix	34.89%	Netflix	32.39%
2	Netflix	9.48%	YouTube	14.04%	YouTube	13.25%
3	HTTP	7.18%	HTTP	8.62%	HTTP	8.47%
4	SSL	7.05%	Facebook	2.98%	BitTorrent	5.03%
5	YouTube	6.14%	BitTorrent	2.80%	Facebook	2.94%
6	iCloud	4.41%	iTunes	2.77%	SSL	2.63%
7	Skype	2.77%	MPEG - OTHER	2.66%	iTunes	2.55%
8	Facebook	2.60%	Amazon Video	2.58%	MPEG - OTHER	2.44%
9	FaceTime	2.38%	SSL	2.14%	Amazon Video	2.37%
10	Dropbox	1.48%	Hulu	1.41%	Hulu	1.20%
		68.98%		74.89%		73.28%

sandvine

Table 2 - Top 10 Peak Period Applications - North America, Fixed Access

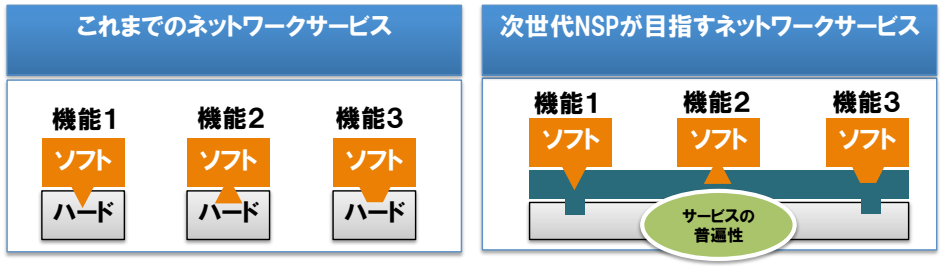
	Upstream		Downstream		Aggregate	
Rank	Application	Share	Application	Share	Application	Share
1	Facebook	22.36%	YouTube	19.75%	Facebook	19.43%
2	Google Cloud	11.97%	Facebook	19.05%	YouTube	18.02%
3	HTTP	9.85%	HTTP	11.44%	HTTP	11.26%
4	SSL	9.22%	MPEG - OTHER	6.32%	MPEG - OTHER	5.72%
5	YouTube	4.56%	Netflix	4.51%	SSL	4.63%
6	Instagram	2.55%	Instagram	4.49%	Instagram	4.27%
7	Snapchat	1.94%	SSL	4.03%	Netflix	4.10%
8	BitTorrent	1.88%	iTunes	3.20%	Google Cloud	4.09%
9	FaceTime	1.59%	Google Cloud	3.07%	iTunes	2.96%
10	Skype	1.53%	Pandora Radio	2.72%	Pandora Radio	2.53%
		67.44%		78.57%		77.02%

sandvine

Table 4 - Top 10 Peak Period Applications - North America, Mobile Access

「次」のネットワークインフラに求められるもの

- 動的な構成変更を可能とするネットワークサービスプラットフォーム
  - VNF + サービスチェイニングによる動的構成
  - SDN による可搬性
- 「位置」の呪縛から解き放たれる → 普遍性

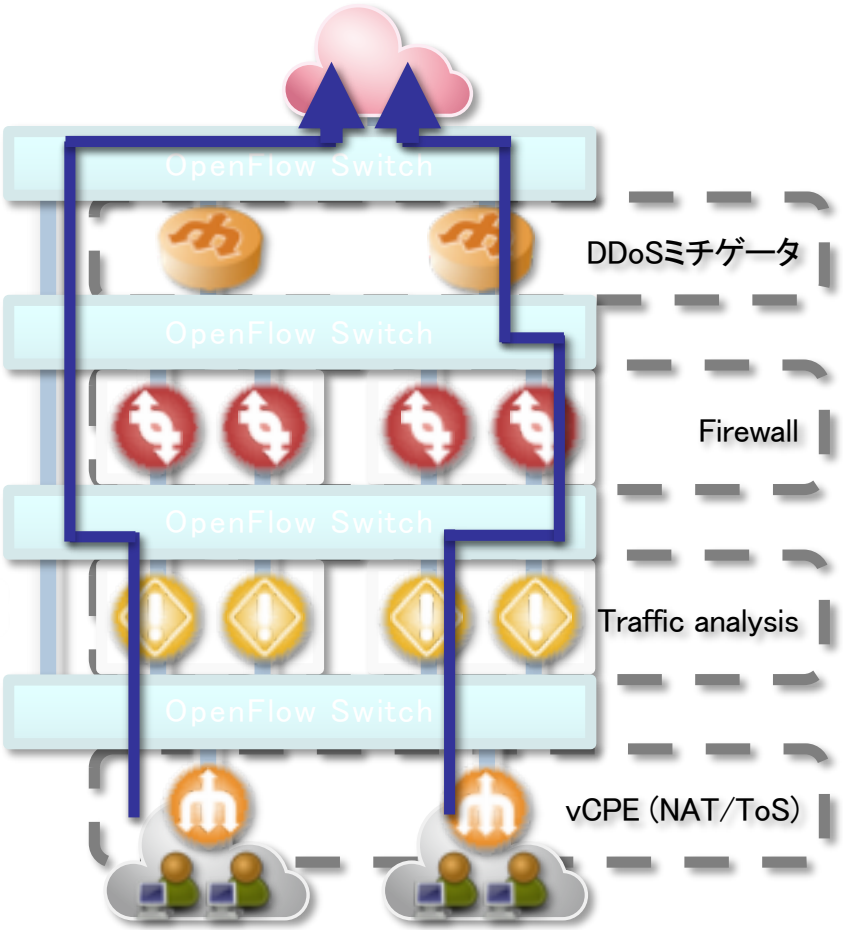


- 専用機器
- 長納期
- 位置の呪縛

- 汎用機器
- 短納期
- 柔軟な規模性
- 必要な場所で必要なサービス

SDN/NFV を利用したアーキテクチャ事例

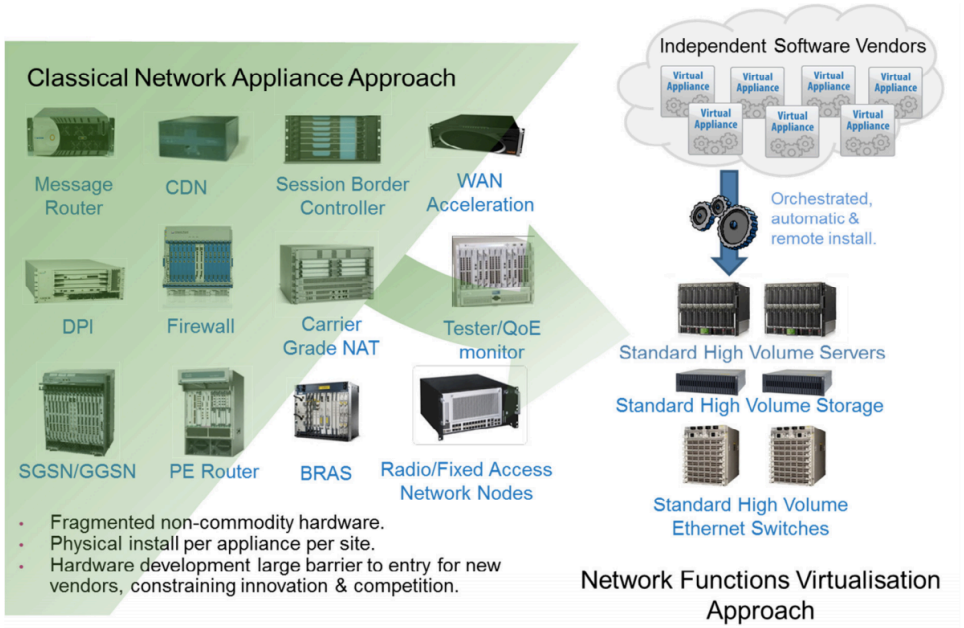
- ユーザごとにサービスチェインを選択
- vCPEでパケットにマーキング
  - 各サービスをToSの各bitに埋め込む
- OpenFlowでVNFへ入れるか判断
  - ToSの特定bitをチェック
  - 1ならVNFへ、0なら迂回
- 1つの機能(VNF)を複数のVAで構築
  - OpenFlowでトラフィックを複数VAへ分散
  - 送信元アドレスをハッシュしてパケットをいれるVAを決定



NFV とは

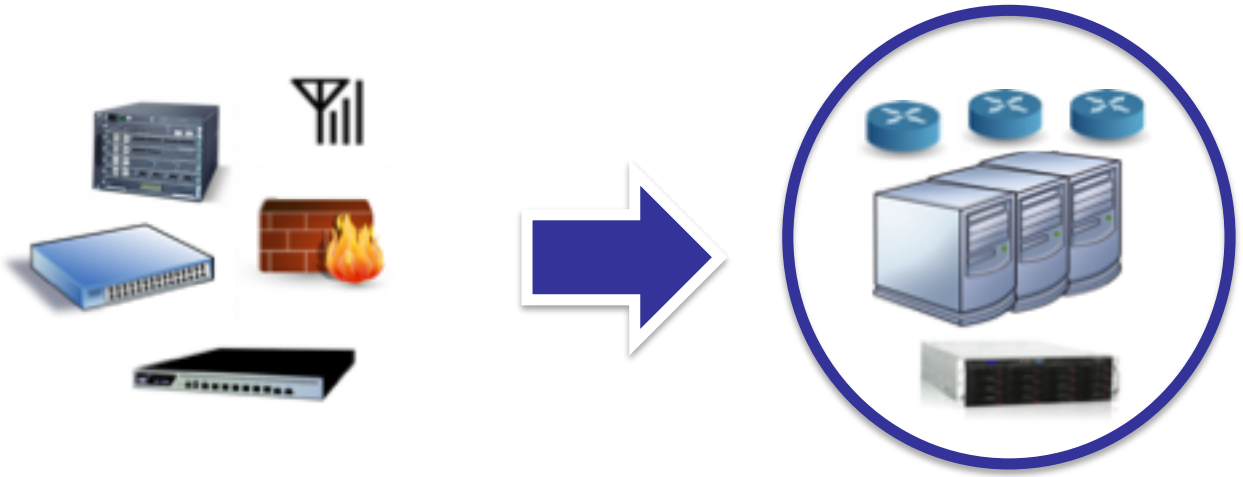
NFV

- NFV : Network Function(s) Virtualization (Virtualisation)
  - ネットワーク機能の仮想化
  - 専用機器で行われていたサービスを仮想化を用いて
    汎用機器で提供する



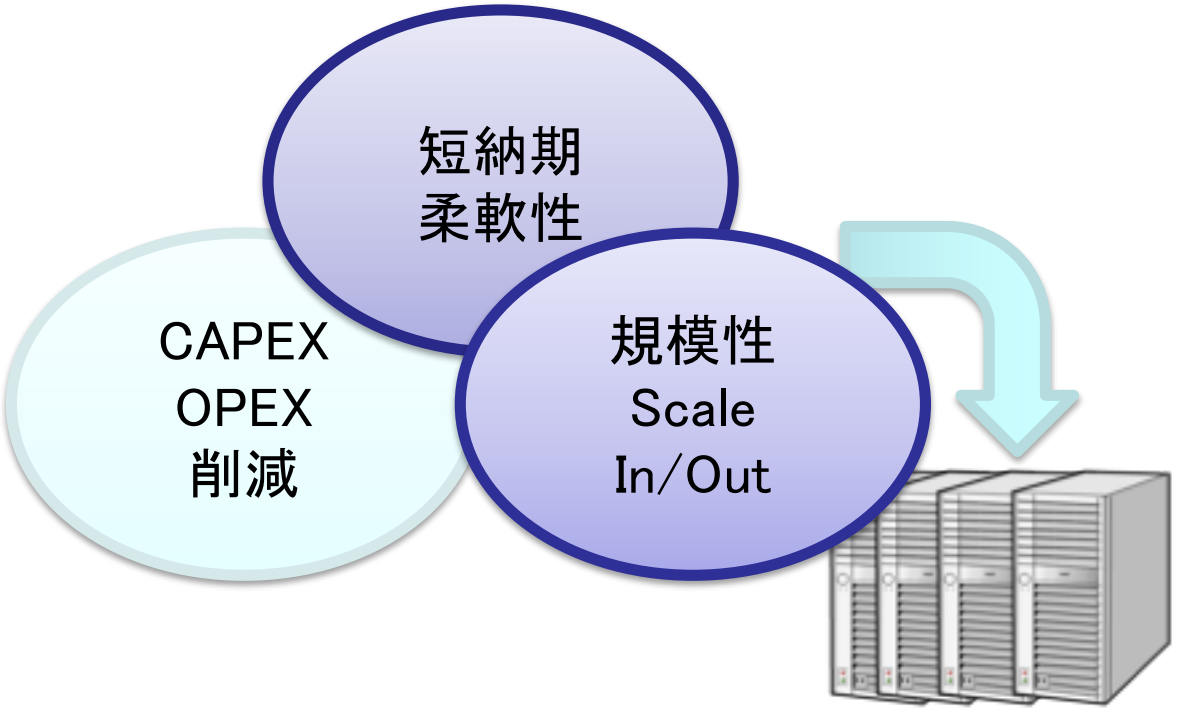
NFV の実装

- VNF (Virtual Network Function)
  - 現状は仮想アプライアンス (VM) として従来のハードウェア機器を置き換える
  - ネットワーク機能を提供する
  - VM にすることで構成の柔軟性を実現



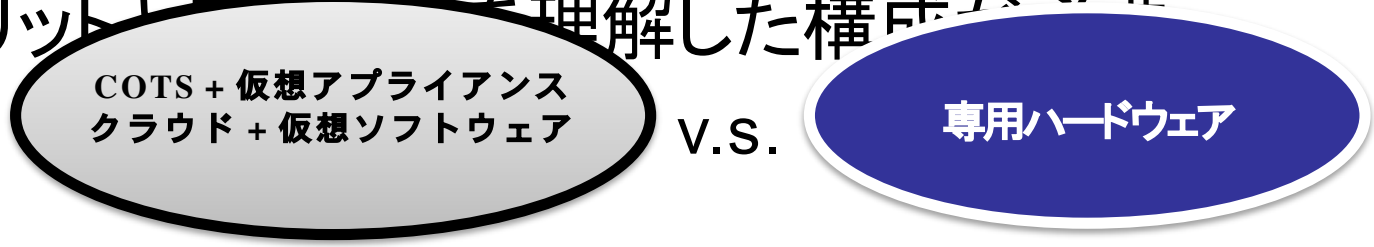
NFV の利点

- VNF
  - CGN
  - BRAS (Broadband Remote Access Server)
  - EPC (Evolved Packet Core)
  - CPE (Customer Provided Equipment)
  - VPN Gateway
  - Firewall / DPI
  - Load Balancer
  - Router

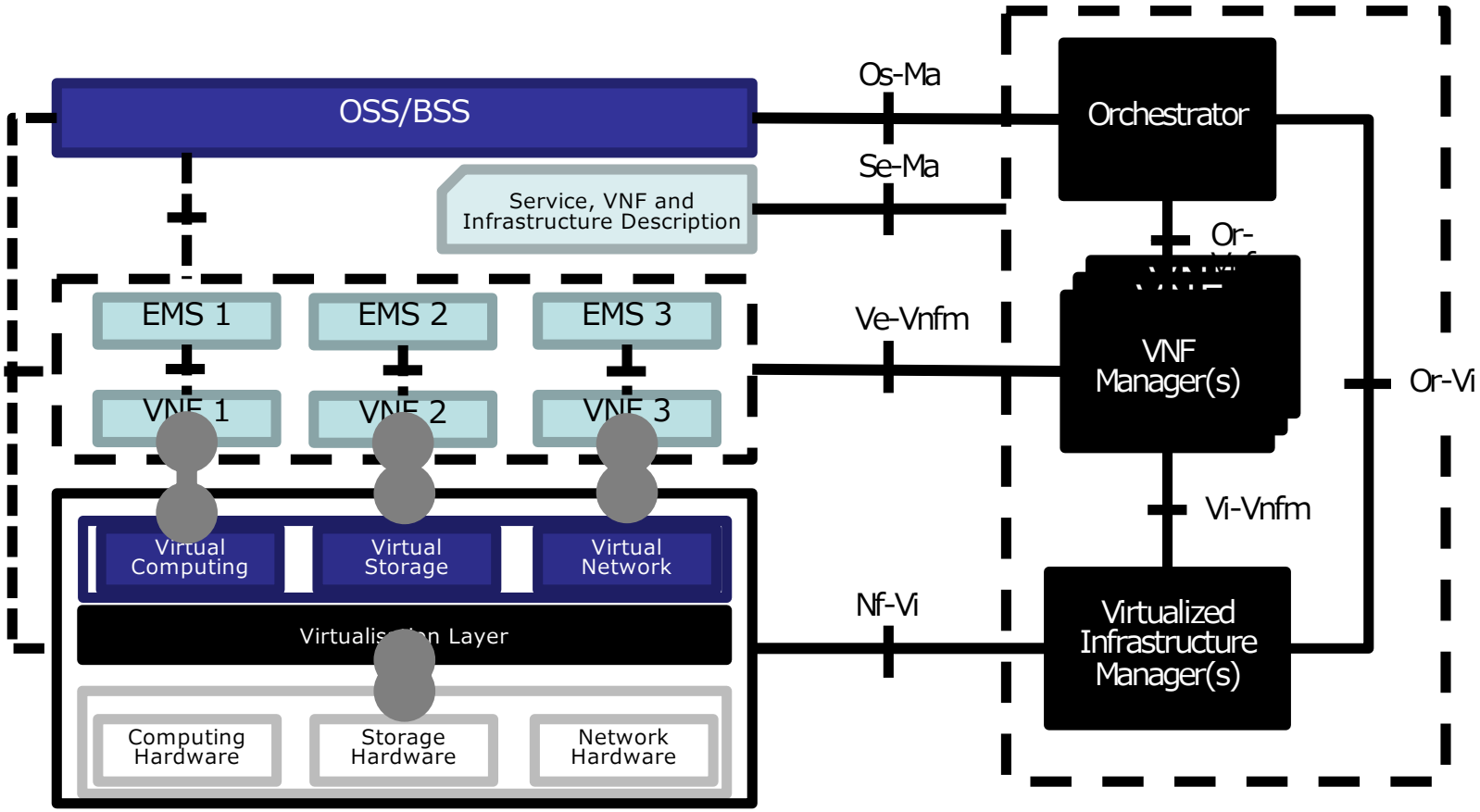


本当にコストは安くなるのか

- 実は専用機でも多重化、仮想化はできる
  - リソース割り当ての変更による融通は可能
  - 役割の変化は不可能
- 「性能」は専用ハードウェアに軍配が上がる場合が多い
  - クラウド + 仮想ソフトウェアは高性能な場合もあるが時間単価が高い
- メリットデメリットを理解した構成が必要



ETSI NFVアーキテクチャ



必要となる要素技術

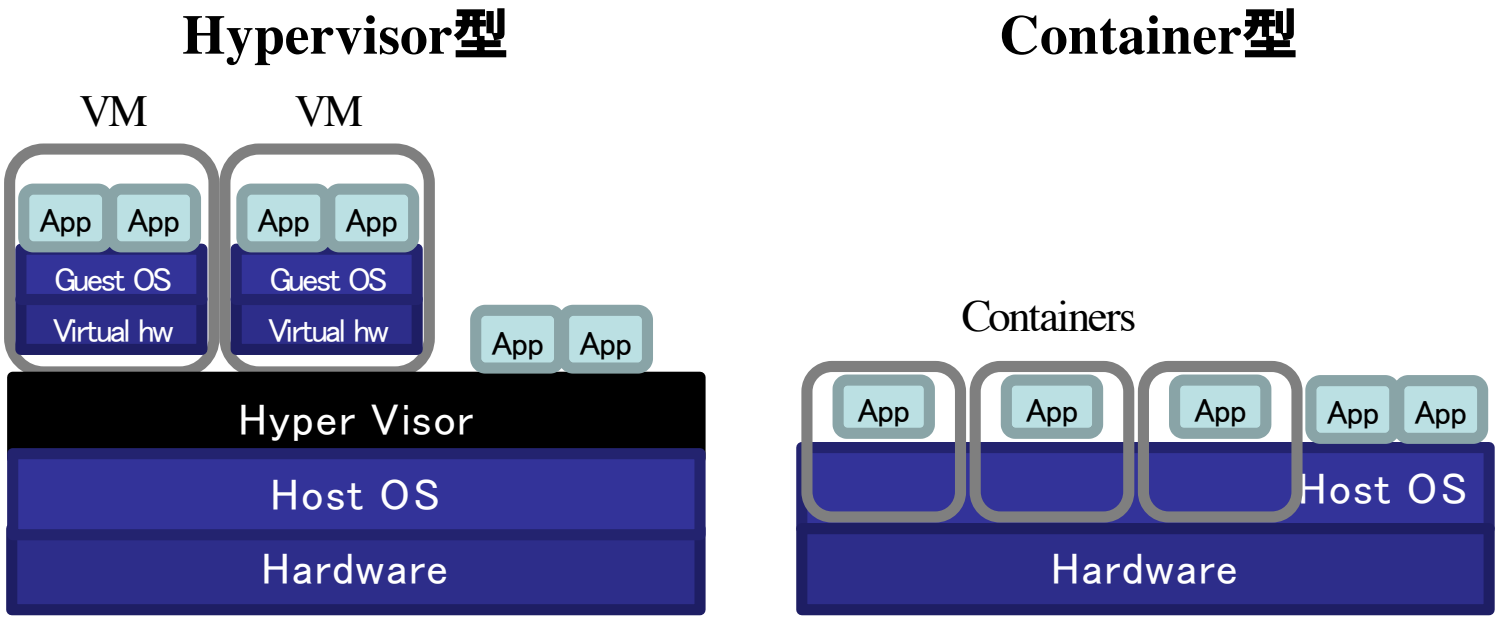
- サービス仮想化技術
  - ホスト仮想化 (VM)
  - プロセス仮想化 (コンテナ)
- ネットワーク仮想化技術
  - ネットワークコントローラ (SDN コントローラ)
  - オーバーレイネットワーク制御 (ソフトウェアネットワーク)
  - ネットワーク仮想化
- ファンクション制御
  - ファンクションの定型化と起動・停止
  - 仮想化ファンクションの制御と監視 (ライフサイクルマネージメント)
  - リソース使用量と負荷の監視
- オーケストレーション
  - API により全ての部品を統合管理

サービス仮想化

- ホスト仮想化（VM）
  - VMware や kvm, Hyper-V 等の仮想化技術と
  - それを制御する OpenStack や vCenter 等の VM マネージメント環境
- プロセス仮想化
  - Docker に代表されるコンテナ管理技術

- どちらもサービス単位を従来の専用ハードウェアよりも細かく制御可能
- 専用ハードウェアよりも性能面や安定性で劣る
- **NFVI（NFV Infrastructure）と呼ばれる部分**

ホスト仮想化技術


Hypervisor型
VM
VM
App
App
App
App
Guest OS
Guest OS
Virtual hw
Virtual hw
App
App
Hyper Visor
Host OS
Hardware
Container型
Containers
App
App
App
App
App
Host OS
Hardware

ホスト仮想化技術の比較

Hypervisor型

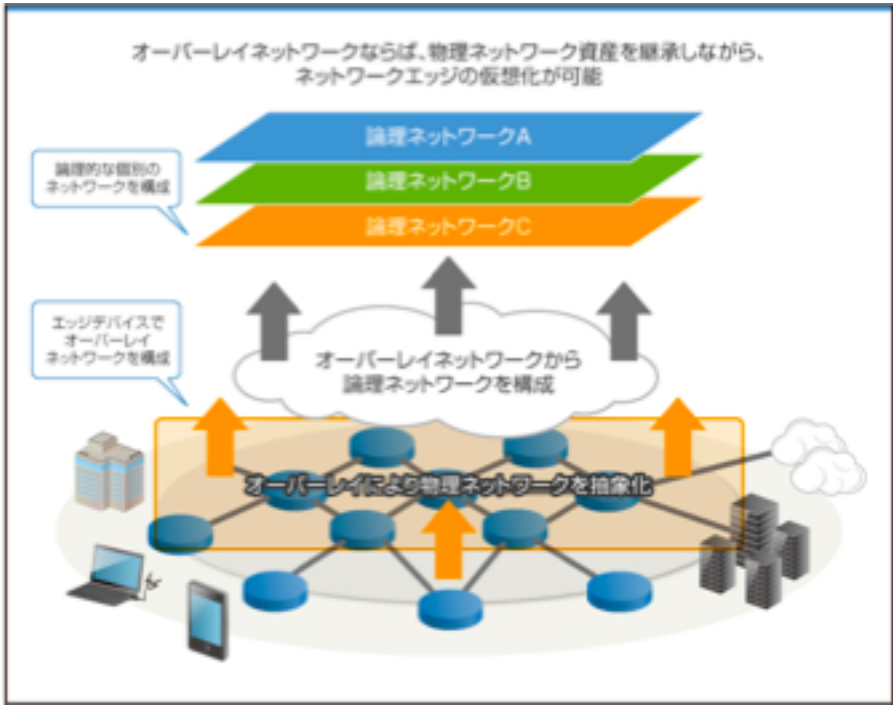
- Hypervisorが存在している
- Machineのハードウェアを仮想的に再現している
- よってオーバヘッドが存在

Container型

- 隔離したところでプロセス(アプリ)を実行しているだけ
- オーバーヘッドがほぼゼロ

ネットワーク仮想化技術

- オーバーレイ技術を用いたネットワーク仮想化
  - VXLAN + EVPN
  - VMware Distributed Switch
  - VMware NSX
  - OVN (OVS distributed switch)
- ネットワークコントローラによる制御
  - OpenDaylight
  - ONOS
  - 商用コントローラ
- ネットワーク回線にとらわれない論理ネットワーク構成
- NFVI の一部とみなされる



出典 : http://cn.teldevice.co.jp/column/detail/id/21