JANOG 52 Lightning Talk #5

全国キャリア網の耐障害設計で、OSS使って シミュレーションしたら爆速で切り分けられた件

2023年7月5日

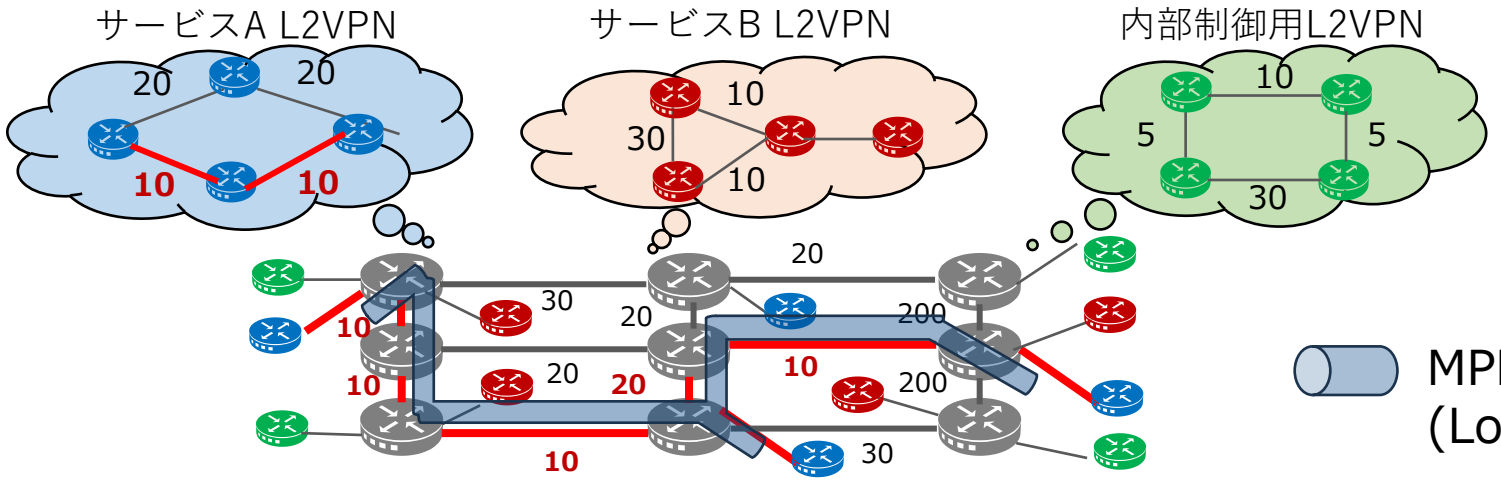
NTTコミュニケーションズ株式会社
門脇 伸明

全国キャリア網におけるIGP周りの課題

日本全国に張り巡らされた専用線と数百台のルータを抱え、多くのオーバーレイネットワークを提供する
閉域のキャリア網では、IGPのメトリック設計が複雑で、設計やテスト、障害時トラブルシュートが大変

オーバーレイ（L2VPN）
レイヤ

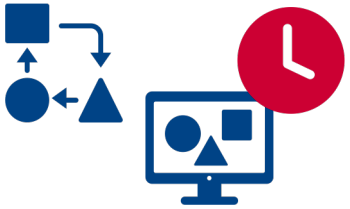
アンダーレイ（インフラ）
レイヤ



MPLS-VPNのLSP
(LooseでIGP制御)

想定経路と障害時迂回
経路計算が複雑

期待外経路か瞬時の判別がつかず、
切り分け時間かかる

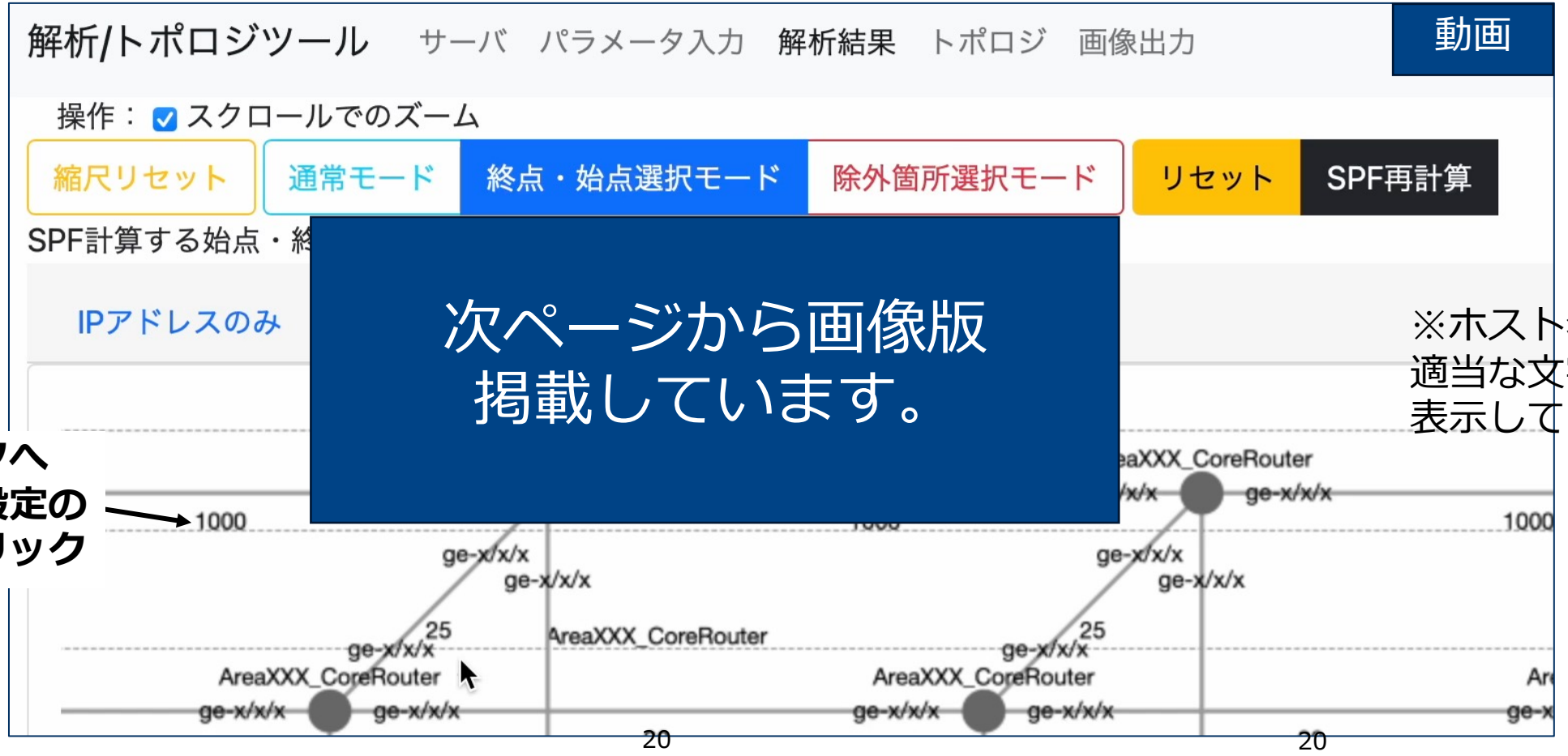


想定経路の**作図に時間がかかる**

~~面倒＋時間がなくて~~作図敬遠
→ リモートワークで伝わらない

例1.OSSでの経路シミュレーション描画（OSPFでの例）

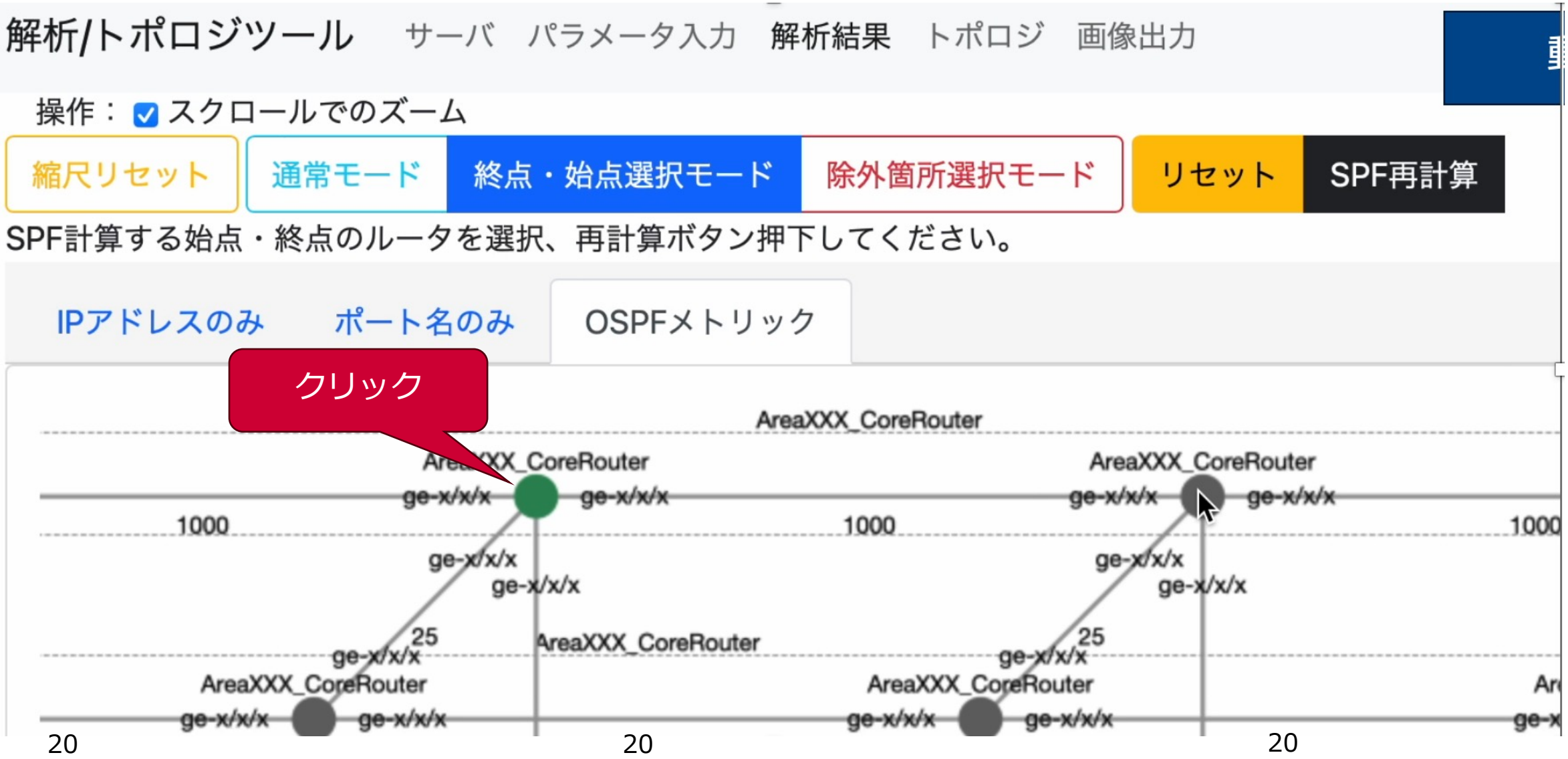
IGPメトリックに従った経路や迂回経路を計算し、トポロジ図にオーバーレイ表示



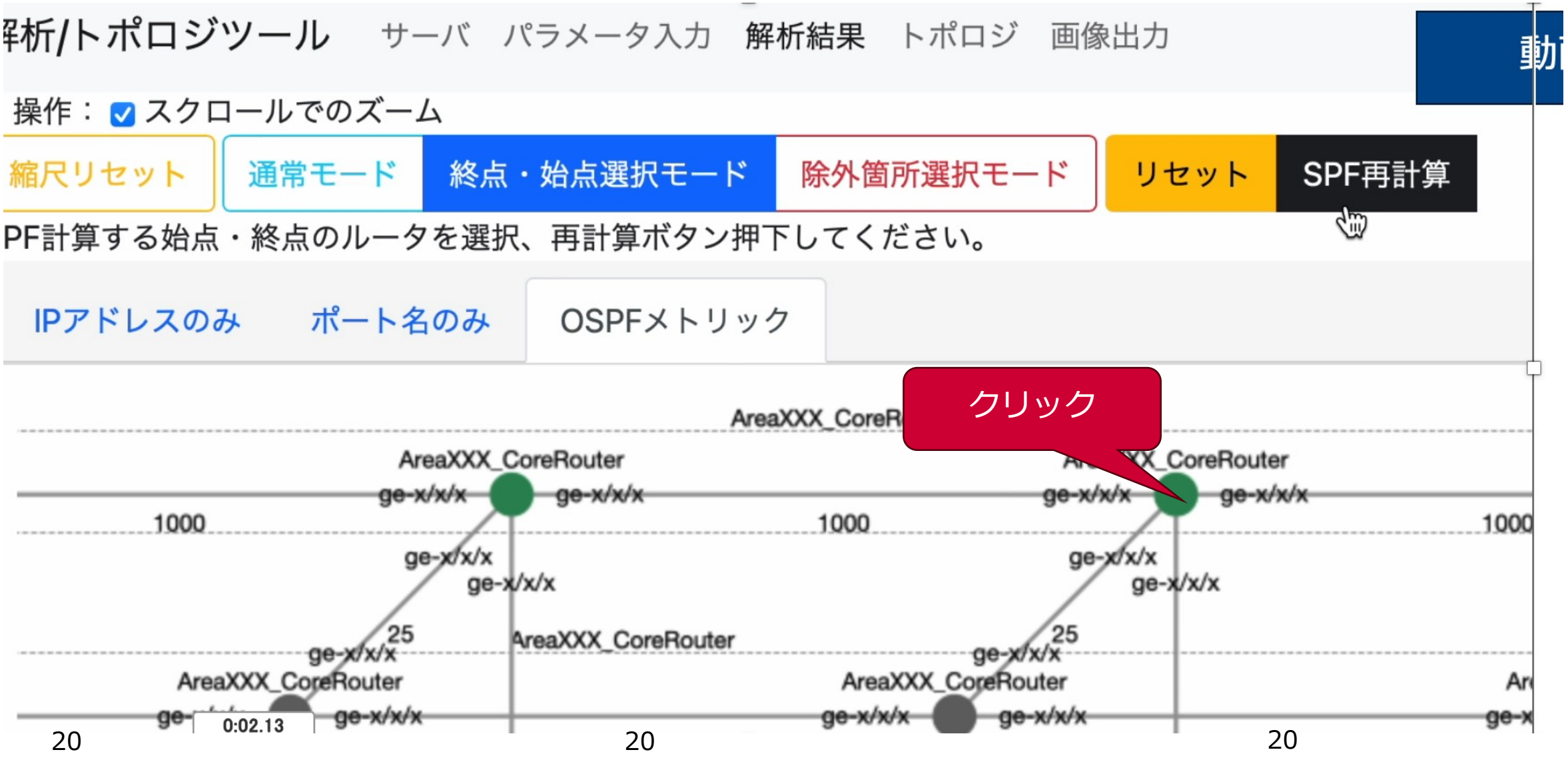
※ホスト名、ポート名は適当な文字に置き換えて表示しています。

メトリックから正常時/障害時迂回経路を簡単に確認

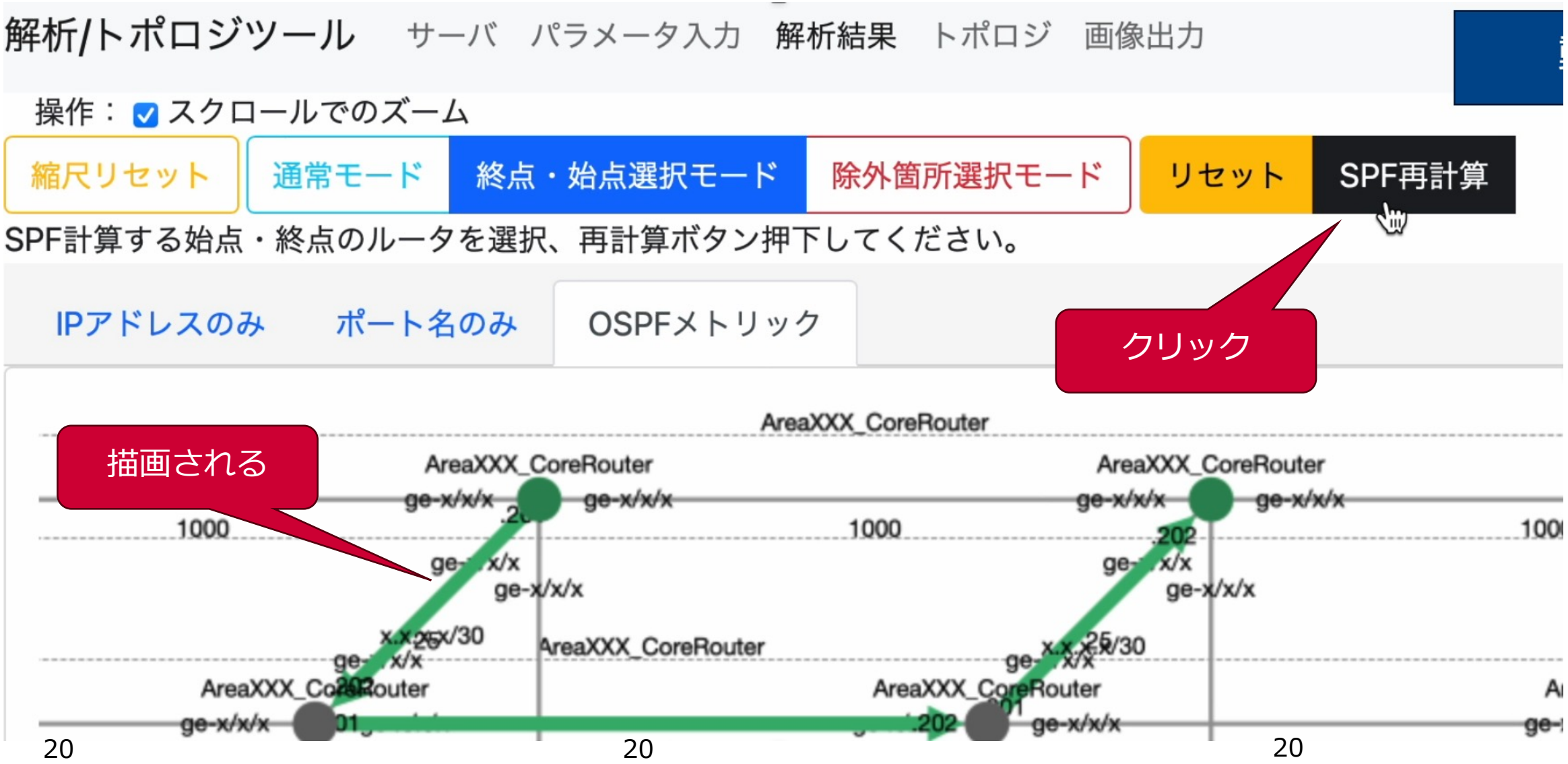
step1.始点(source/送信元)のルータ指定



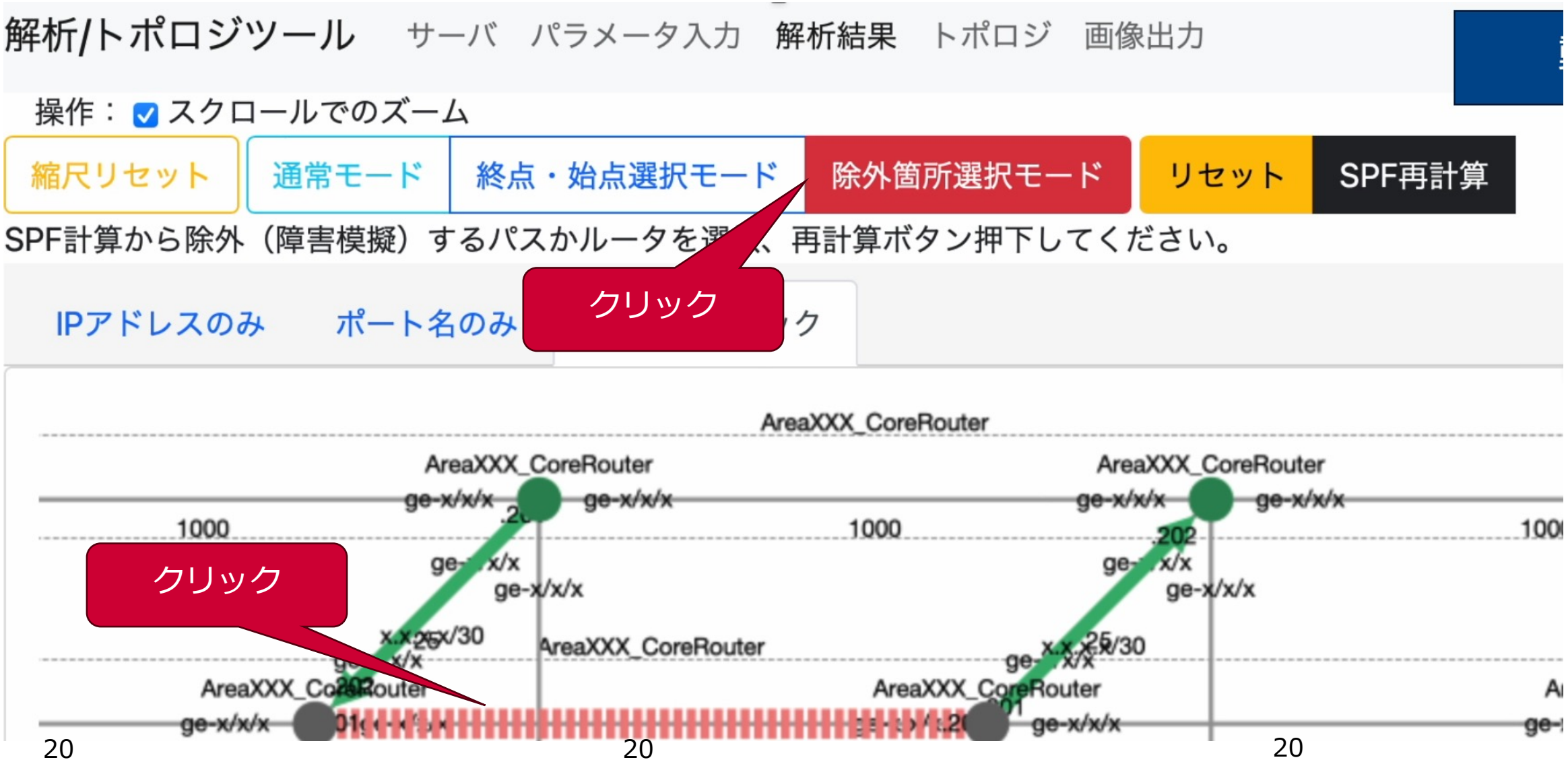
step2.終点(dest/宛先)のルータ指定



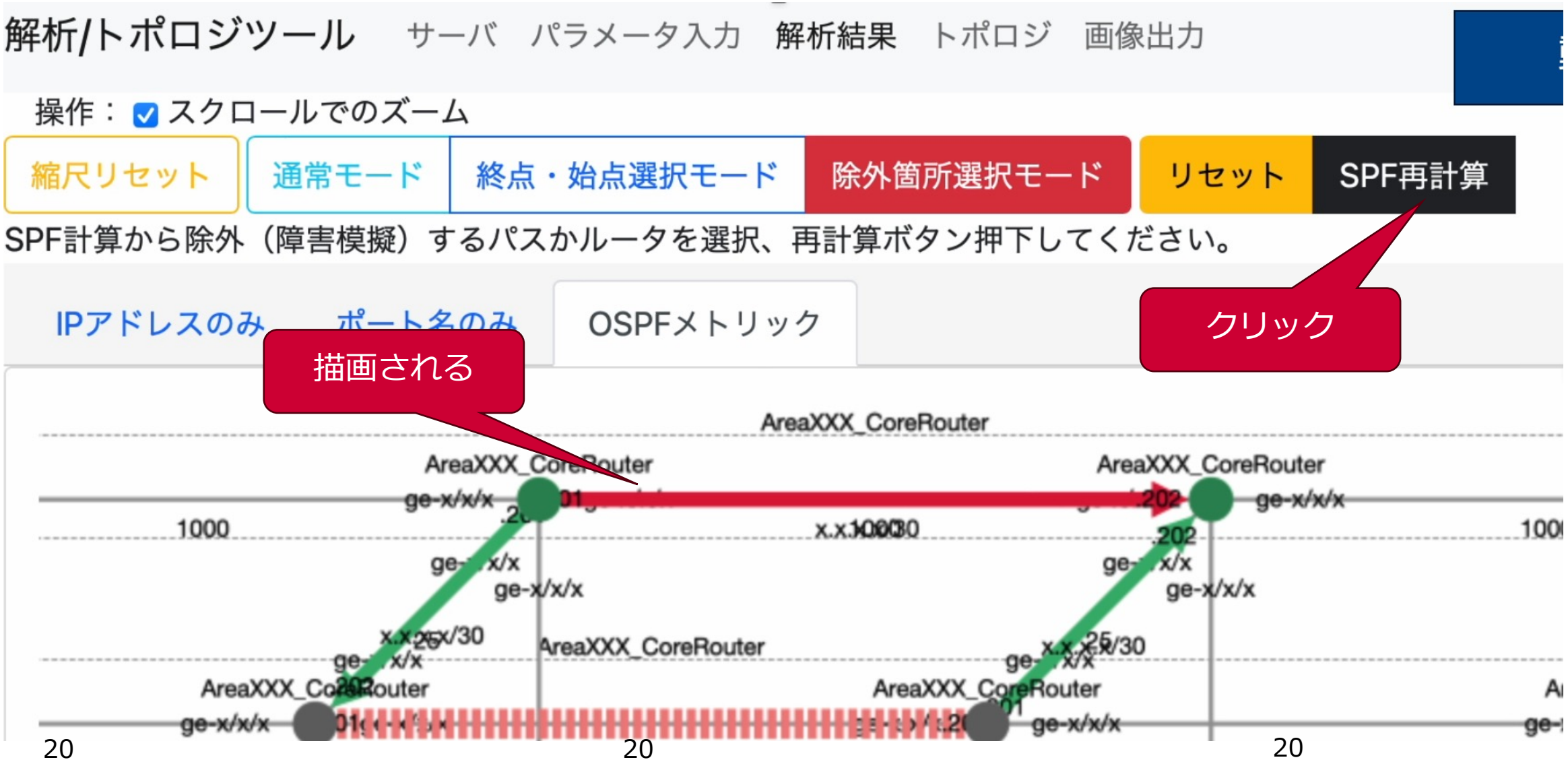
step3.SPF計算ボタン押下で想定経路描画



step4.障害模擬箇所（リンクorルータ）を選択



step5.障害時の迂回経路を赤で描画



例2.OSSでの実際の経路（Traceroute）の可視化例

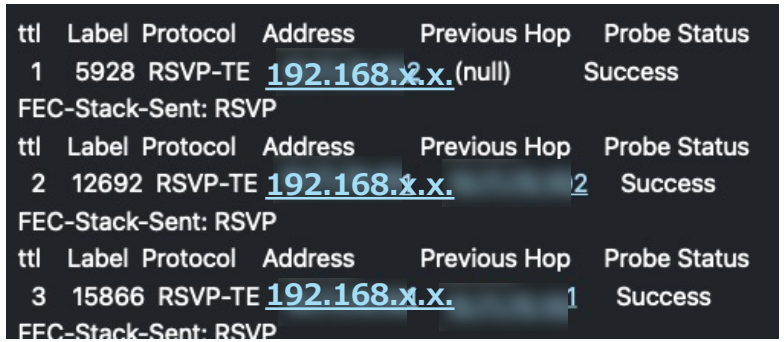
IPアドレスの羅列から経路の正常性判定が難しい

```
testuser@KyushuTestRouter-re0> traceroute mpls test-lsp-tohoku-kyushu
Jan 13 18:38:05

  ttl    Label  Protocol    Address          Previous Hop      Probe Status
1     7042  RSVP-TE     192.168.23.202      (null)          Success
2      647  RSVP-TE     192.168.30.222      192.168.23.202
3     1955  RSVP-TE     192.168.34.122      192.168.30.222
<省略>
13       3  RSVP-TE     192.168.70.121     192.168.67.122     Egress
```

ルータでのtracerouteログ

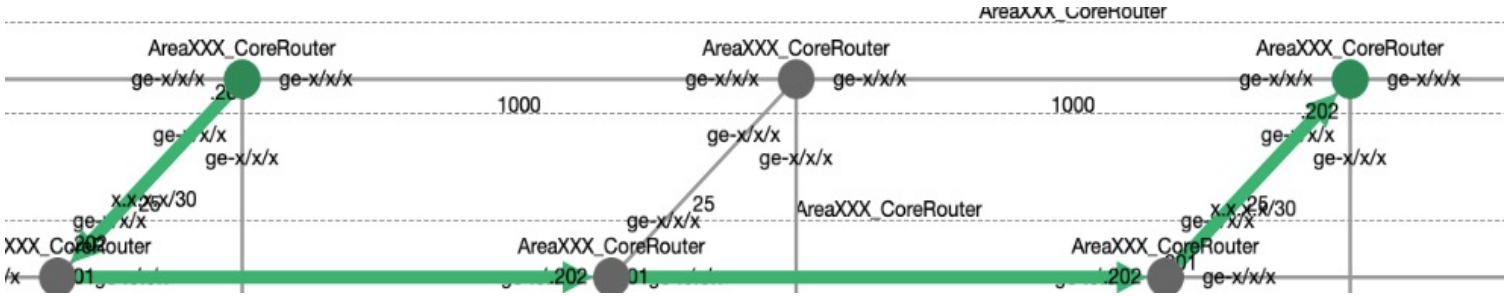
ツールに貼り付け



上のホップから順にアドレスをクリック

Cytoscape.js

Graph theory (network) library for visualisation and analysis



Traceroutelogから実際の経路を描画

実際の経路の描画で設計通りかの確認が容易

作り方

残り時間で話せるところまで…

続きはWeb掲載資料で！

Step1.コンフィグからOSPF設定とポート設定抽出

コンフィグからテンプレートエンジンで、設定をデータベース化

コンフィグ

テキストパーサ

google/**textfsm**

Python module for parsing semi-structured text into python tables.

アドレスや
OSPFメトリックの
データベース

```
set system host-name Kanto2-PE

set interfaces xe-0/1/0 unit 30 description Kanto_Core_10G
set interfaces xe-0/1/0 unit 30 vlan-id 30
set interfaces xe-0/1/0 unit 30 family inet address 192.168.60.12/30
```

インターフェースのアドレス設定
＋OSPFメトリック設定

```
1   Value Name (\S+)
2   Value LogicalUnit (\d+)
3   Value Desc (\S+)
4   Value Addr (\S+)
5   Value Speed (\S+)
11  Start
12    ^set interfaces ${Name} speed ${Speed} -> Record
13    ^set interfaces ${Name} unit 0 family inet address ${Addr} -> Record
14    ^set interfaces ${Name} description ${Desc} -> Record
```

コンフィグ抽出の
テンプレート

db.sqlite3 — conf_if 2331 rows, showing page 1

new_table_name Create

table name...
conf_clock
conf_if
host
link
port
Toggle helper tables

Structure | Content | Query | Drop | Import

id	host	name	unit	desc	speed	ip_address	vlan_id	ospf_metric	rsvp	ospf
1	[redacted]	et-0/0/[redacted]		[redacted]	100g	[redacted].2/30		2000	True	True
2	[redacted]	et-1/0/[redacted]		[redacted]	100g	[redacted].1/30		2000	True	True

Step2.ポート設定からグラフ構造データ自動構築

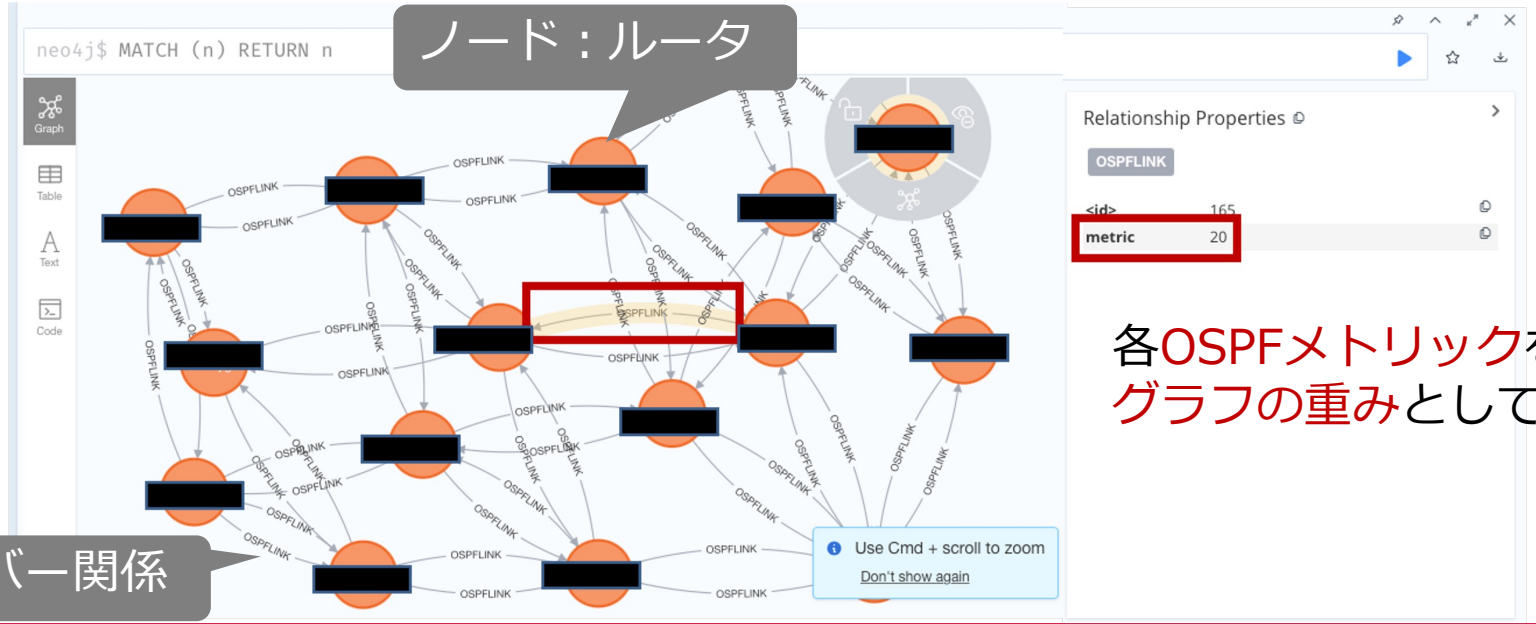
インターフェース設定のデータベース情報から、ルータ間の接続関係を推定して、グラフ構造データに変換

OSPFメトリックの
データベース

ネットワークトポロジを
グラフデータとして登録

neo4j

グラフデータベースOSS



エッジ：OSPFネイバー関係

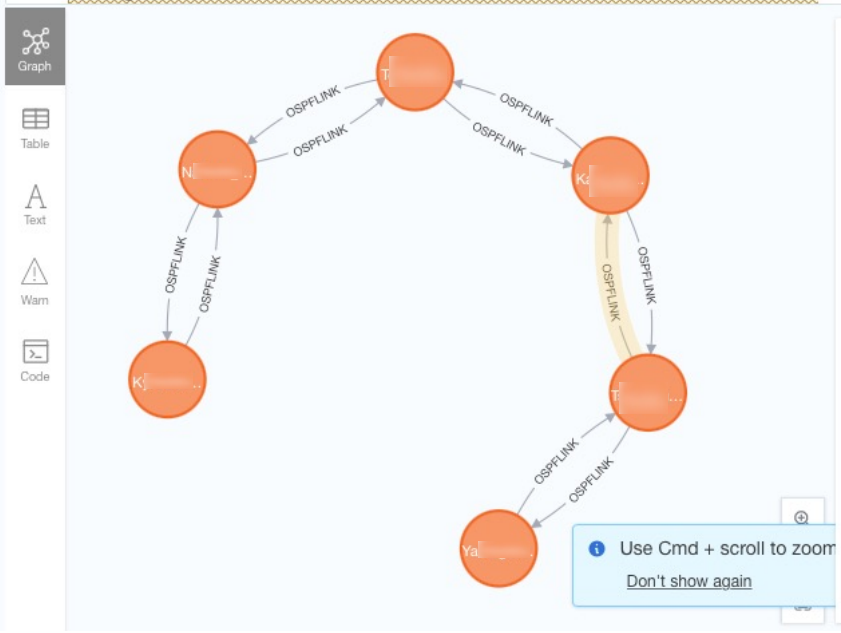
各OSPFメトリックを
グラフの重みとして登録

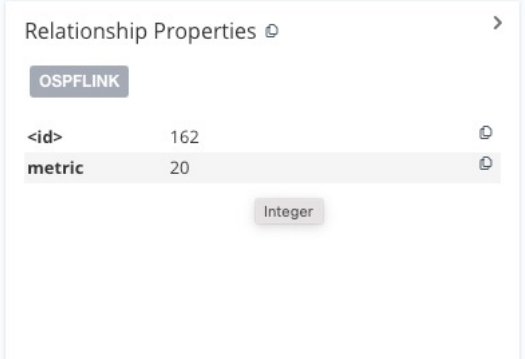
Step3.グラフデータベースOSSで経路計算

複数の冗長経路の中から、OSPFメトリックが最小となる経路を要求

```
neo4j$
1  //ShowCostShortestPath_GDS
2  MATCH (sourceR:Router {name: 'Y          A'}), (targetR:Router {name:
   'K          R_A'})
3  CALL gds.shortestPath.dijkstra.stream('test-proj', {
4      sourceNode: sourceR,
5      targetNode: targetR,
6      relationshipWeightProperty: 'metric'
7  })
8  YIELD index, sourceNode, targetNode, totalCost, nodeIds, costs, path
9  RETURN
10     index,
11     gds.util.asNode(sourceNode).name AS sourceNodeName,
```

始点・終点ホスト名を指定

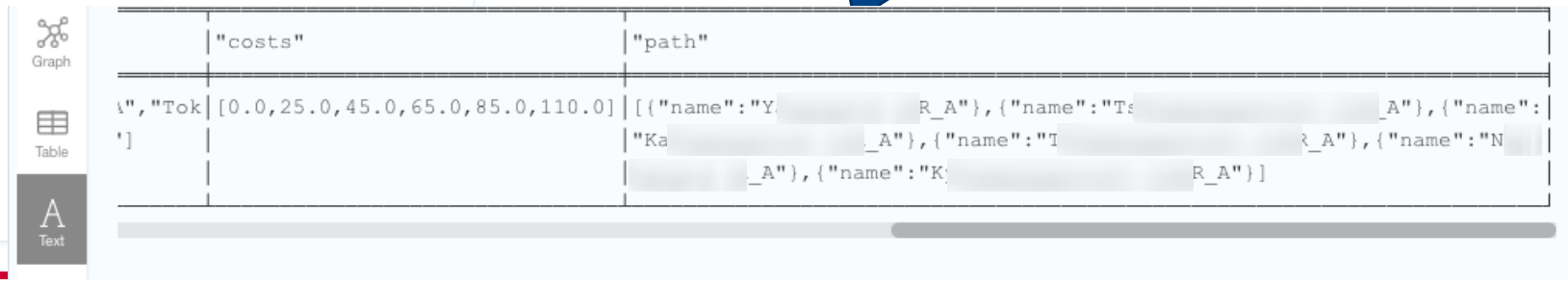




neo4j

グラフデータベースの
最短経路計算機能を利用

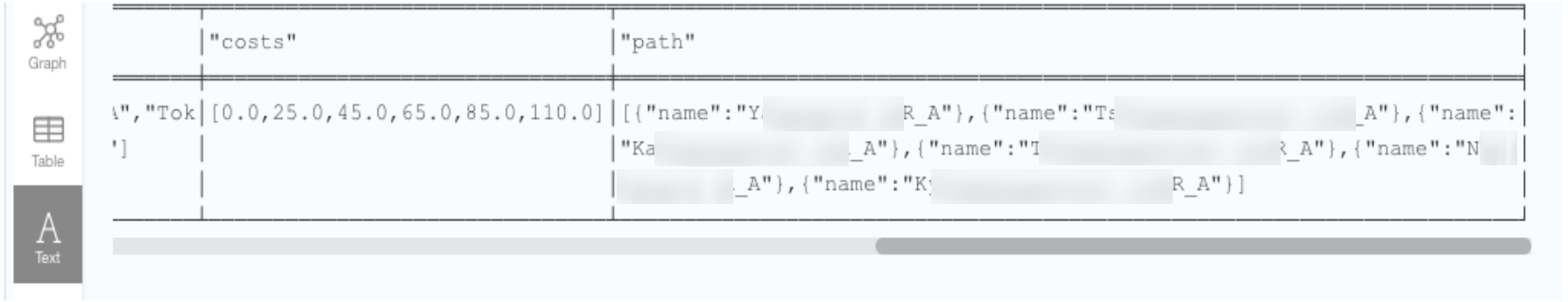
各ホップのホスト名が
Json形式で返ってくる



Step4.算出経路をグラフデータの可視化OSSで描画

neo4j

計算した最短経路



変換

点と線のデータ

Cytoscape

```
{ "data": { "id": "AreaXX_CoreRouter_A"}},
{ "data": { "id": "AreaXX_CoreRouter_B"}},
```

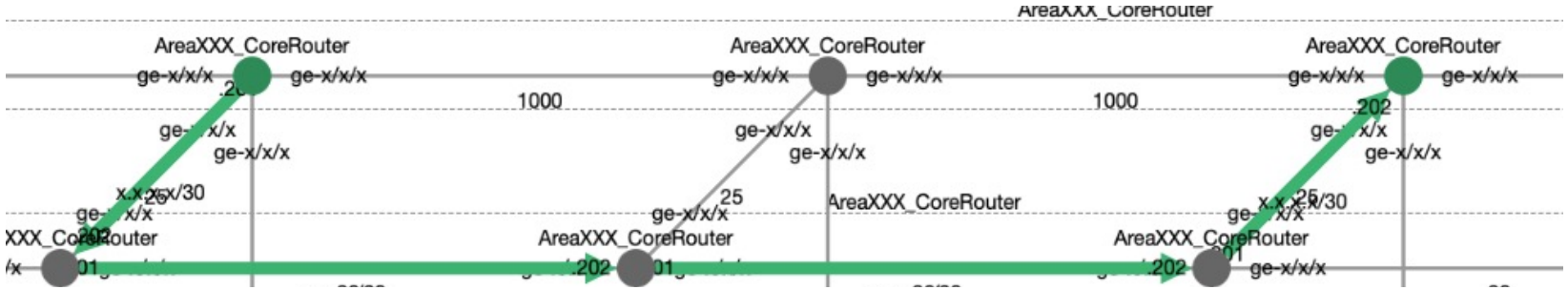
ルータ（点の情報）

```
{ "data":{ "id": "LinkXX",
  "source": "AreaXX_CoreRouter_A", "target": "AreaXX_CoreRouter_B",
  "label": "1000", "source-label": "ge-x/x/x", "target-label": "ge-x/x/x" }
```

リンクおよび算出経路
（辺・線の情報）

描画

トポロジ図
＋経路図



参考スライド（Web掲載用）

OSSでの実際の経路（Traceroute）の可視化

IPアドレス・ホスト名の羅列から経路の正常性判定が難しい

```
testuser@KyushuTestRouter-re0> traceroute mpls test-lsp-tohoku-kyushu
Jan 13 18:38:05

 ttl   Label  Protocol   Address          Previous Hop      Probe Status
1     7042  RSVP-TE    192.168.23.202     (null)          Success
2      647  RSVP-TE    192.168.30.222     192.168.23.202
3     1955  RSVP-TE    192.168.34.122     192.168.30.222
<省略>
13       3  RSVP-TE    192.168.70.121     192.168.67.122     Egress
```

障害試験時のルータでのtraceroute
出力例（13ホップ）

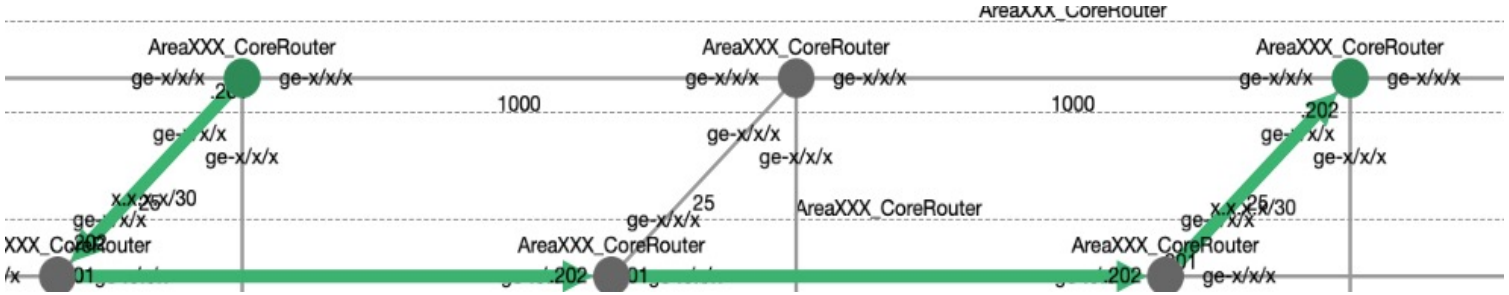
step1のDBから対応ホスト名割り出し

```
ttl   Label  Protocol   Address          Previous Hop      Probe Status
 1    5928  RSVP-TE    192.168.x.x.      (null)           Success
FEC-Stack-Sent: RSVP
ttl   Label  Protocol   Address          Previous Hop      Probe Status
 2   12692  RSVP-TE    192.168.x.x.                       Success
FEC-Stack-Sent: RSVP
ttl   Label  Protocol   Address          Previous Hop      Probe Status
 3   15866  RSVP-TE    192.168.x.x.                       Success
FEC-Stack-Sent: RSVP
```

アドレスのリンクをクリックで、
クリックしたアドレスに対応するホスト名から、step4のCytoscapeの緑線の始点・終点に置き換えてJson形式で入力

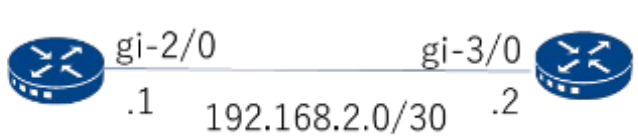
Cytoscape.js

Graph theory (network) library for visualisation and analysis



Cytoscape使えば実際の経路の描画も容易

トポロジの推定（IGPネイバー）



下記アプローチが一番正確だが。。。

- LLDP（Link Layer Discovery Protocol）
- BGP Link State
- OSPF

適用が難しい

- 構築前の開発工程 → 実機が少ない
- 検証網　→ 全国網を模擬していない

商用適用予定のコンフィグを
テキストマイニングしてトポロジ推定

設定ファイル
host-name Tohoku-a
interface gi-2/0
 address 192.168.2.1/30
 description CoreLink_Tohoku-Kanto

設定ファイル
host-name Kanto-a
interface gi-3/0
 address 192.168.2.2/30
 description CoreLink_Kanto-Tohoku

テンプレートエンジンで抜き出し

Host	Port	Address	Desc
Tohoku-a	gi-2/0	192.168.2.1/30	CoreLink_Tohoku-Kanto
Tohoku-a	gi-2/1	…	…

Host	Port	Address	Desc
Kanto-a	gi-3/0	192.168.2.2/30	CoreLink_Tohoku-Kanto
Kanto-a	gi-3/1	…	…

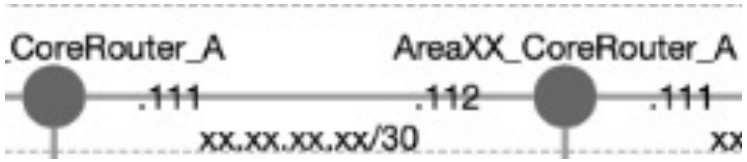
トポロジ推論

各ネットワークアドレス算出
同一セグメントが直結と推論

Segment	Host	Port1	Description	Port2
192.168.2.0/30	Tohoku-a	gi-2/0	Kanto-a	gi-3/0
	Tohoku-a	gi-2/1	…	…

専用線なら1:1接続で/30が多い

トポロジ



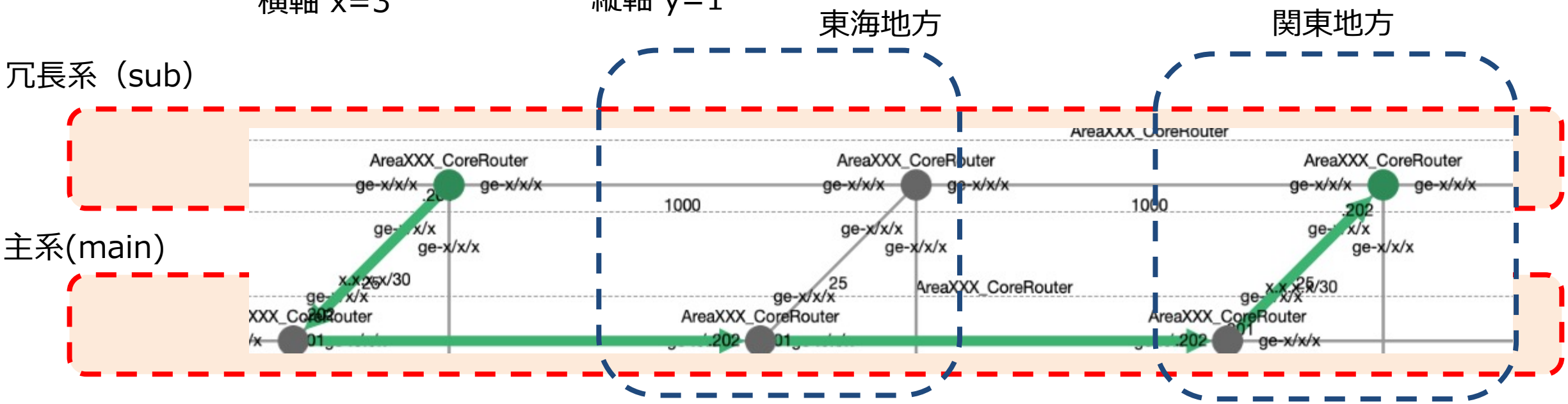
可視化の工夫

毎度ルータの位置がランダムに変わり、運用に使いづらい可視化ソリューションが多い

→ 今回はホスト名から描画位置を決定

ホスト名例：　AreaXXX_CoreRouter_main

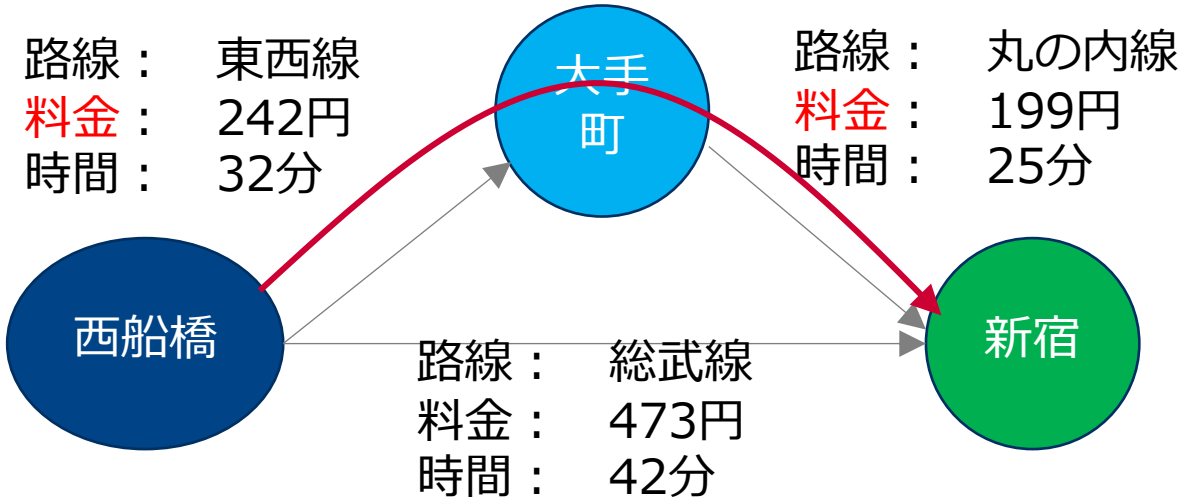
横軸 x=3　　縦軸 y=1



見やすいトポロジ図　→ 可視化ソリューションの質を決める

グラフデータベース

グラフ構造をベースとしたNoSQLの一種で、
ノード（点）とエッジ（線・辺）の結びつきでデータの表現・検索が得意



グラフデータベースへの登録（イメージ）

西船橋は新宿と総武線で接続　CREATE (:Station 西船橋) -> [:Train Line:総武線 Fare:473 Time: 42] -> (:Station 新宿)

西船橋は大手町と東西線で接続　CREATE (:Station 西船橋) -> [:Train Line:東西線 Fare:242 Time: 32] -> (:Station 大手町)

大手町は新宿と丸の内線で接続　CREATE (:Station 大手町) -> [:Train Line:丸の内線 Fare:199 Time: 25] -> (:Station 新宿)

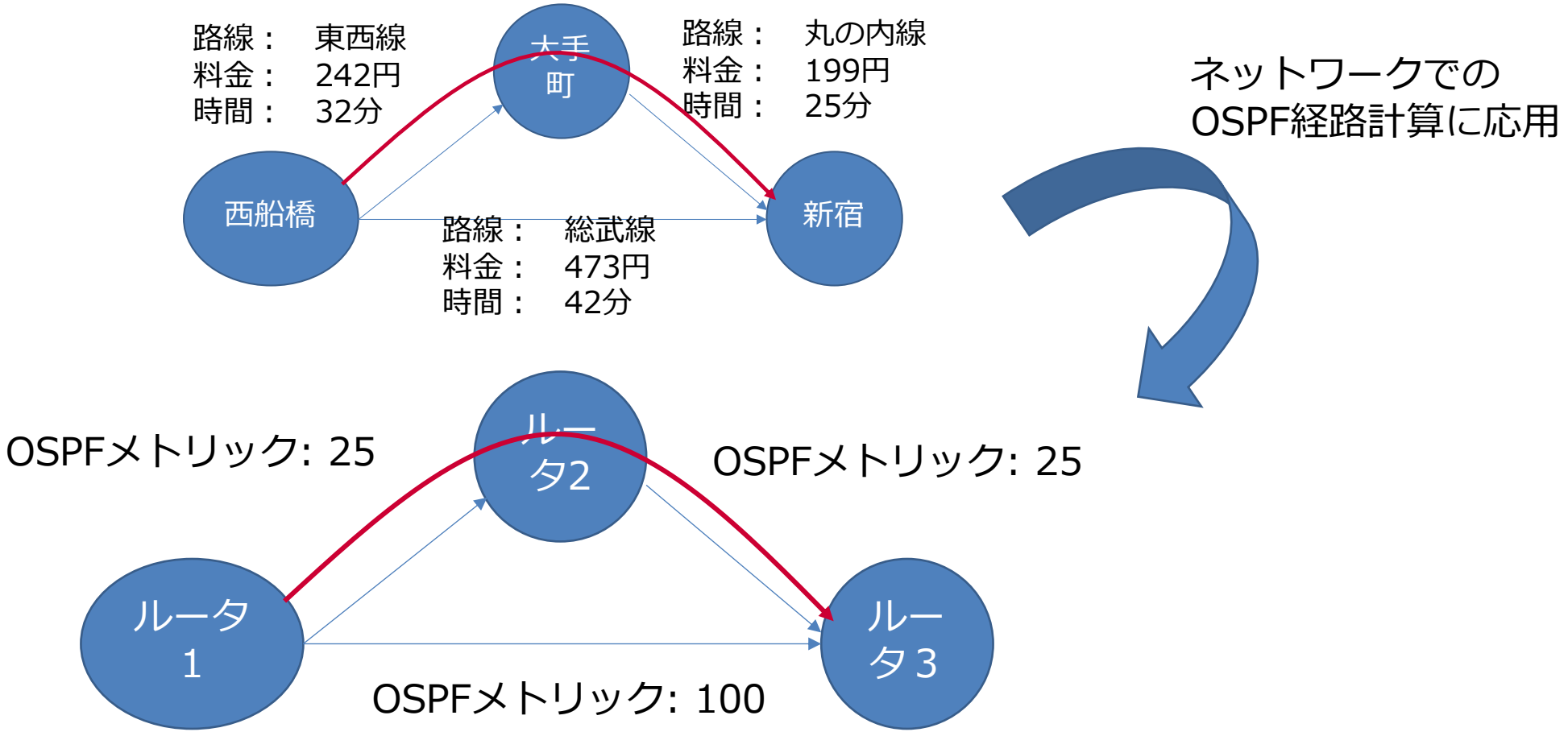
グラフデータベースへの経路検索クエリ（イメージ）

西船橋～新宿で料金が最安の経路は？
（≠時間、乗り換え回数）

```
MATCH (s:Station 西船橋), (d:Station 新宿)
, p = shortestPath((s)->[:Train*]->(d)) WHERE Fare….
```

グラフデータベースの経路計算への適用

グラフの点（ノード）をルータ、辺（エッジ）をルータ間のリンク、コストをOSPFメトリックにすることで、
OSPFメトリックが最小の最短経路（SPF）が計算できる



グラフデータベース

テストコンフィグから自動生成した、グラフデータ