

JANOG 52 Lightning Talk #5

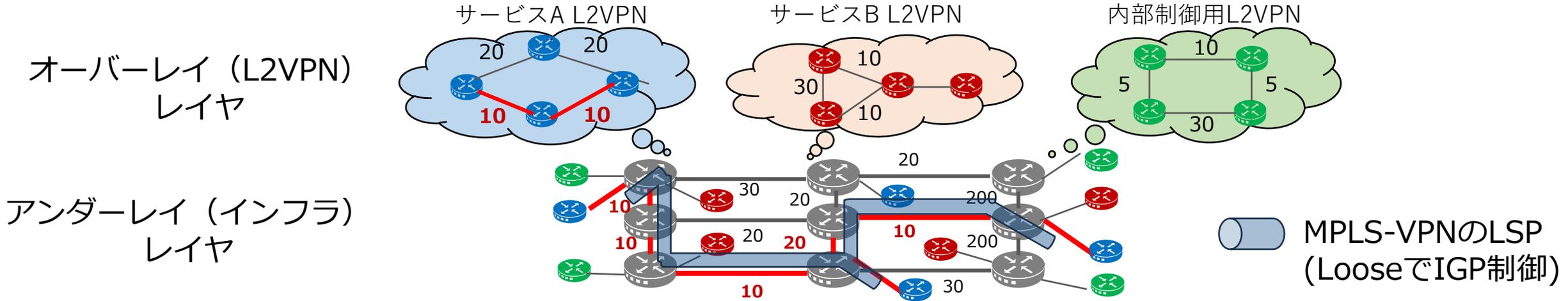
**全国キャリア網の耐障害設計で、OSS使って
シミュレーションしたら爆速で切り分けられた件**

2023年7月5日

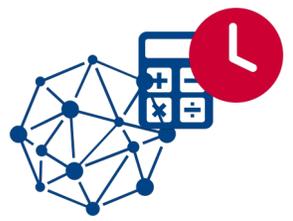
NTTコミュニケーションズ株式会社
門脇 伸明

全国キャリア網におけるIGP周りの課題

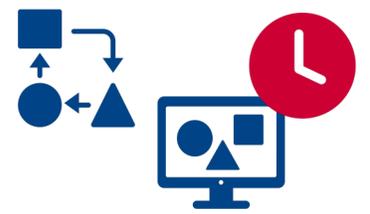
日本全国に張り巡らされた専用線と**数百台のルータ**を抱え、**多くのオーバーレイネットワーク**を提供する閉域のキャリア網では、IGPのメトリック設計が複雑で、設計やテスト、障害時トラブルシューティングが大変



想定経路と障害時迂回
経路計算が複雑



期待外経路が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

The screenshot shows a web-based interface for network simulation. At the top, there are navigation tabs: 解析/トポロジツール, サーバ, パラメータ入力, 解析結果, トポロジ, 画像出力, and a blue 動画 (Video) button. Below the tabs, there are control buttons: 縮尺リセット (Reset Scale), 通常モード (Normal Mode), 終点・始点選択モード (End/Start Selection Mode), 除外箇所選択モード (Exclude Selection Mode), リセット (Reset), and SPF再計算 (Recalculate SPF). A central blue box contains the text: 次ページから画像版掲載しています。 (The image version is published on the next page). To the right, a note says: ※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。 (Host names and port names are replaced with appropriate characters for display). The main area shows a network diagram with nodes labeled 'AreaXXX_CoreRouter' and links labeled 'ge-x/x/x'. Metrics are displayed on the links: 1000, 25, and 20. On the left, an arrow points to the '静的設定のメトリック' (Static configuration metrics) with the text 'ルータへ' (To the router).

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

解析/トポロジツール サーバ パラメータ入力 解析結果 トポロジ 画像出力

操作: スクロールでのズーム

縮尺リセット

通常モード

終点・始点選択モード

除外箇所選択モード

リセット

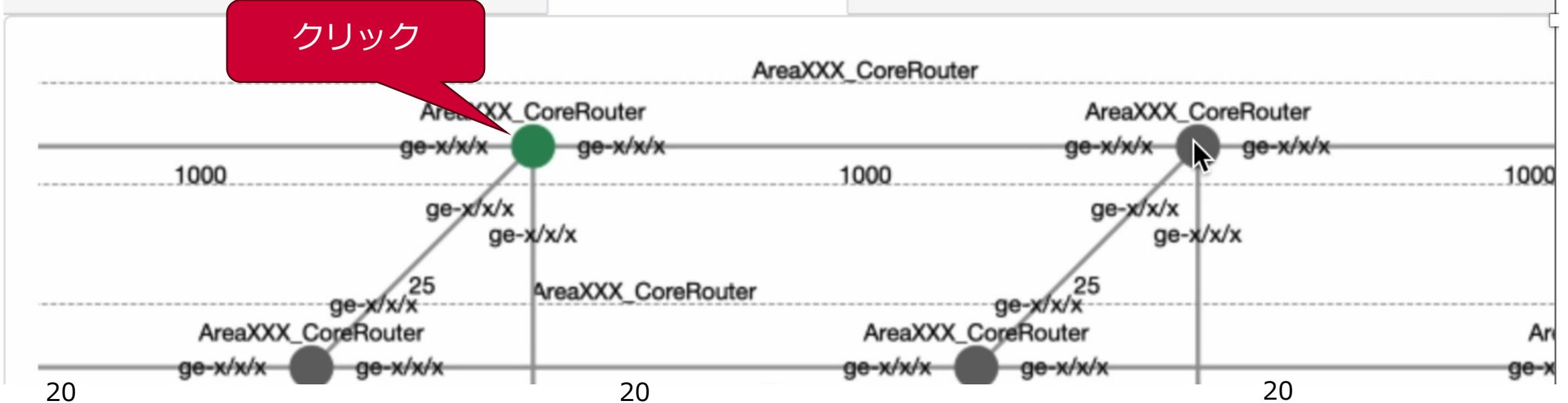
SPF再計算

SPF計算する始点・終点のルータを選択、再計算ボタン押下してください。

IPアドレスのみ

ポート名のみ

OSPFメトリック



step2. 終点(dest/宛先)のルータ指定

解析/トポロジツール サーバ パラメータ入力 解析結果 トポロジ 画像出力

動

操作: スクロールでのズーム

縮尺リセット

通常モード

終点・始点選択モード

除外箇所選択モード

リセット

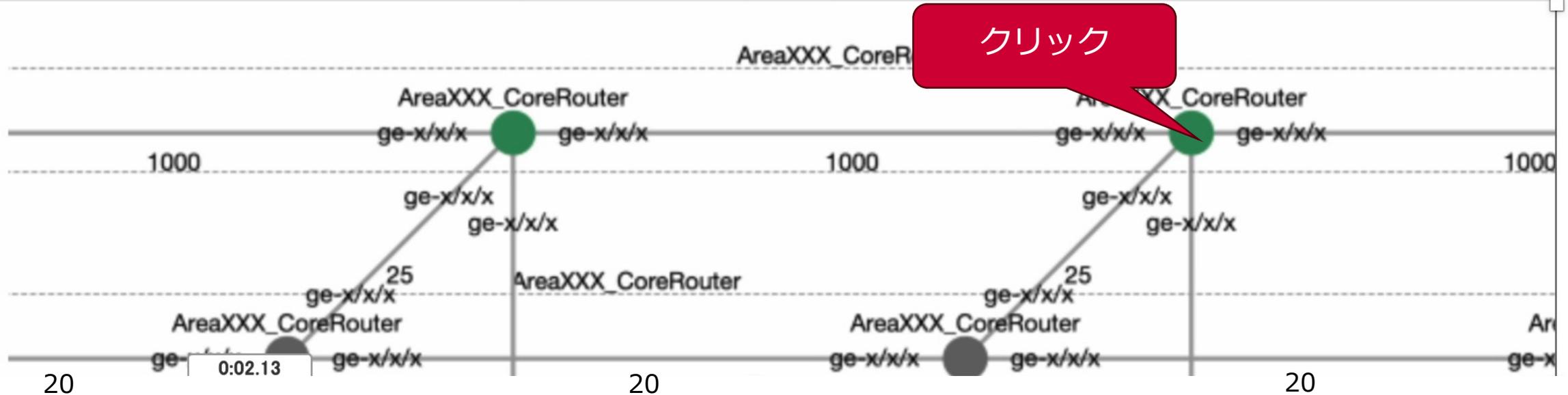
SPF再計算

SPF計算する始点・終点のルータを選択、再計算ボタン押下してください。

IPアドレスのみ

ポート名のみ

OSPFメトリック



step3.SPF計算ボタン押下で想定経路描画

解析/トポロジツール サーバ パラメータ入力 解析結果 トポロジ 画像出力

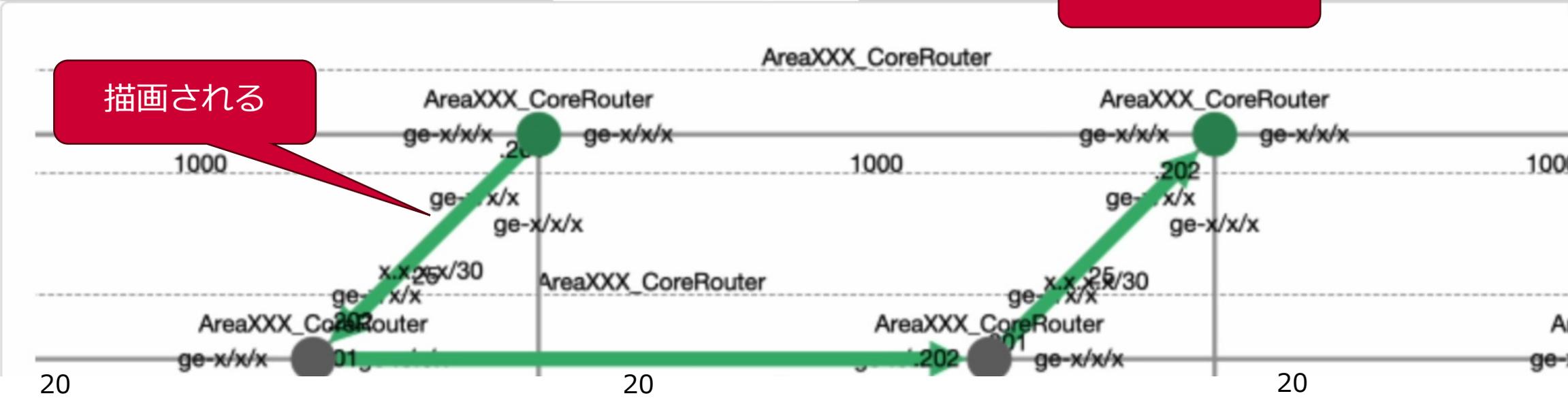
操作: スクロールでのズーム

縮尺リセット 通常モード **終点・始点選択モード** 除外箇所選択モード リセット **SPF再計算**

SPF計算する始点・終点のルータを選択、再計算ボタン押下してください。

IPアドレスのみ ポート名のみ OSPFメトリック

クリック



step4.障害模擬箇所（リンクorルータ）を選択

解析/トポロジツール サーバ パラメータ入力 解析結果 トポロジ 画像出力

操作： スクロールでのズーム

縮尺リセット

通常モード

終点・始点選択モード

除外箇所選択モード

リセット

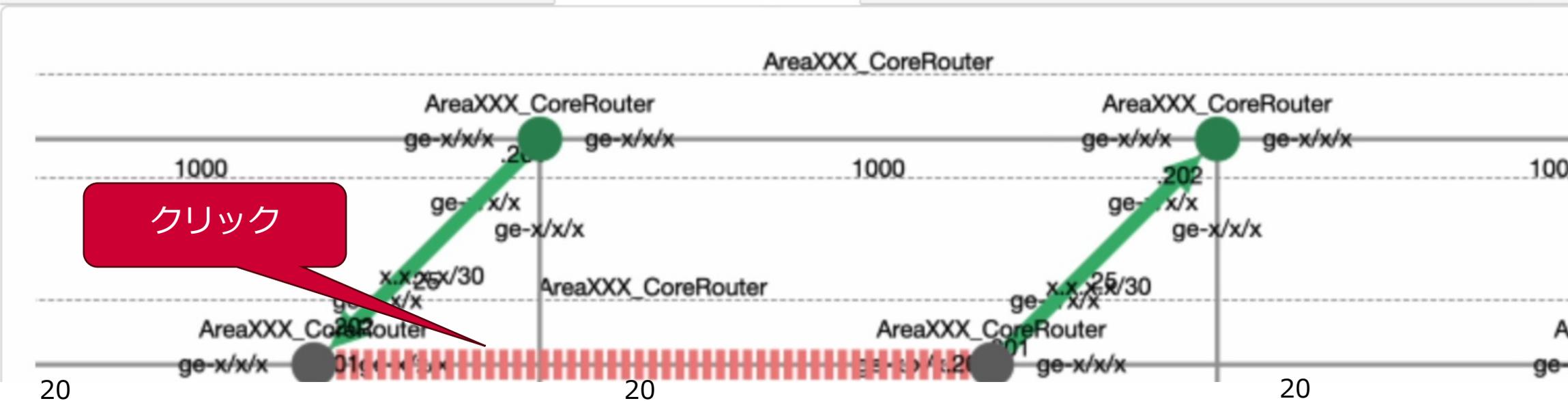
SPF再計算

SPF計算から除外（障害模擬）するパスかルータを選択し、再計算ボタン押下してください。

IPアドレスのみ

ポート名のみ

クリック



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

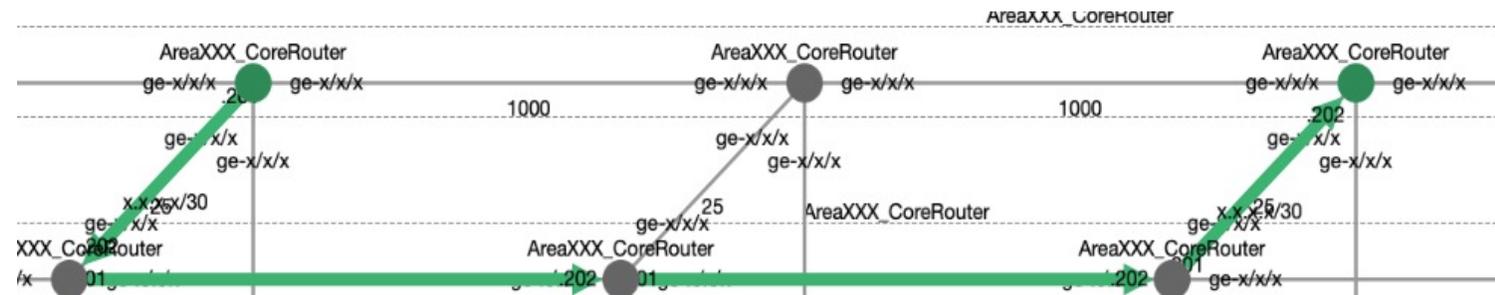
ツールの
貼り付け

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

上のホップから順に
アドレスをクリック



Graph theory (network) library for visualisation and analysis



Tracerouteログ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

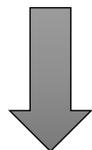
残り時間で話せるところまで…

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

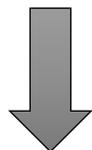
コンフィグ



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

```
config_parse > textfsm_template > conf_if.template
1 Value Name (\S+)
2 Value LogicalUnit (\d+)
3 Value Desc (\S+)
4 Value Addr (\S+)
5 Value Speed (\S+)
```

コンフィグ抽出の
テンプレート

```
11
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
```

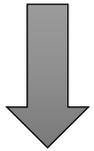
db.sqlite3 — conf_if 2331 rows, showing page 1

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

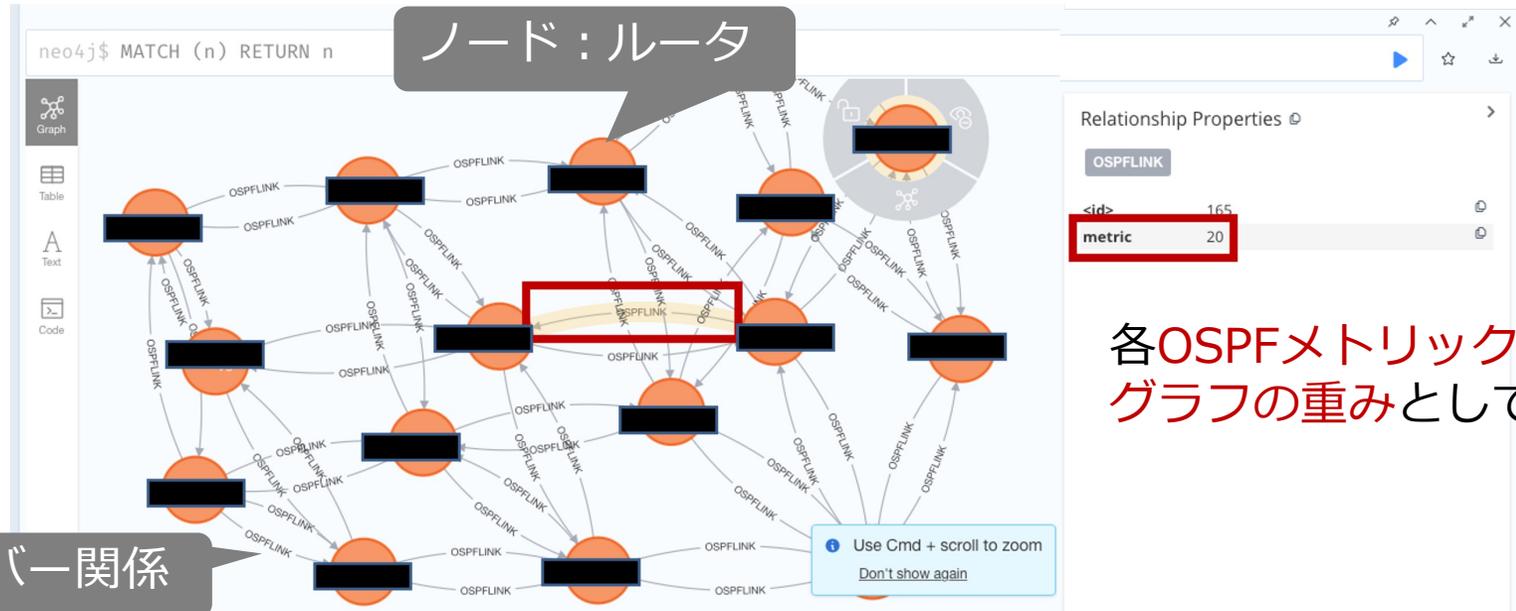


db.sqlite3 — conf_if 2331 rows, showing page 1

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

同セグなら対向ポート/OSPFネイバーと推定

ネットワークポロジをグラフデータとして登録



各OSPFメトリックをグラフの重みとして登録

エッジ: OSPFネイバー関係

Step3. グラフデータベースOSSで経路計算

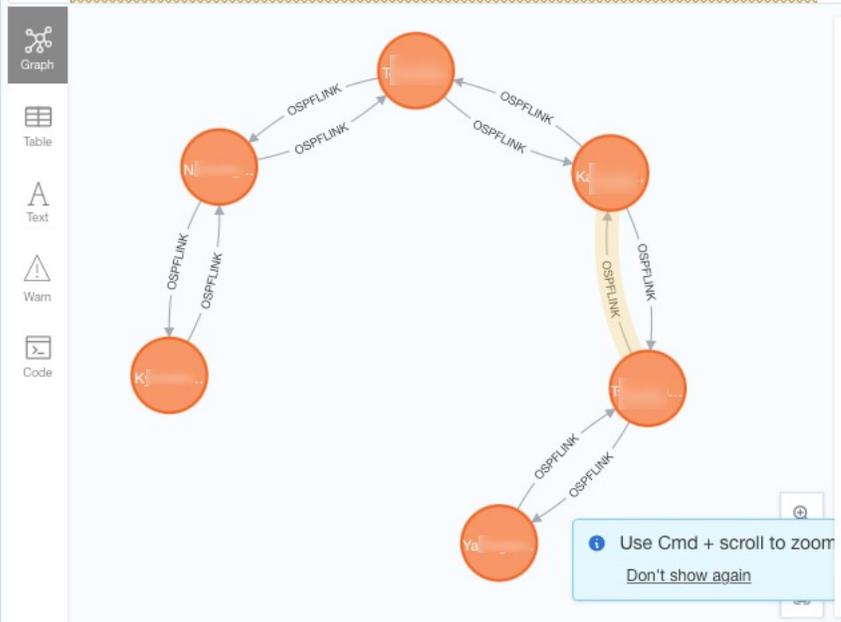
複数の冗長経路の中から、OSPFメトリックが最小となる経路を要求

```
neo4j$
1 //ShowCostShortestPath_GDS
2 MATCH (sourceR:Router {name: 'Y. [redacted] A'}), (targetR:Router {name:
  'K. [redacted] 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



グラフデータベースの最短経路計算機能を利用



Relationship Properties

OSPFLINK	
<id>	162
metric	20

Integer

各ホップのホスト名がJson形式で返ってくる

	"costs"	"path"
1, "Tok	[0.0, 25.0, 45.0, 65.0, 85.0, 110.0]	[{"name": "Y. [redacted] R_A"}, {"name": "T. [redacted] A"}, {"name": "K. [redacted] R_A"}, {"name": "I. [redacted] A"}, {"name": "N. [redacted] A"}, {"name": "K. [redacted] R_A"}]

Step4. 算出経路をグラフデータの可視化OSSで描画



計算した最短経路



変換

点と線のデータ



描画

トポロジ図
+ 経路図

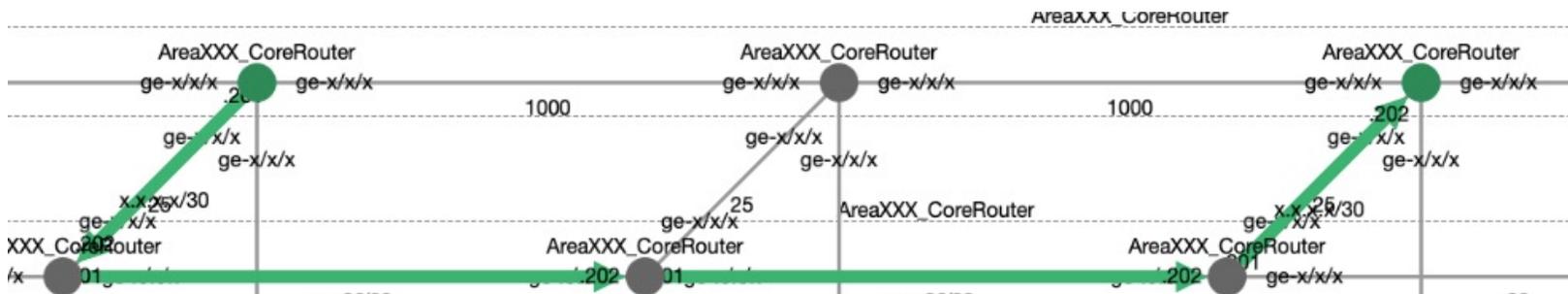
	"costs"	"path"
'', "Tok	[0.0, 25.0, 45.0, 65.0, 85.0, 110.0]	[{"name": "Y", "id": "AreaXX_CoreRouter_A"}, {"name": "Ts", "id": "AreaXX_CoreRouter_B"}, {"name": "Ka", "id": "AreaXX_CoreRouter_A"}, {"name": "I", "id": "AreaXX_CoreRouter_A"}, {"name": "N", "id": "AreaXX_CoreRouter_A"}, {"name": "K", "id": "AreaXX_CoreRouter_A"}]

```
{ "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)





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

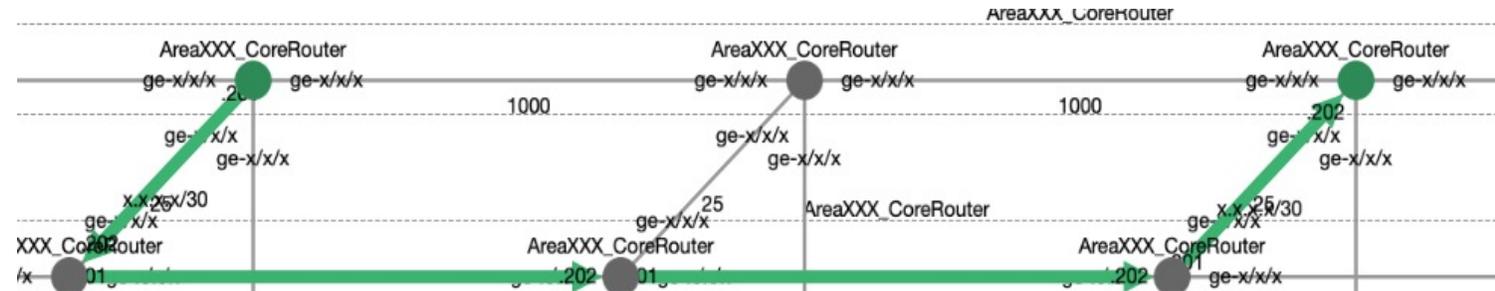
障害試験時のルータでのtraceroute出力例 (13ホップ)



step1のDBから対応ホスト名割り出し

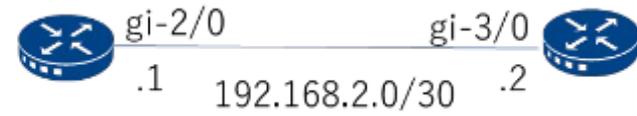
ttl	Label	Protocol	Address	Previous Hop	Probe Status
1	5928	RSVP-TE	192.168.x.x	(null)	Success
FEC-Stack-Sent: RSVP					
2	12692	RSVP-TE	192.168.x.x	2	Success
FEC-Stack-Sent: RSVP					
3	15866	RSVP-TE	192.168.x.x	1	Success
FEC-Stack-Sent: RSVP					

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

```
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

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

トポロジ推論

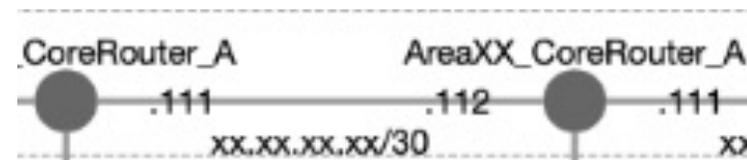
各ネットワークアドレス算出
同一セグメントが直結と推論

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

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

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

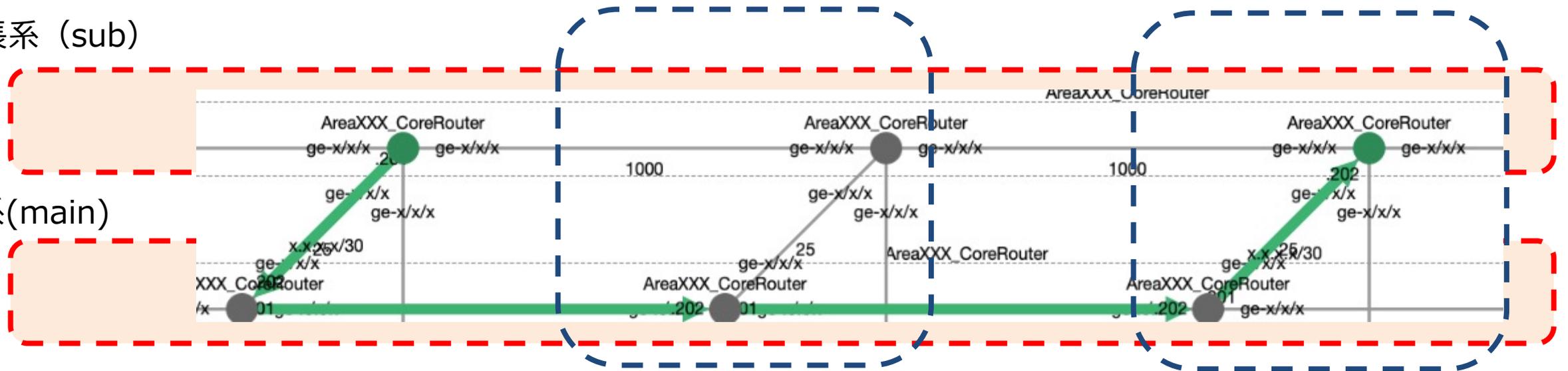
縦軸 y=1

冗長系 (sub)

主系(main)

東海地方

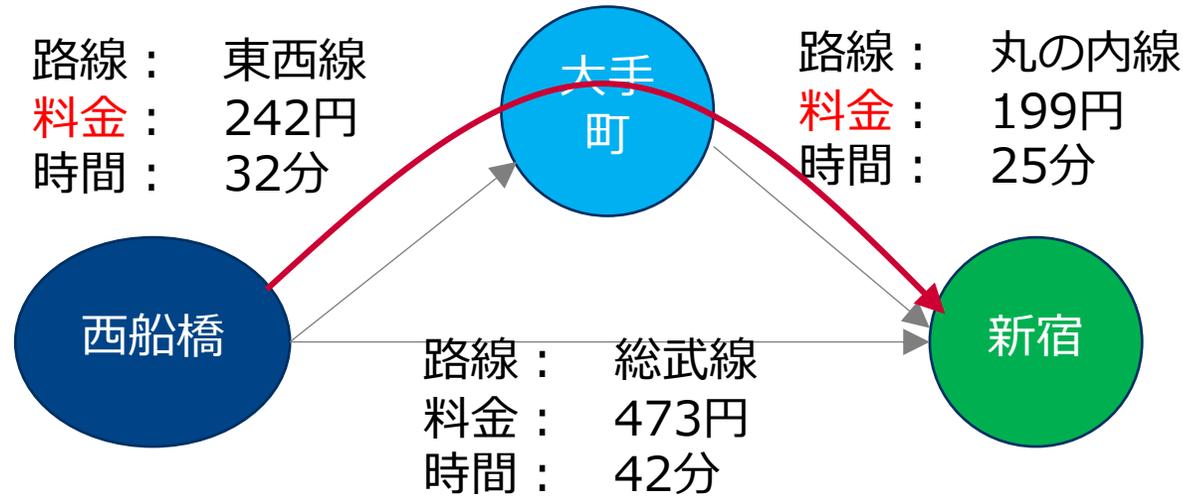
関東地方



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)`

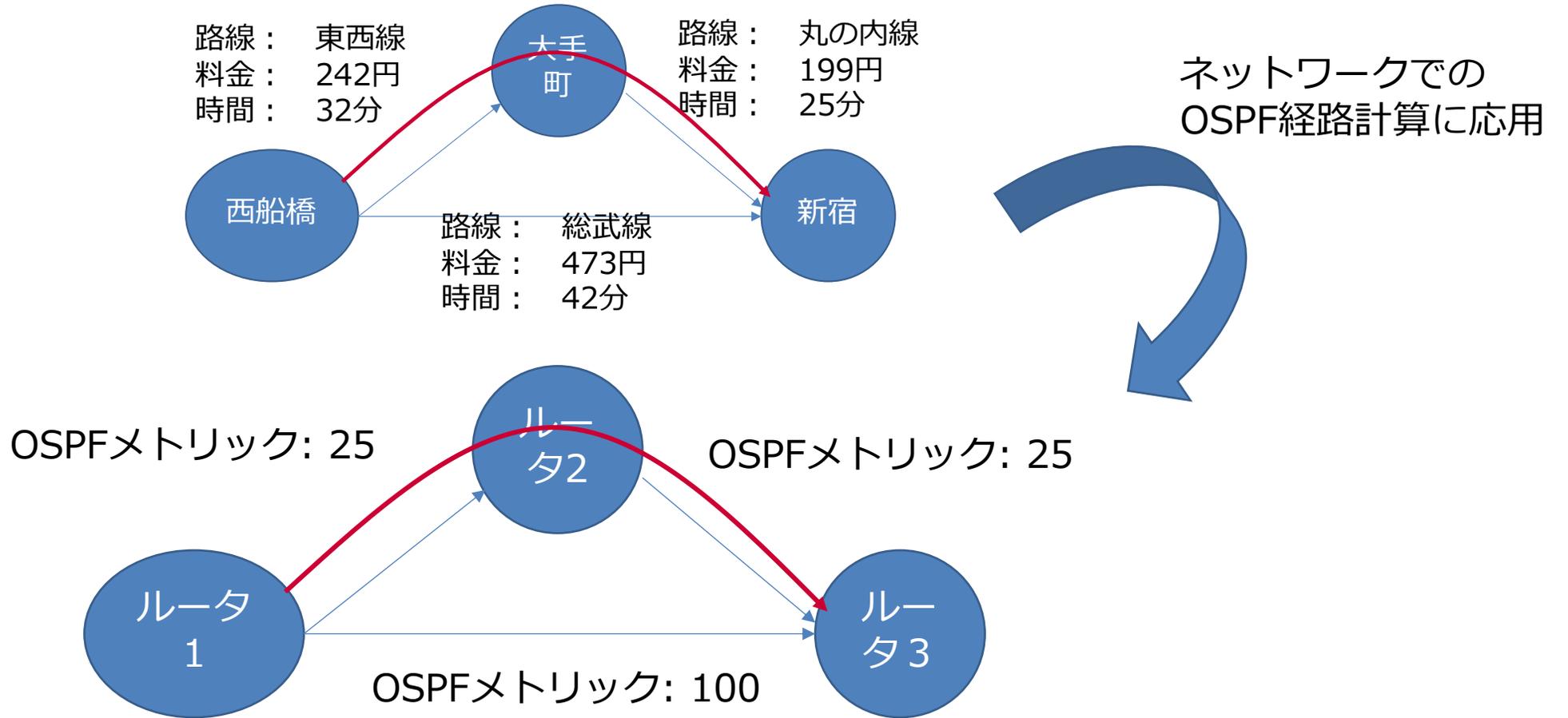
グラフデータベースへの経路検索クエリ (イメージ)

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

neo4j\$ MATCH (n) RETURN n

Relationship Properties

OSPFLINK	
<id>	165
metric	20

とあるルータ間の
リンクの
OSPFメトリック

Use Cmd + s
Don't show ag