Service Aware Networking using SR

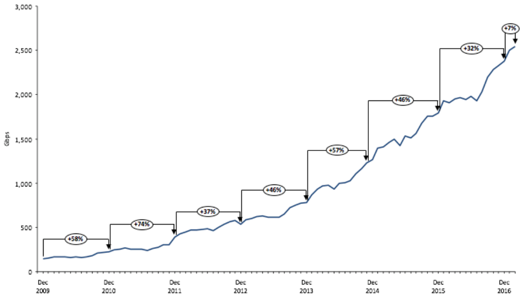
IETF101 PANRG meeting – March 20th 2018

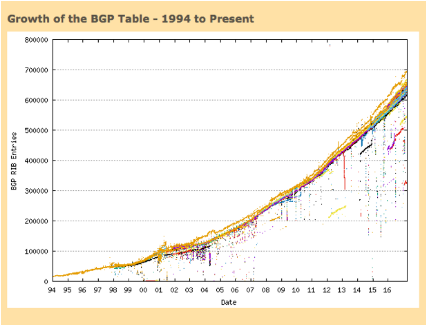
daniel.bernier@bell.ca
daniel.voyer@bell.ca

Telco Challenges

The Internet is growing … exponentially !!

- Physical networks are static.
- Long term migration cycles.
- Faster than we can adapt … or pay for.



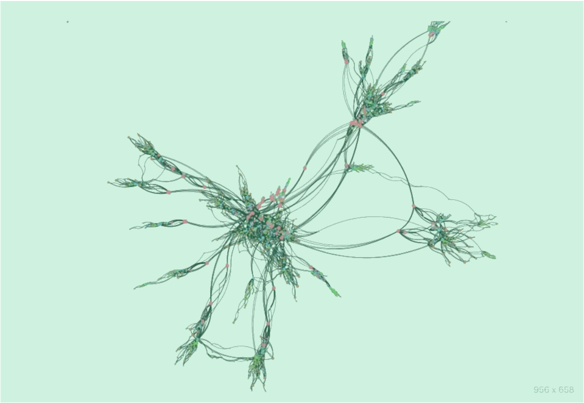


source https://bgp.potaroo.net

Legacy HW become obsolete quicker

- 2014 TCAM scale issue … widespread outage.
- Costing us more $$$

Managing our own complexity through mergers and acquisitions



No E2E Traffic Engineering

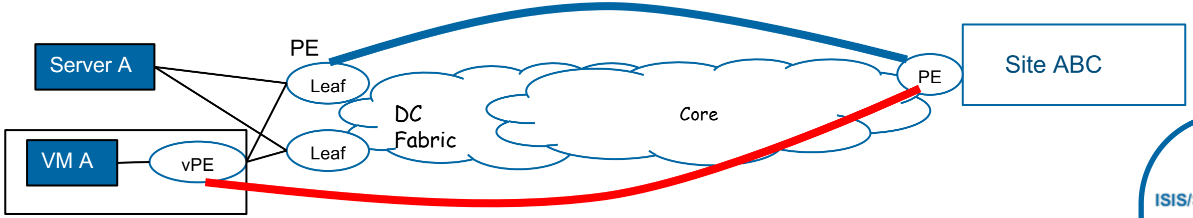
- Keeps complex state in the network
- Static and hardcoded … always on

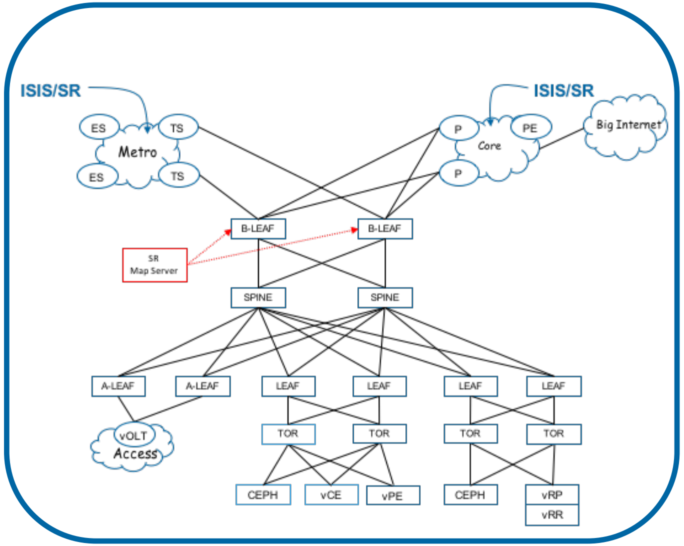
No E2E OAM

- Reduced to hop-by-hop troubleshooting
- Poor visibility on the status of TE tunnels

Network Transformation – First Response

- The goal for network transformation (Bell Network 3.0) is to move the complexity from core transport to the CO/DC and virtualize network components.
  - Leverage existing Data Plane – MPLS
  - With a vision towards IPv6
  - Simplify Control Plane – SR fabric in DC





- DC fabric and core network seen as a common IP network

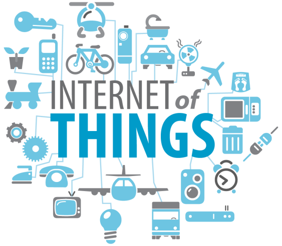
- CO/DC Challenges Solved by SR
  - Classic DCI overlay is wrong for ”inline” CO/DC, we need a seamless integration to leverage network assets.

Reference: MPLSWC 2017 – Daniel Voyer CO/DC Network Transformation
https://www.youtube.com/watch?v=66M8ipFaTeM

Telco Challenges … Round Two

- Not that easy to convert 70+ year old central offices into data centers.
  - Limited Power Availability
  - Limited Cooling
  - Low Margin for investment in retrofitting
  - Example: 1 DMS = 1 Compute Rack … *But it takes a year to decommission !*
- Virtualizing the network is NOT easy
  - VNFs are not always playing nice.
  - Virtualizing **CORE** network functions into Openstack is not trivial.
- Specialized hardware (FPGAs, GPUs, etc.) are becoming the new norm.

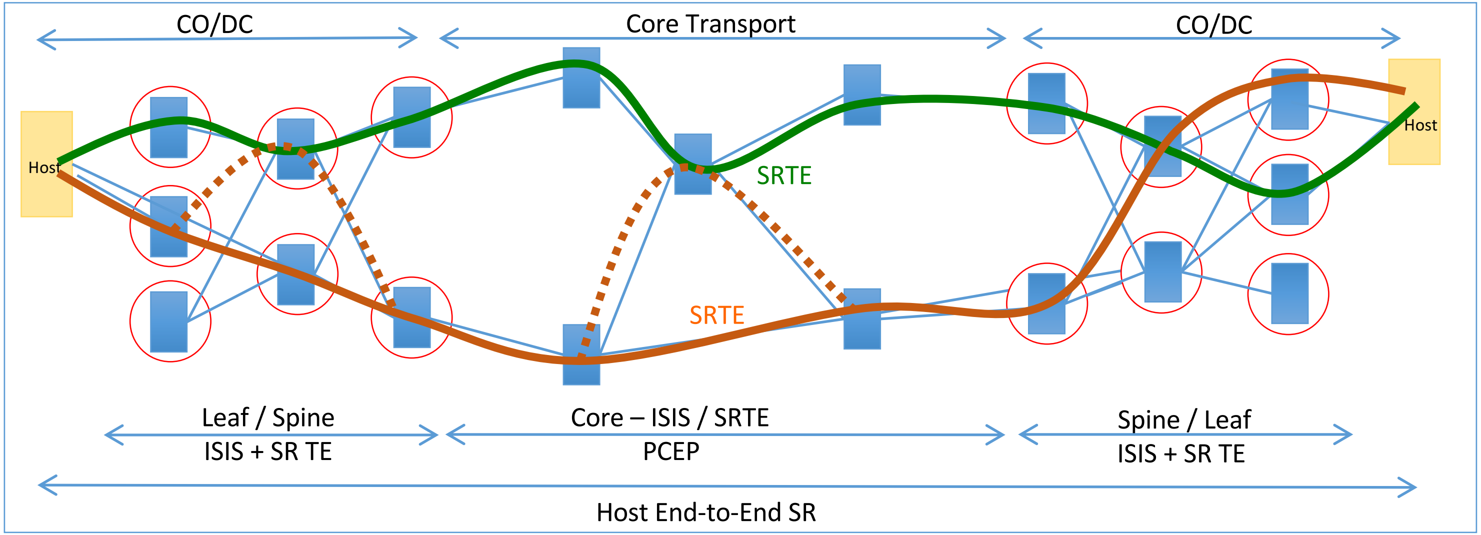
By the Time we Get it Right, Industry has moved On



An estimated 50 Billion devices will be connected by 2020. Interconnecting and securing these at scale cannot use our current network toolkit.

Network Transformation – Pushing it Further

- SR between and within DCs
- Extend SR to the host (hypervisor, kernel, VNF) → Moving the Edge Further
- Leverage SRTE
  - On-Demand SRTE
  - On-Demand Next-Hop

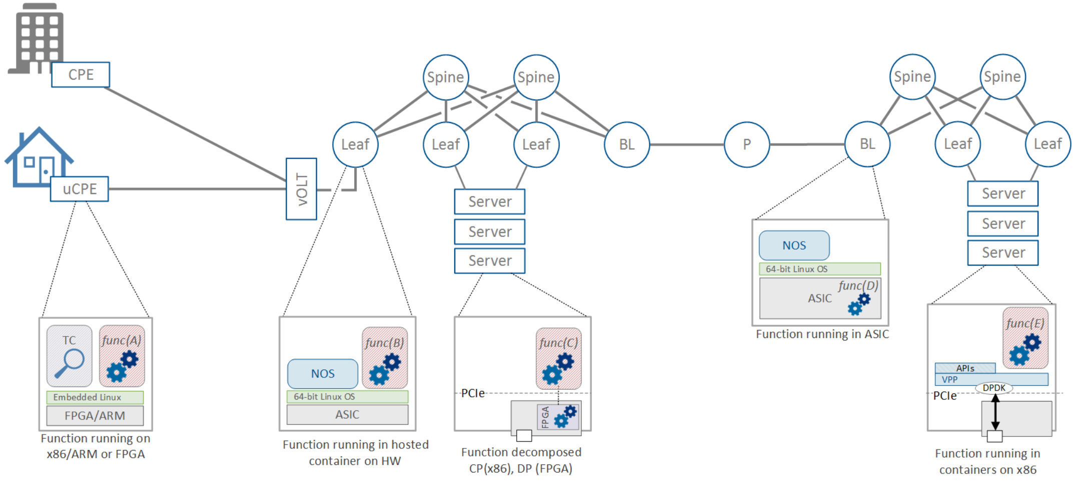


One step closer to Application(Service) aware networking

But Why ?

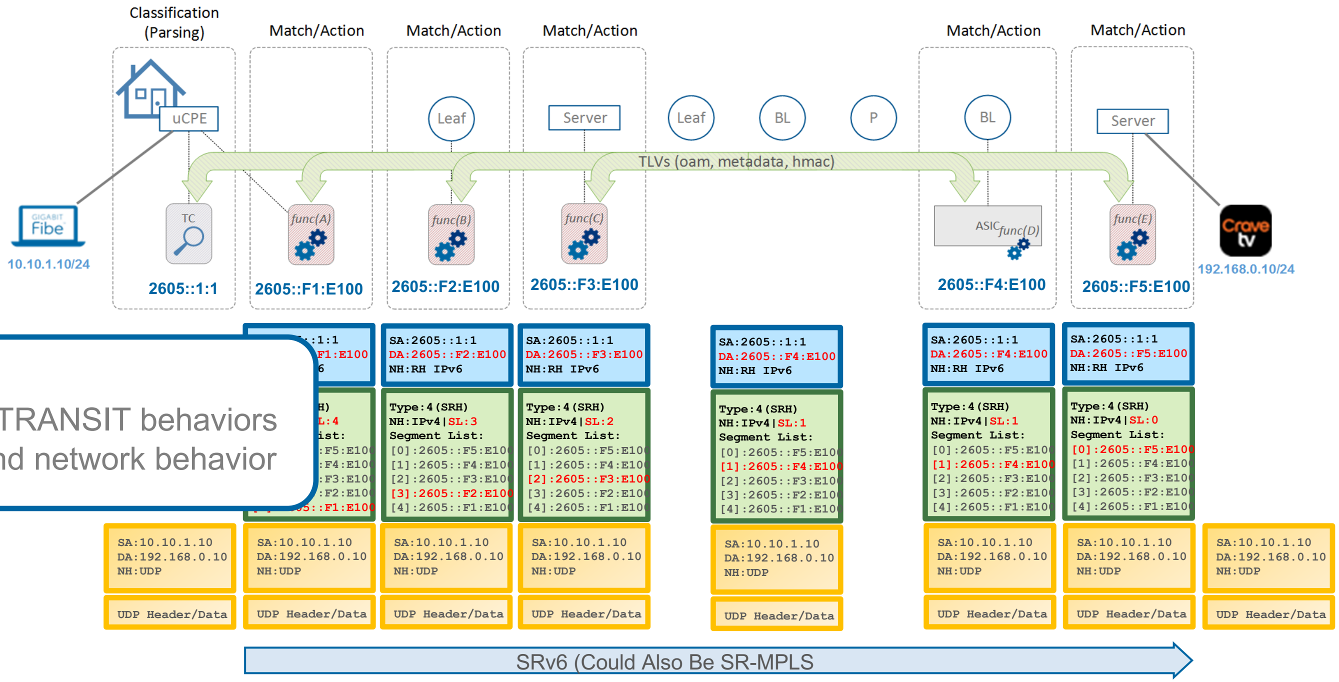
- Make the underlay **stateless**
  - Avoid tunnels and « decap/encap » middle boxes
- **Distribute** function processing
  - Push state to the edges
  - 100s instances of a function scales better than a few big ones … *but you NEED automation* !
- Abstract complexity of network constructs through policies
  - App owners do not need to know the subtleties of the network.
- Service chains are now a more specialized set of segments in a **Network Policy**

Network Policies are made of
- Service Functions
- TE Behaviors
- … SRv6 END functions



A View on ”Path Aware” Network Policy

- Traffic classification/identification at the edge of the network →Parsing
- Simplified *Match/Action* primitive looking at the function Identifier → Distributed Processing
- Based on Applications Requirements → Application Aware



SA:10.10.1.10
DA:192.168.0.10
NH:UDP

UDP Header/Data

::1:1
F1:E100
6

H)
L:4
ist:
:F5:E100
:F4:E100
:F3:E100
:F2:E100
05::F1:E100

SA:10.10.1.10
DA:192.168.0.10
NH:UDP

UDP Header/Data

SA:2605::1:1
DA:2605::F2:E100
NH:RH IPv6

Type:4(SRH)
NH:IPv4|SL:3
Segment List:
[0]:2605::F5:E100
[1]:2605::F4:E100
[2]:2605::F3:E100
[3]:2605::F2:E100
[4]:2605::F1:E100

SA:10.10.1.10
DA:192.168.0.10
NH:UDP

UDP Header/Data

SA:2605::1:1
DA:2605::F3:E100
NH:RH IPv6

Type:4(SRH)
NH:IPv4|SL:2
Segment List:
[0]:2605::F5:E100
[1]:2605::F4:E100
[2]:2605::F3:E100
[3]:2605::F2:E100
[4]:2605::F1:E100

SA:10.10.1.10
DA:192.168.0.10
NH:UDP

UDP Header/Data

SA:2605::1:1
DA:2605::F4:E100
NH:RH IPv6

Type:4(SRH)
NH:IPv4|SL:1
Segment List:
[0]:2605::F5:E100
[1]:2605::F4:E100
[2]:2605::F3:E100
[3]:2605::F2:E100
[4]:2605::F1:E100

SA:10.10.1.10
DA:192.168.0.10
NH:UDP

UDP Header/Data

SA:2605::1:1
DA:2605::F4:E100
NH:RH IPv6

Type:4(SRH)
NH:IPv4|SL:1
Segment List:
[0]:2605::F5:E100
[1]:2605::F4:E100
[2]:2605::F3:E100
[3]:2605::F2:E100
[4]:2605::F1:E100

SA:10.10.1.10
DA:192.168.0.10
NH:UDP

UDP Header/Data

SA:2605::1:1
DA:2605::F5:E100
NH:RH IPv6

Type:4(SRH)
NH:IPv4|SL:0
Segment List:
[0]:2605::F5:E100
[1]:2605::F4:E100
[2]:2605::F3:E100
[3]:2605::F2:E100
[4]:2605::F1:E100

SA:10.10.1.10
DA:192.168.0.10
NH:UDP

UDP Header/Data

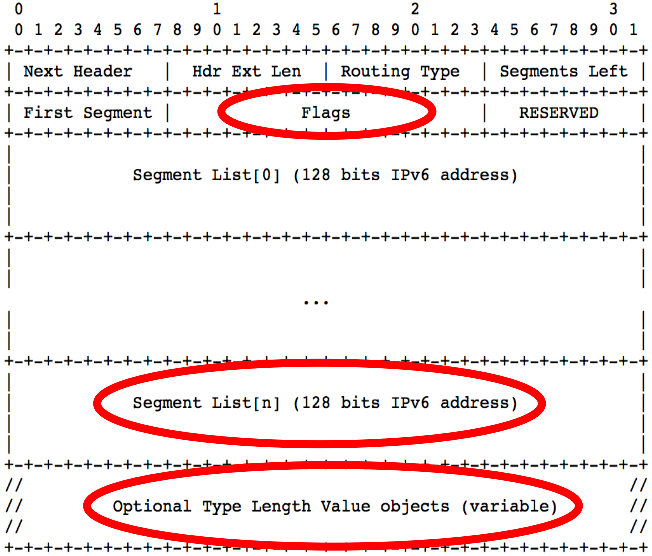
SA:10.10.1.10
DA:192.168.0.10
NH:UDP

UDP Header/Data

Programming at all Layers

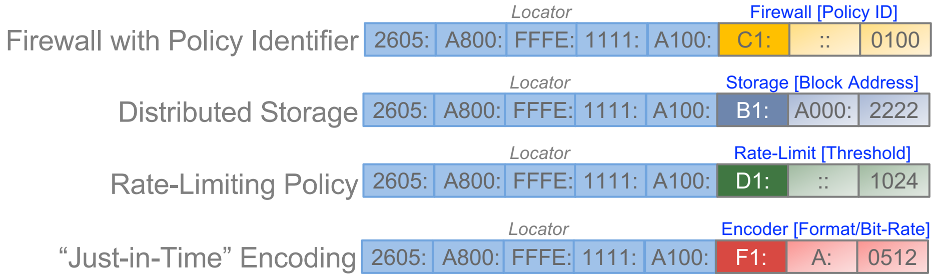
- Local SIDs define END and TRANSIT behaviors
- SRTE defines the End-To-End network behavior

One Way To Express Network Behavior



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Header   |  Hdr Ext Len  | Routing Type  | Segments Left |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| First Segment |            Flags              |   RESERVED    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|            Segment List[0] (128 bits IPv6 address)            |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
                              ...
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|            Segment List[n] (128 bits IPv6 address)            |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
//                                                             //
//         Optional Type Length Value objects (variable)       //
//                                                             //
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- ○ Leverage Flags for simple classification/policing
- ○ Leverage ”Locator/function/argument” structure of the SID to embed function encoding
- ○ Leverage TLVs for more complex metadata

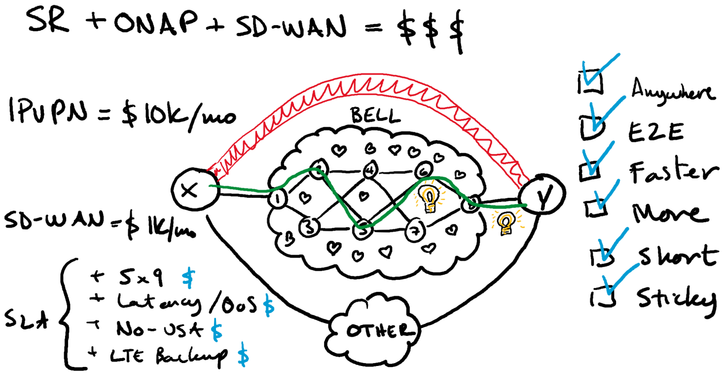


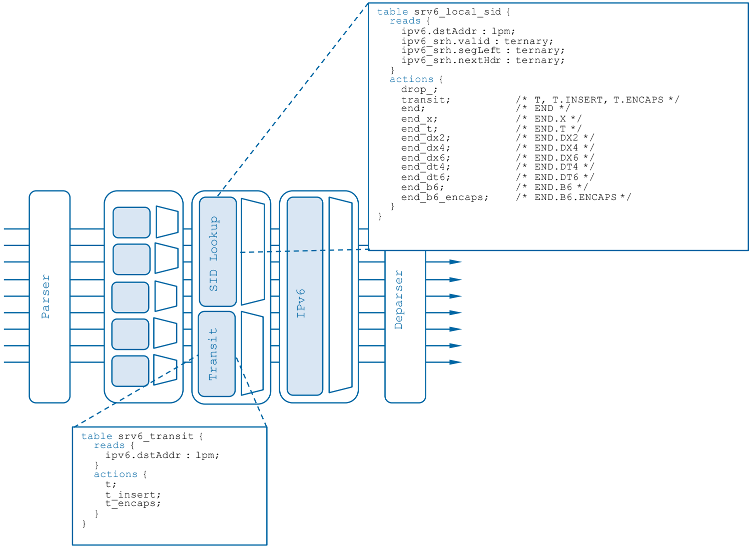
	Locator					Firewall [Policy ID]		
Firewall with Policy Identifier	2605:	A800:	FFFE:	1111:	A100:	C1:	::	0100
	Locator					Storage [Block Address]		
Distributed Storage	2605:	A800:	FFFE:	1111:	A100:	B1:	A000:	2222
	Locator					Rate-Limit [Threshold]		
Rate-Limiting Policy	2605:	A800:	FFFE:	1111:	A100:	D1:	::	1024
	Locator					Encoder [Format/Bit-Rate]		
“Just-in-Time” Encoding	2605:	A800:	FFFE:	1111:	A100:	F1:	A:	0512

But how do we program these behaviors on the host ?

Implementing SR on the Host

- Lots of available implementations
  - SRH in Linux Kernel
  - SR-MPLS/SRv6 in VPP
  - P4 for hardware programming (Tofino, FPGAs).
- Work needed to implement in cloud platforms
  - Adding SR data plane in VIMs (Openstack, etc.)
- Work needed on End-to-End orchestration





```
table srv6_local_sid {
  reads {
    ipv6.dstAddr : lpm;
    ipv6_srh.valid : ternary;
    ipv6_srh.segLeft : ternary;
    ipv6_srh.nextHdr : ternary;
  }
  actions {
    drop_;
    transit;           /* T, T.INSERT, T.ENCAPS */
    end;               /* END */
    end_x;             /* END.X */
    end_t;             /* END.T */
    end_dx2;           /* END.DX2 */
    end_dx4;           /* END.DX4 */
    end_dx6;           /* END.DX6 */
    end_dt4;           /* END.DT4 */
    end_dt6;           /* END.DT6 */
    end_b6;            /* END.B6 */
    end_b6_encaps;     /* END.B6.ENCAPS */
  }
}
```

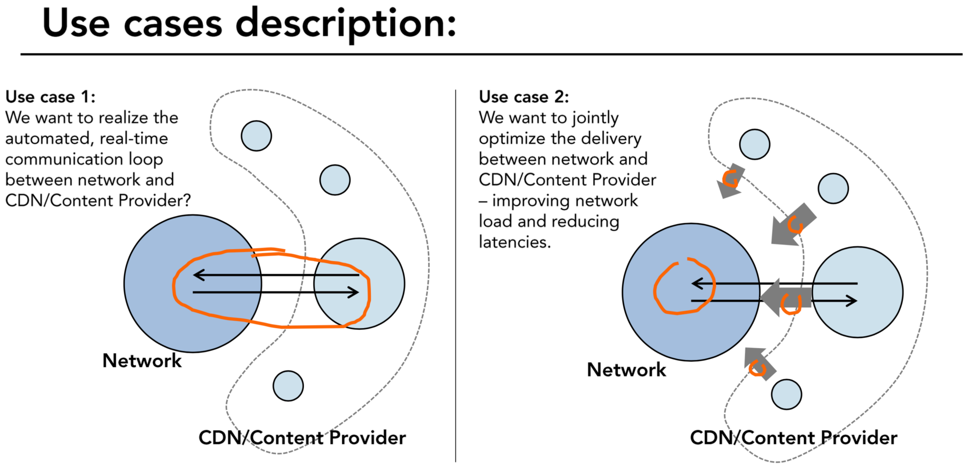
```
table srv6_transit {
  reads {
    ipv6.dstAddr : lpm;
  }
  actions {
    t;
    t_insert;
    t_encaps;
  }
}
```

Now we know how to program a SR on a host … But ?

Our Challenge on Path Awareness

- We Are Getting Good At Telling the Edge How to Talk with the Network
  - From BGP extended communities to Binding SID and PvDs.
  - From SRTE in the control-plane to configuration via management plane (NC/gRPC).

- There is also progress on how to exchange network behaviors between providers
  - Use of ALTO to exchange network capabilities and requirements (https://telecominfraproject.facebook.com/notes/tip-greenfield-networks-app-aware-networking/application-aware-networking-a-first-step-towards-intent-based-networking/1941364519455351/)



Use cases description:

Use case 1: We want to realize the automated, real-time communication loop between network and CDN/Content Provider?

Use case 2: We want to jointly optimize the delivery between network and CDN/Content Provider – improving network load and reducing latencies.

But how can an app or a host ask for a policy ?

How about a ”Path Awareness” Intent ?

- There is a need for a simple mechanism to ask (express) a network path based on need.
  - From BGP extended communities to Binding SID and PvDs.
  - From SRTE in the control-plane to configuration via management plane (NC/gRPC).

- But we need to **abstract network complexity** from the app owners … and make it automated.

- Some have proposed ideas.
  - D. Lebrun, M. Jadin, F. Clad, C. Filsfils, O. Bonaventure, Software Resolved Networks: Rethinking Enterprise Networks with IPv6 Segment Routing, ACM SOSR 2018 https://conferences.sigcomm.org/sosr/2018/sosr18-finals/sosr18-final15.pdf

Software Resolved Networks

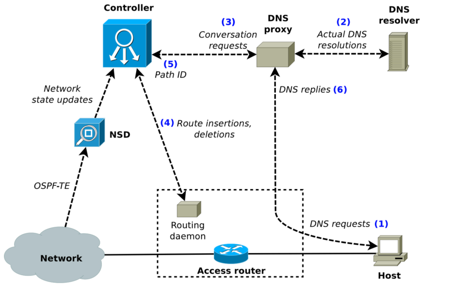


Figure 3: Illustration of the components of a SRN. The figure shows the exchanges involved in a conversation request.

We Need To Put More Focus On This

Thank You