The Extensible Network
Evolution in Protocol and Data Plane Agility

Daniel Bernier, *Bell Canada*; Milad Sharif, *Barefoot Networks*; Clarence Filsfils*, Cisco Systems*
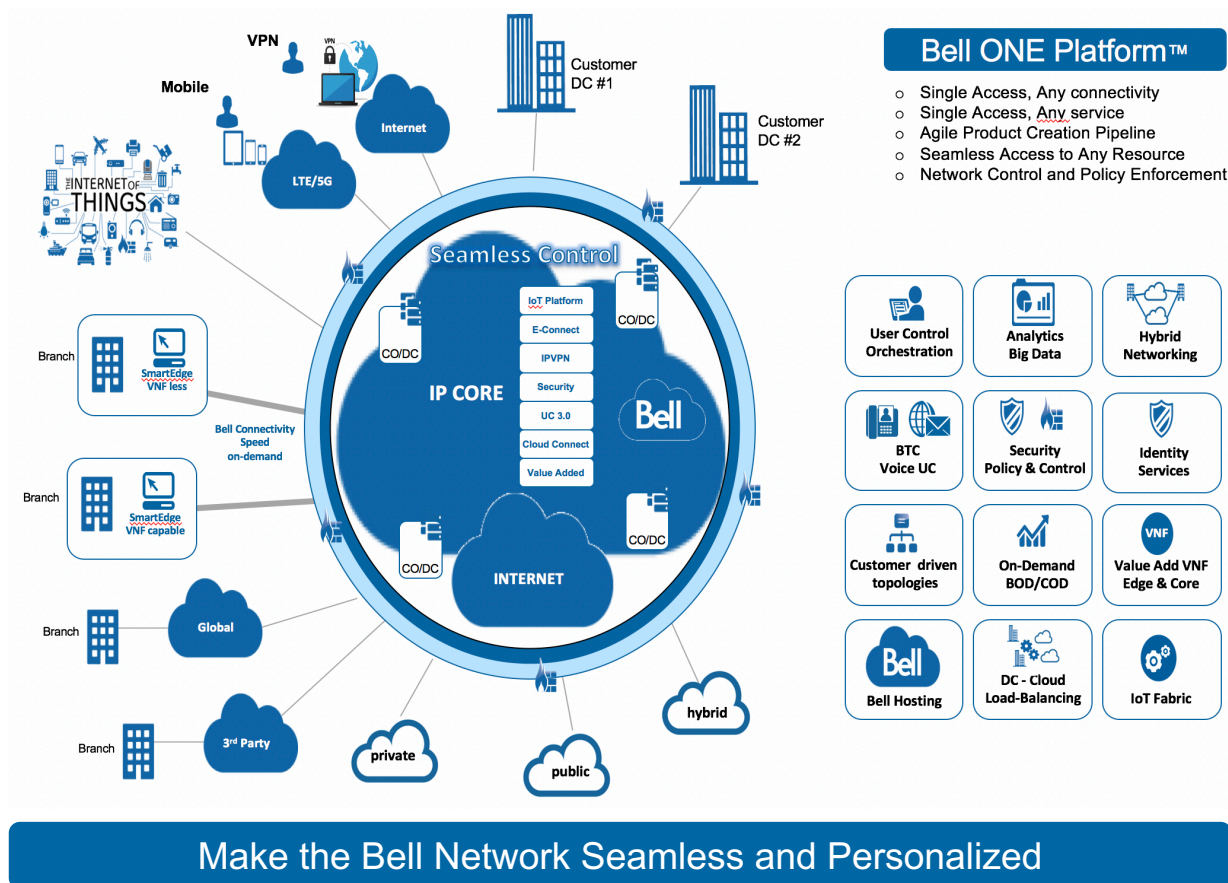
A Bit of Context

With the **Network 3.0** initiative, Bell Canada is evolving toward heterogeneous, highly interoperable platform that supports an array of relevant business and residential services.

- o Large investment in fiber (FTTH, FTTN, etc.) to support future requirements and exponential bandwidth growth.
- o Massive network simplification, automation and virtualisation program.
  - Fueled by it's CO/DC transformation (https://www.youtube.com/watch?v=66M8ipFaTeM)
- o Disaggregation of connectivity and value-add services from the underlying physical network.

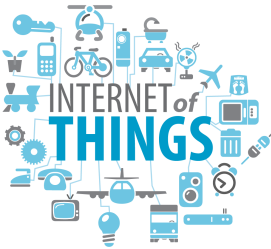**Make the Bell Network Seamless and Personalized**

But … "*Winter is Coming*"

> We're gonna need a bigger boat!
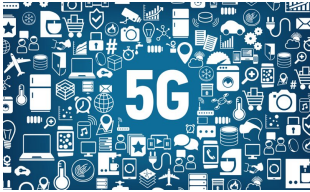
5G will tax the network from multiple ends. From low latency and edge proximity to high capacity and flexible orchestration.
The old "throw bandwidth at it" will not suffice.

An estimated 50 Billion devices will be connected by 2020. Interconnecting and securing these at scale cannot use our current network toolkit.
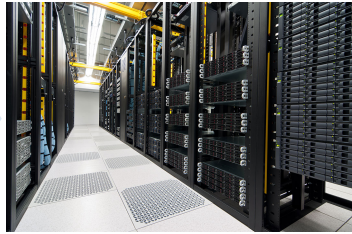
Infrastructure challenges associated to converting 60+ year old central offices to data centers, using commodity hardware has limited the ability for Telco's to leverage proximity to the end users. We are now seeing a return to specialized hardware (FPGAs, GPUs, etc.).

While the industry is still working to virtualize the network efficiently, only few VNFs have moved to micro-services and yet cloud as already moved further (e.g. *FaaS*, *AWS F1*, etc.).

How Do We Rethink our Network ?

Bell

Scaling Things Out

o **Make the underlying network stateless**
  - Push state to the edges
  - Simplify the protocol soup.

o **Distribute functions where they make most sense**
  - Functions can be placed anywhere … from network elements to the cloud.
  - That's where a common language for multiple targets comes in handy.

o **Distribute function processing**
  - 100s of distributed functions will scale better then a few big ones.

o **Leverage abstracted function identifiers**
  - Make them referenceable and potentially supporting resolution

Function running on x86/ARM or FPGA

Function running in hosted container on HW

Function decomposed CP(x86), DP (FPGA)

Function running in ASIC

Function running in containers on x86
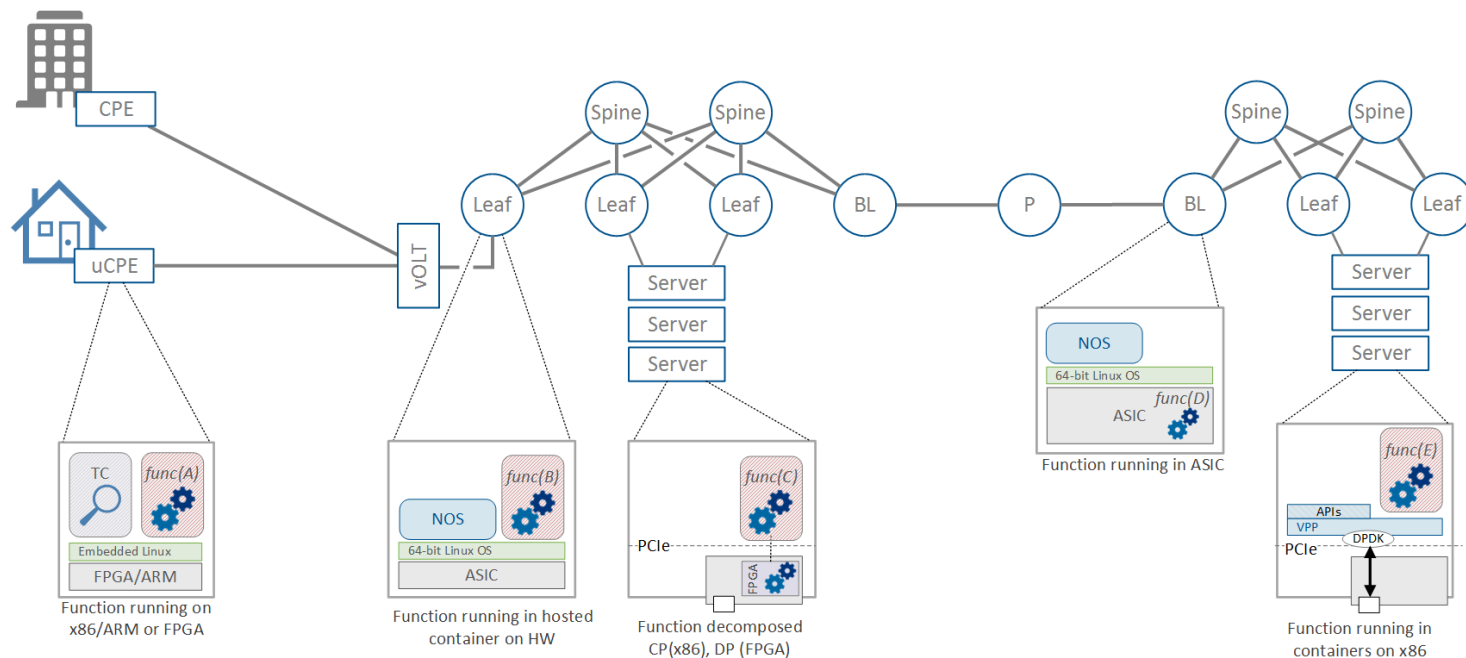
The "*Network-as-an-ASIC*"

o Traffic classification at the edge of the network → e.g. parsing.
o Simplified *Match/Action* primitive looking at the function Identifier.
o Contextual metadata carried through TLVs
o Programming at All Layers
  • P4 to define the END and TRANSIT behaviors in data plane.
  • SRv6 to define the "*end to end network behavior*"

Classification (Parsing) | Match/Action | Match/Action | Match/Action | | | | Match/Action | Match/Action

uCPE — Leaf — Server — Leaf — BL — P — BL — Server

TLVs (oam, metadata, hmac)

10.10.1.10/24

TC — func(A) — func(B) — func(C) — ASIC func(D) — func(E) — 192.168.0.10/24

2605::1:1 2605::F1:E100 2605::F2:E100 2605::F3:E100 2605::F4:E100 2605::F5:E100

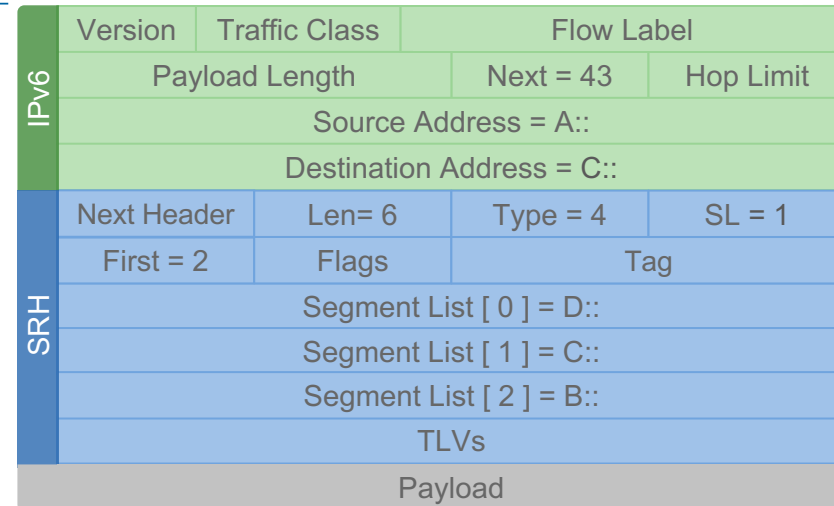|					
SA:2605::1:1 DA:2605::F1:E100 NH:RH IPv6	SA:2605::1:1 DA:2605::F2:E100 NH:RH IPv6	SA:2605::1:1 DA:2605::F3:E100 NH:RH IPv6	SA:2605::1:1 DA:2605::F4:E100 NH:RH IPv6	SA:2605::1:1 DA:2605::F4:E100 NH:RH IPv6	SA:2605::1:1 DA:2605::F5:E100 NH:RH IPv6
Type:4(SRH) NH:IPv4|SL:4 Segment List: [0]:2605::F5:E100 [1]:2605::F4:E100 [2]:2605::F3:E100 [3]:2605::F2:E100 [4]:2605::F1:E100	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	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	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	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	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	SA:10.10.1.10 DA:192.168.0.10 NH:UDP	SA:10.10.1.10 DA:192.168.0.10 NH:UDP	SA:10.10.1.10 DA:192.168.0.10 NH:UDP	SA:10.10.1.10 DA:192.168.0.10 NH:UDP	SA:10.10.1.10 DA:192.168.0.10 NH:UDP
UDP Header/Data	UDP Header/Data	UDP Header/Data	UDP Header/Data	UDP Header/Data	UDP Header/Data

(Additional packet columns also present: SA:2605::1:1 DA:2605::F4:E100 NH:RH IPv6 / SA:10.10.1.10 DA:192.168.0.10 NH:UDP / UDP Header/Data; and final Server column SA:10.10.1.10 DA:192.168.0.10 NH:UDP / UDP Header/Data)

SRv6

Bell

How Can This Be Achieved ?

Bell

A Bit of SRv6 Theory

https://tools.ietf.org/html/draft-filsfils-spring-srv6-network-programming-00
https://tools.ietf.org/html/draft-ietf-6man-segment-routing-header-06
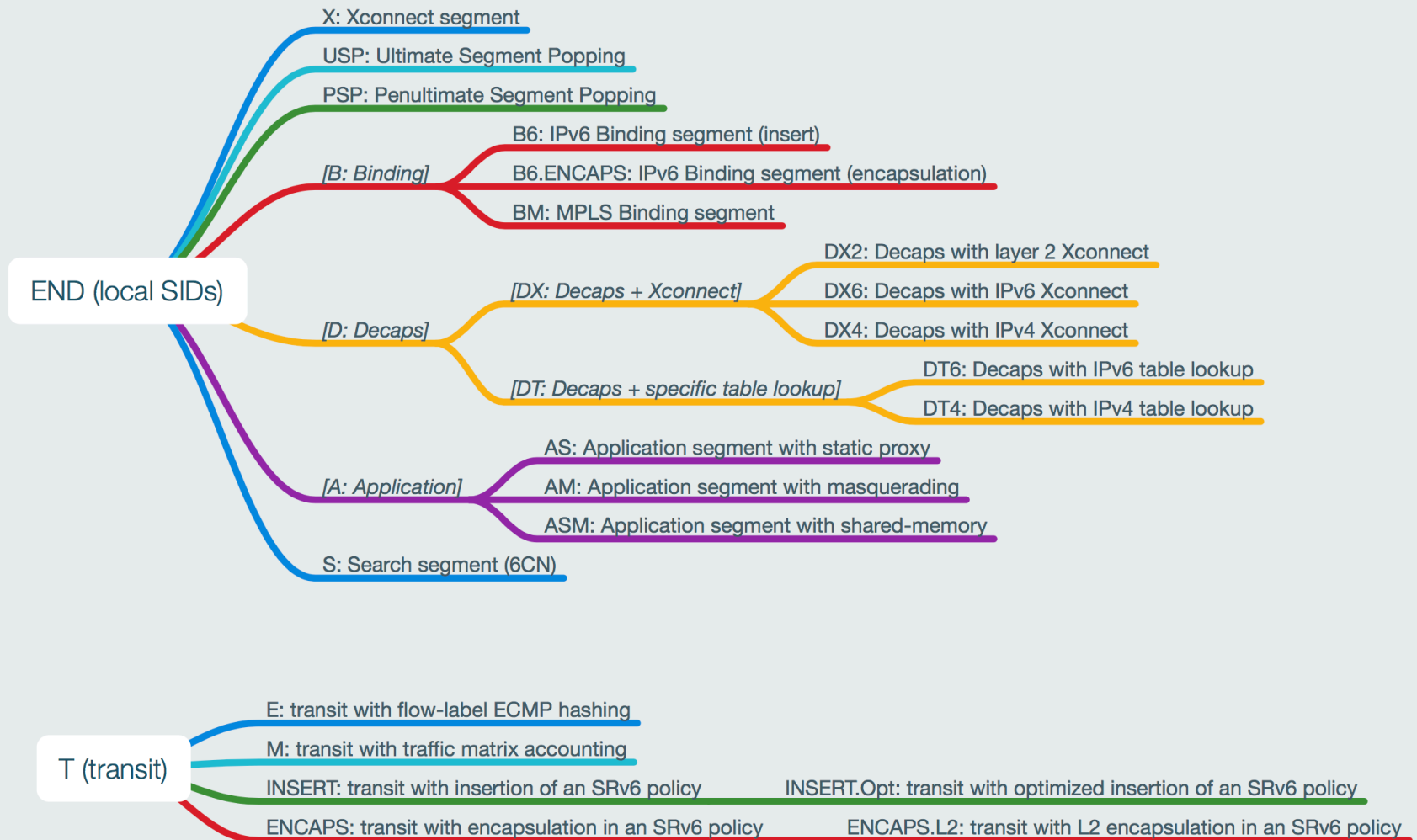
o **Each segment is an IPv6 Address**

o **The SID list is encoded in reverse order.**
  - **Last segment** in the list as an index value of 0.
  - **First Segment** indexes the first segment in the list.
  - **Segments Left** refers to the active segment index.

o **The *Destination Address* of the IPv6 header is copied from the *Active Segment* in the SID list.**

o **TLVs can be used to convey contextual information.**
  - HMAC
  - Metadata
  - OAM values (e.g. INT)

o **SRv6 SIDs are 128-bit addresses**
  - **Locator**: most significant bits are used to route the segments to its parent node
  - **Function**: least significant bits identify the action to be performed on the parent node
  - **Argument**:[optional] last bits can be used as a local function argument

o **Specific SID formatting only needs to be understood by the parent node**

o **SIDs have to be specifically enabled as such on their parent node**
  - A IPv6 address is not necessarily a local SID
  - A local SID is not necessarily bound to an interface.

IPv6	Version	Traffic Class		Flow Label	
	Payload Length		Next = 43		Hop Limit
	Source Address = A::				
	Destination Address = C::				
SRH	Next Header	Len= 6	Type = 4		SL = 1
	First = 2	Flags		Tag	
	Segment List [0] = D::				
	Segment List [1] = C::				
	Segment List [2] = B::				
	TLVs				
	Payload				

	Locator				*Function* [Arguments]		
2605:	A800:	FFFE:	1111:	A100:	C1:	0010:	2222

SRv6 SIDs as Universal Identifiers

Bell

Predefined END and TRANSIT Behaviors

END (local SIDs)
- X: Xconnect segment
- USP: Ultimate Segment Popping
- PSP: Penultimate Segment Popping
- [B: Binding]
  - B6: IPv6 Binding segment (insert)
  - B6.ENCAPS: IPv6 Binding segment (encapsulation)
  - BM: MPLS Binding segment
- [D: Decaps]
  - [DX: Decaps + Xconnect]
    - DX2: Decaps with layer 2 Xconnect
    - DX6: Decaps with IPv6 Xconnect
    - DX4: Decaps with IPv4 Xconnect
  - [DT: Decaps + specific table lookup]
    - DT6: Decaps with IPv6 table lookup
    - DT4: Decaps with IPv4 table lookup
- [A: Application]
  - AS: Application segment with static proxy
  - AM: Application segment with masquerading
  - ASM: Application segment with shared-memory
- S: Search segment (6CN)

T (transit)
- E: transit with flow-label ECMP hashing
- M: transit with traffic matrix accounting
- INSERT: transit with insertion of an SRv6 policy   INSERT.Opt: transit with optimized insertion of an SRv6 policy
- ENCAPS: transit with encapsulation in an SRv6 policy   ENCAPS.L2: transit with L2 encapsulation in an SRv6 policy

Bell

Key Enablers - Software Data Plane (VPP)

o **Fully open programmable data plane implementation using directed graphs**

o **Plugin architecture allows for deviation from core implementation**
  - Hardware Offload or Augmentation (e.g. SmartNICs)
  - Capabilities add-on

o **SRv6 already Implemented (release 17.04)**
  - Sample LocalSID plugin to create new functions

```
set sr encaps source addr A1::
sr policy add bsid B::999:1 next B2:: next D3::2 encap
sr steer l2 GigabitEthernet0/4/0 via sr policy bsid B::999:1
sr localsid address A1::2 behavior end.dx2 GigabitEthernet0/4/0
```
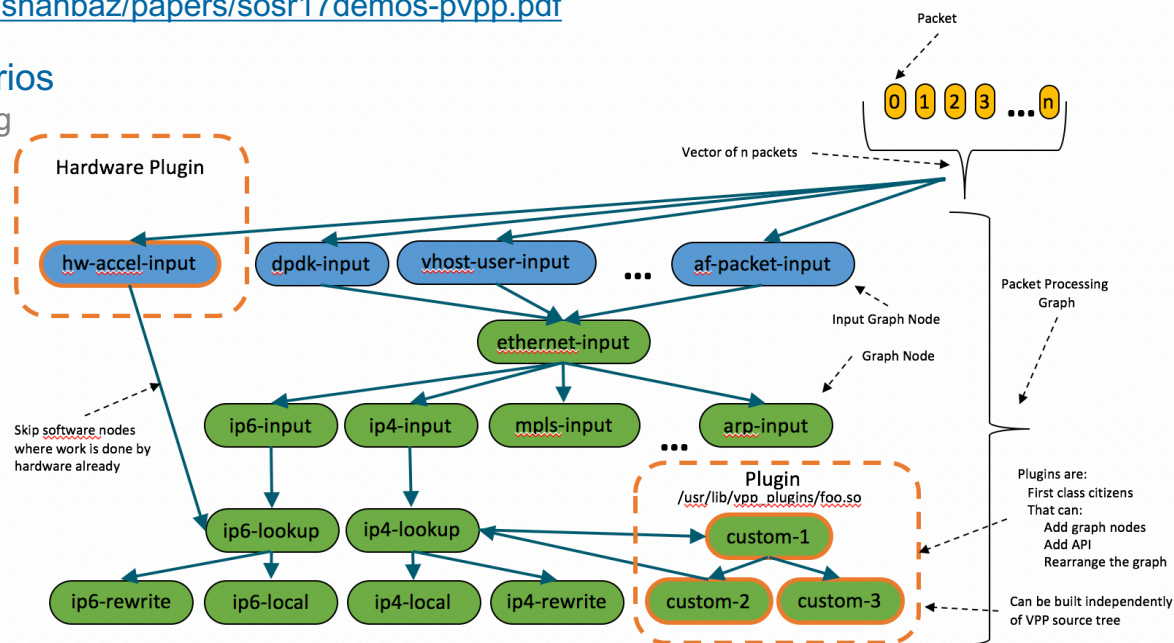
o **VPP as a P4 target is a reality**
  PVPP: A Programmable Vector Packet Processor (SOSR '17)
  http://www.cs.princeton.edu/~mshahbaz/papers/sosr17demos-pvpp.pdf

o **Deployable in multiple scenarios**
  - Hypervisor/Host Networking
  - Embedded Systems
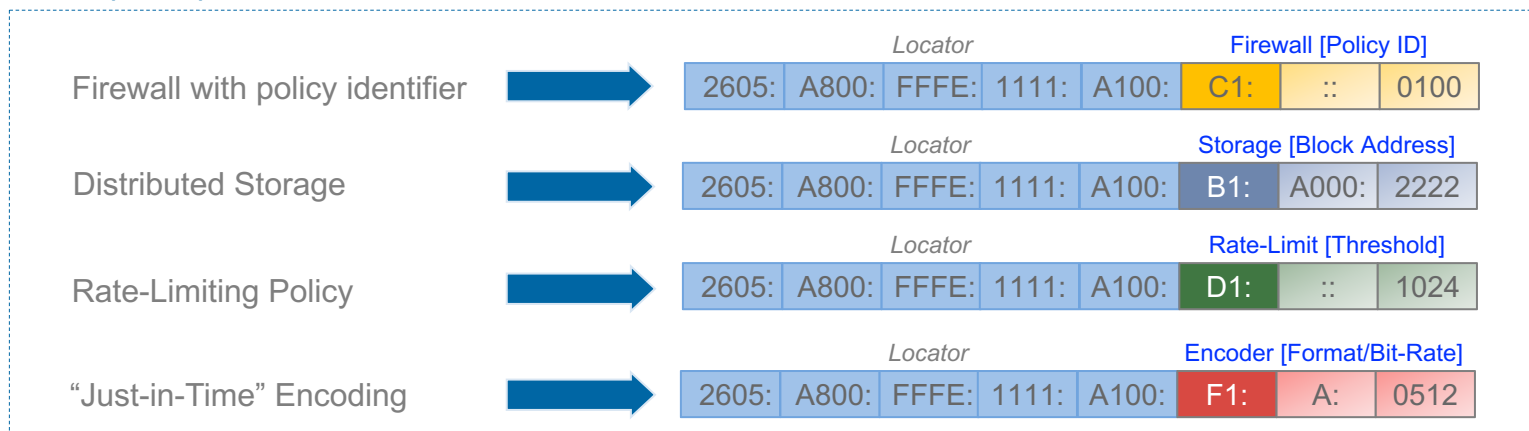 - vNF/cNF Data Plane
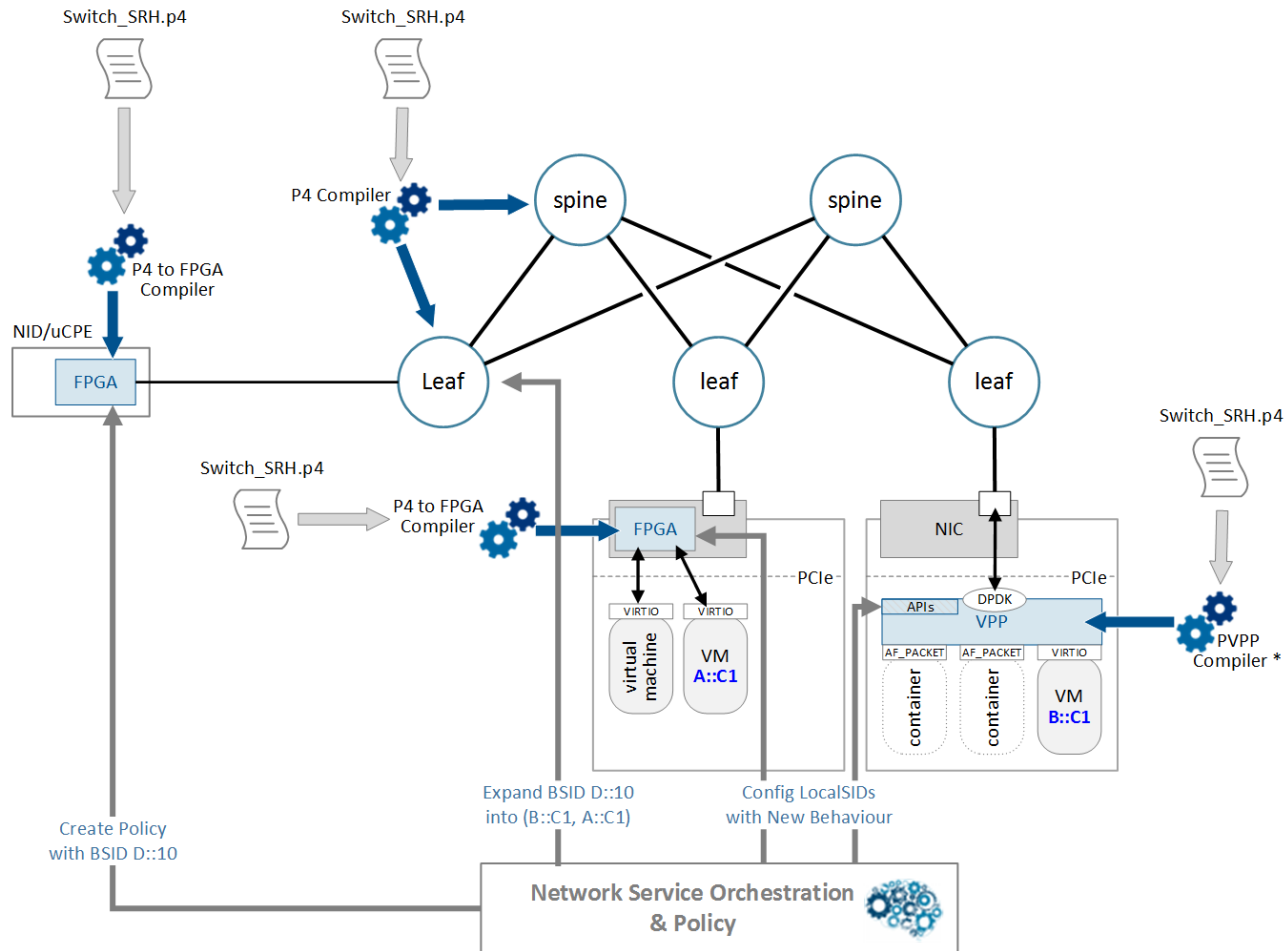

source: https://wiki.fd.io

Bell

Key Enablers - P4 and Programmable Hardware

o **P4 on SmartNICs (FPGAs, NFPs) can tie END behaviors from the data plane directly to the app.**
  - e.g. SR-SPRAY on a NIC

o **P4 to FPGA/NPU compilers give access to existing brownfield.**
  - Can't replace all the hardware in the field … but some can be reprogrammed.
  - Configuring actions based on an EH might prove simpler than trying to support shiny new encaps.

o **Consolidating on P4 helps with usability**
  - Keeping developers focused on a single protocol greatly accelerates innovation.
  - Simpler for end users than support target dependent languages.
  - Builds a larger development community.

o **Leveraging *Binding SIDs* can abstract complex policies for resource limited targets.**

o **Potentially leverage P4 to parse beyond Ethernet (e.g. Optical) ?**
  - Opens the door to new possibilities (LocalSID for wavelength?)

Example of potential END behaviors

			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

Bell

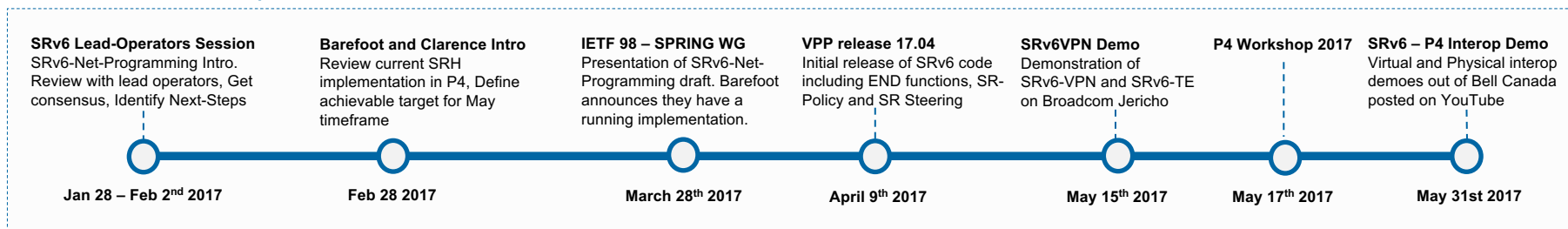# Extending the Network With a New Behavior

- o Program the new data plane behavior in P4.
- o Compile to multiple targets.
- o Program the control plane and/or northbound APIs required.
- o Configure the new function using the behavior (e.g. attach a LocalSID to the behavior).

Continuously Extending the Network … With no new Protocol
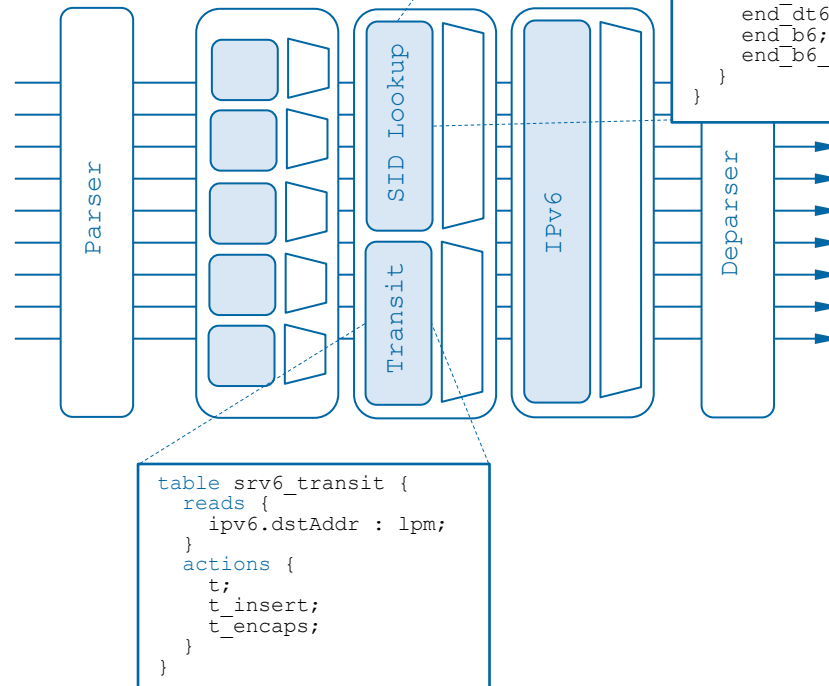
SRv6 Implementation in P4

o **Idea initiated after the review of the SRv6 Network Programming concept and its surrounding academic work.**
  - Lead Operators face to face in January 2017.

o **Collaborative work between the SR team at Cisco Systems, Barefoot Networks and Bell Canada.**

o **Provided a great use case to prove the disruptiveness of PISA.**
  - Evaluate behavior of P4 syntax versus hardware taxing encapsulation scheme.
  - Demonstrate rapid innovation open programmable data planes offer.

o **Based on SRv6 network programming draft**
  - https://tools.ietf.org/html/draft-filsfils-spring-srv6-network-programming-00
  - ~90% of predefined behaviors implemented before draft submission at IETF98
  - **10 segments** in two SRH with L2/L3 forwarding
  - ~400 lines of P4 codes

o **Complete virtual and physical interop topologies in Bell Canada labs.**
  - Demo will be public in a few week

SRv6 in P4 Development Timeline

SRv6 Lead-Operators Session
SRv6-Net-Programming Intro.
Review with lead operators, Get
consensus, Identify Next-Steps

Barefoot and Clarence Intro
Review current SRH
implementation in P4, Define
achievable target for May
timeframe

IETF 98 – SPRING WG
Presentation of SRv6-Net-
Programming draft. Barefoot
announces they have a
running implementation.

VPP release 17.04
Initial release of SRv6 code
including END functions, SR-
Policy and SR Steering

SRv6VPN Demo
Demonstration of
SRv6-VPN and SRv6-TE
on Broadcom Jericho

P4 Workshop 2017

SRv6 – P4 Interop Demo
Virtual and Physical interop
demoes out of Bell Canada
posted on YouTube

Jan 28 – Feb 2nd 2017   **Feb 28 2017**   **March 28th 2017**   **April 9th 2017**   **May 15th 2017**   **May 17th 2017**   **May 31st 2017**

Bell

SRv6 Implementation in P4

o **Concurrent design and implementation.**
- Changes are frequent - 4 revisions of SRH draft in 2017 alone.
- Can track the latest spec.

o **Rapid Prototyping and Testing**
- New SRv6 endpoint functionalities can be added and tested in just a few hours !!

o **Zero Effort Silicon Migration**
- Can develop/test with simulation or P4 targets seamlessly.

```
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;
  }
}
```

Bell

… it's not that easy

Still a Lot of Work To Do

Programmable data planes and P4 are key in achieving this *Extensible Network* but some challenges still need to be addressed:

o Coding in P4 does not equal to knowing how to program a data plane.
   • Effective data plane programming is an expertise not easily found.
   • Co-development with OEMs will be helpful (code revision, sanity checking, etc.).
   • There will be an industry need for services around P4 code development and multi-target SDE environments.

o A programmable data plane alone does not make a network.
   • We can now rapidly create new SID behaviors to address a requirement or new network function.
   • Leveraging them still needs a control-plane element (e.g. mapping generated APIs to CP, northbound orchestration is essential).

o Lots of standardization efforts at IETF
   • Header insertion is still progressing in the working groups (https://tools.ietf.org/html/draft-ietf-6man-rfc2460bis-11, https://www.ietf.org/id/draft-voyer-6man-extension-header-insertion-00.txt)
   • For interoperability, providers still need standards … but these need to follow the pace of innovation.
   • Programmable data planes give us the ability to move forward iteratively and still reach standardisation (e.g. coding a draft revision in hours instead of months).

Bell

Thank You