IEEE IM™

IEEE/IFIP IM 2019
Tutorial T3

Packet-Programmable Networks and BPP: A New Way to Program the Internet

Washington DC, 8 April 2019

Alexander Clemm, Uma Chunduri, Padma Pillay-Esnault
{alexander.clemm|uma.chunduri|padma@huawei.com}

Outline

- Motivation: Why are we here?
- Existing approaches and state-of-the-art
- New Networking and Network Programming Framework: BPP
- Use cases and example applications
- Open research questions
- Conclusions

Outline

- Motivation: Why are we here?
- Existing approaches and state-of-the-art
- New Networking and Network Programming Framework: BPP
- Use cases and example applications
- Open research questions
- Conclusions

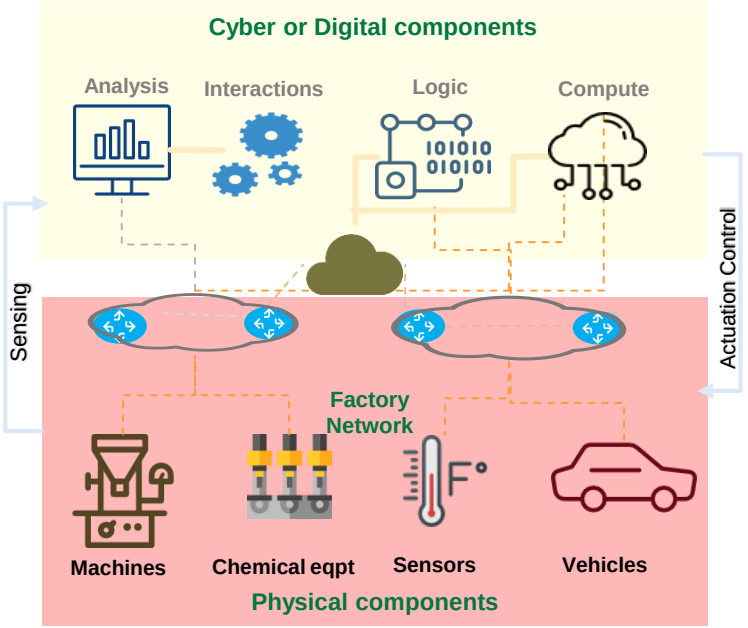
Shifts on how we approach networking

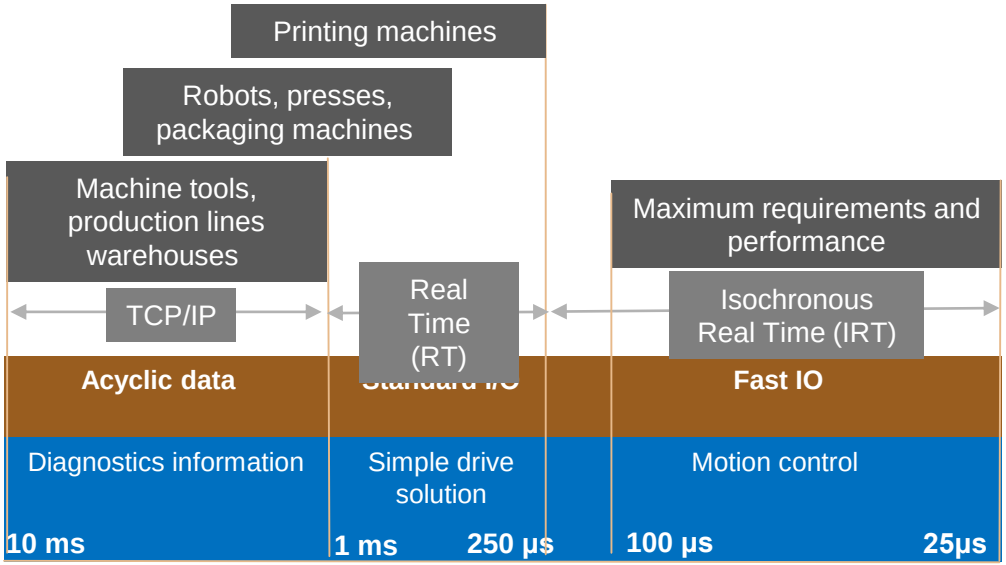
- **From vendor-defined to operator-defined networks**
  (examples: NFV/SFC, SDN, DevOps)
  Network programmability challenges legacy network appliance model
- **From decentralized routing to source routing and packet-defined paths**
  (examples: SR, PPR)
- **From decentralized to centralized control**
  (examples: SDN, PCE)
- **From management as second-order problem to intrinsic networking feature** (examples: IOAM, telemetry)
- **From best effort networking to "high precision" networking**
  (examples: DetNet, TSN)

Coupled with a sense that we are bumping into limitations of Internet technology for new services

- "Best Effort" principles increasingly questioned
- Example: ITU-T FG-NET2030 initiative
  - Emerging new use cases
    - Tactile Internet, integrated terrestrial/satellite communications, holographic avatars, ...
  - Implications/requirements for future networking services
    - Quantifiable delay guarantees, coordination across streams, ...
  - Leading to ramifications on networking architectures
    - From one Internet to "Manynets"

Fast Response and Low Latency in Automation and Control





High-Precision Latency in Industrial Internet

Low Latency for Machine Control

Source: James Coleman (Intel) TSNA'15 - Processor and OS Tuning for Event Response and Time Sensitive Systems.pdf

Network impact of holographic media

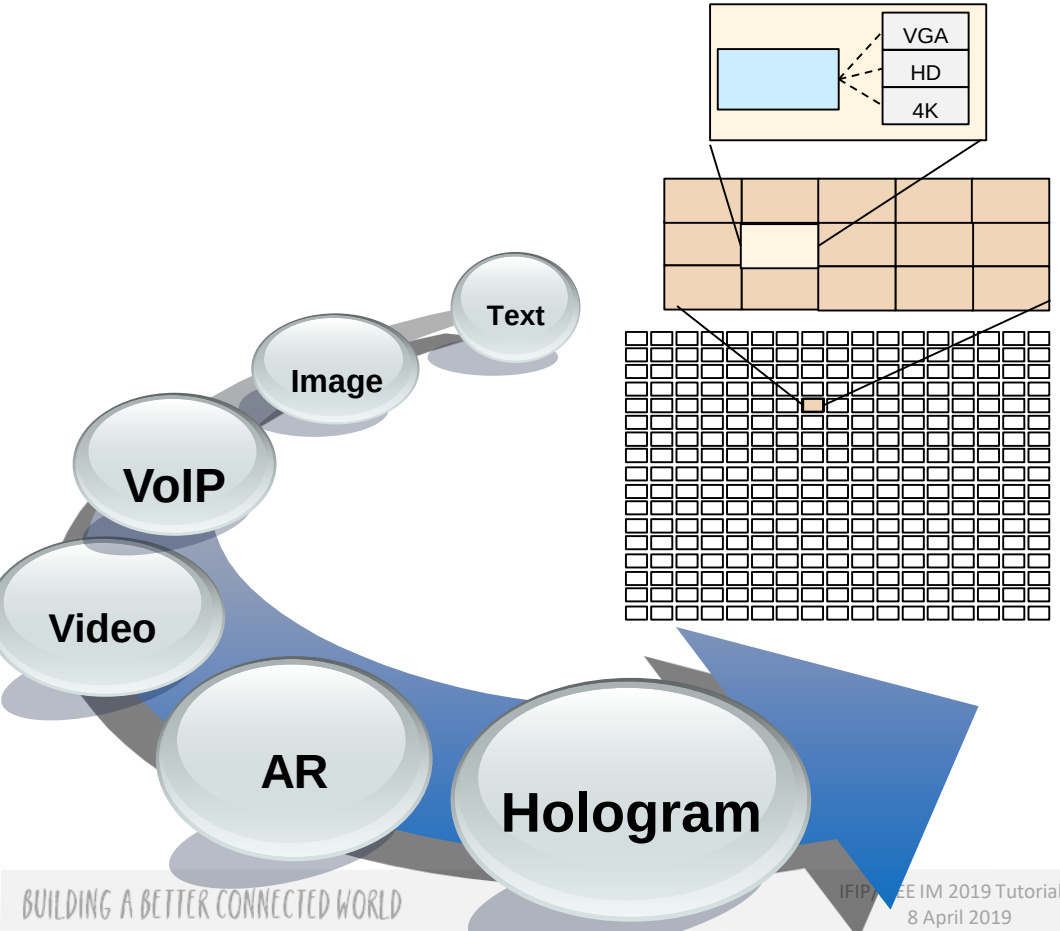


Image	#dim	Variables
Still picture	2	Image resolution, color depth
Video	3	Add framerate
VR	4	Add tiles
HTC	5..7	Add 6DOF (angle, tilt, depth)

- Raw bandwidth reqs. enormous
- Significant compression possible but imposes its own requirements
  - Multi-flow synchronization and coordination (“coordicast”)
  - Advanced priorization techniques

Additional shifts in managing networks

- Greater emphasis on ease of operations
  - SDN: single point of management
  - Rise of analytics and machine learning
  - Rise of telemetry to feed analytics
  - From device to network: IOAM for richer data
- Greater trend: disappearing infrastructure
  - Let users worry about the service they want, not the infrastructure to get it
  - Other domains are ahead of networking but writing is on the wall...
  - “Networkless networking”?

On the road to networkless networking: Serverless Compute

Physical servers

High heterogeneity
High capex
Planning cycles: months
Power, space, ...
High opex
Need to manage phys infra
(Tivoli, CfEngine, ...)

VM – IaaS, PaaS

Low heterogeneity
No capex
Planning cycles: hours
No power, space
Better opex (still considerable)
No need to manage physical infra
Need to manage virtual infra
(OpenStack, Vmware, ...)

Containers

Low heterogeneity
No capex
Planning cycles: minutes
No power, space
Better opex
No need to manage physical infra
Need to manage virtual infra
(Kubernetes, "DevOps", ...)

Serverless ("Lambda")

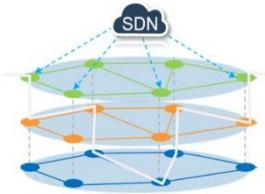
No heterogeneity
No capex
Planning cycles: seconds
No power, space
Best opex
No infra to manage

On the road to networkless networking

Traditional networks

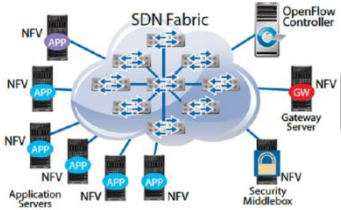
Planning cycles: months
Program granularity:
device+network standards
High capex
High opex
Need to manage phys infra
(HP OV, SolarWinds, ...)

SDN



Planning cycles: hours-days
Program granularity:
controller+device
Lower capex (still considerable)
Lower opex (still considerable)
Need to manage phys infra
(ODL, ACI, ...)

NFV & SFC



Planning cycles: minutes
Program granularity:
controller+network function
Lower capex
Lower opex
Need to manage virtual infra
(Kubernetes, "DevOps",
telemetry/analytics, ...)

???

Planning cycles: subseconds
Program granularity:
packet+flow
No capex
Lowest opex
No infra to manage

Control flows and packets directly,
gain detailed visibility and ensure service level guarantees
at packet and flow level without need to manage infra

How to get there?

- Current approach: proliferation / fragmentation of technology
  - Every point feature results in separate protocol / extensions
    SR, NSH, DetNet, IOAM...
- Concerns:
  - Significant growth in underlying complexity
  - Interoperability concerns
  - Dependence on ever-slowing standardization cycles; ossified, non-agile
  - New feature / requirement gaps open faster than existing ones are addressed

    Examples: Quantifiable SLOs, smarter telemetry, alternative routing schemes (QoS-dependent, incentive-based), SL-dependent accounting, precision measurements, ...
- SDN & co (and some other trends) empower MSDCs and to some extent large SPs, but more agility and better support for other players is needed

What's needed

- Rethink approaches that allow to extend and customize network capabilities
- Provide generalized vs. piecemeal mechanisms
- Extend "DevOps" – vendor → operator → customer-defined

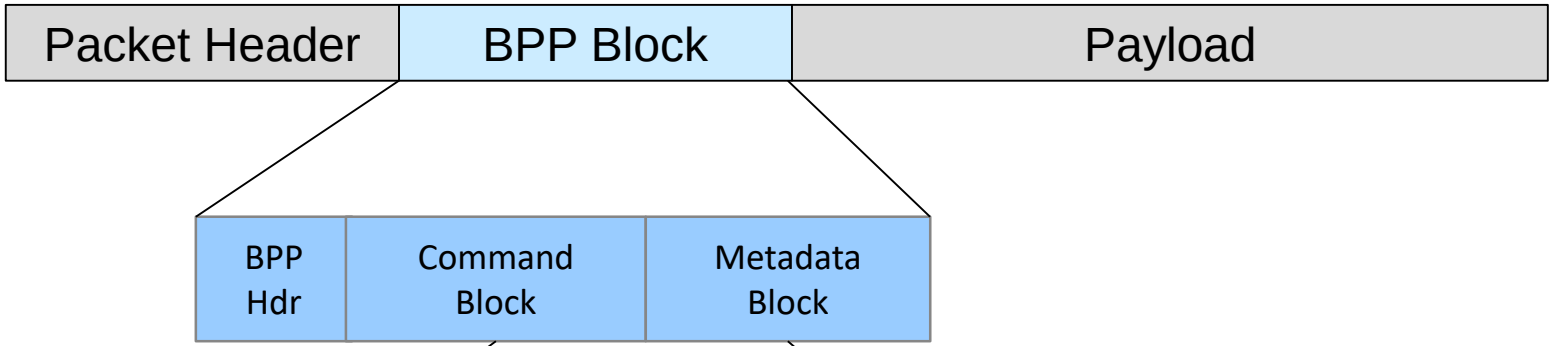
This tutorial:

- One proposed approach – "BPP"
- Still in concept stage, early results (PoC under development)
  - Not a polished system; we don't have all the answers
- Stimulate discussion, point out research opportunities, encourage "out of the box" thinking

What we are proposing

- Processing of packets is guided by commands and metadata carried in the packet
  - Packets carry commands and metadata to define packet and flow behavior
  - QoS, reservations, forwarding decisions, operational visibility
  - Nodes act on SLO metadata, affect forwarding decision depending on local conditions, conduct measurements, assess network/path conditions, ...
  - Stateful flow processing as an option (think programmable flow cache)
  - Many management use cases: more powerful service assurance, accounting, embedded analytics, learning, caching optimizations
- Packets and flows are “programmed” dynamically from the edge
  - Edge node injects metadata and commands into packets
  - No need to reprogram every device
  - User-definable, near-zero-delay control loops
  - Re-converge data and aspects of control plane

BPP Packet (high level)



Benefits

- **Enable high-precision networking services:**
  Determine precise packet treatment, dynamic and context-aware
- **Provide service-level guarantees:**
  Obtain unprecedented visibility, measurements
- **Programmability:**
  No dependence on lengthy standardization and product cycles
- **Isolation:**
  Affect only behavior of own packets and flows, not device as a whole or other packets
- **Holistic:**
  One technology to address multiple concerns concurrently
- **“Networkless”:**
  Networking infra is opaque to users to greatly facilitate management

Outline

- Motivation: Why are we here?
- Existing approaches and state-of-the-art
- New Networking and Network Programming Framework: BPP
- Use cases and example applications
- Open research questions
- Conclusions

Existing approaches, state-of-the-art

- IETF approaches
  - SR
  - iOAM
  - Detnet
- Network softwarization
  - SDN
  - NFV and SFC/NSH
  - P4
- Past approaches: Active Networks
- Summary

Segment Routing

- SR Motivation
  - Shortest-Path Routing does not always result in most desirable paths
  - Operators want more control over paths taken, compute paths via controller optimized not just for cost but resilience, utilization, QoS, ...
- SR Concept
  - SR allows to “source route” packets, putting a specified path sequence of “segment identifiers” into packet headers
  - SR extensions allow to “repurpose” SIDs to invoke additional hop functions, which potentially allows to program the network via packets

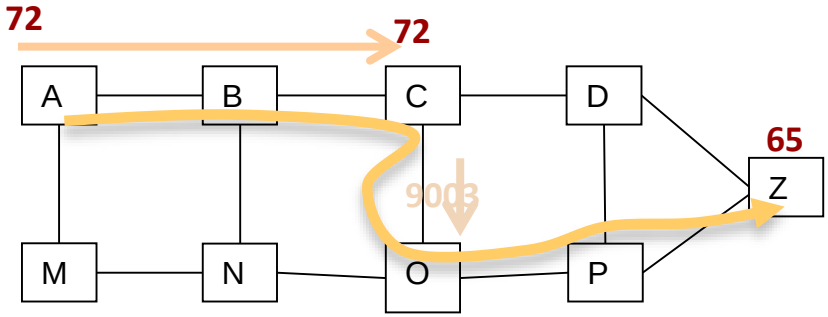
SR Problems being solved & Status

1. Path steering in the network
   - MPLS Data Plane
   - IPv6 Data Plane

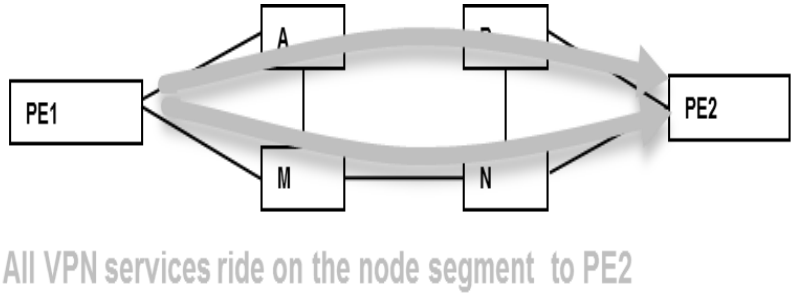
2. Reducing number of protocols in the core network
   - MPLS control protocols LDP/RSVP-TE are not needed

3. Network programmability
   - For IPv6 data plane by encoding functions in the IPv6 address

SRv4 – Quick overview

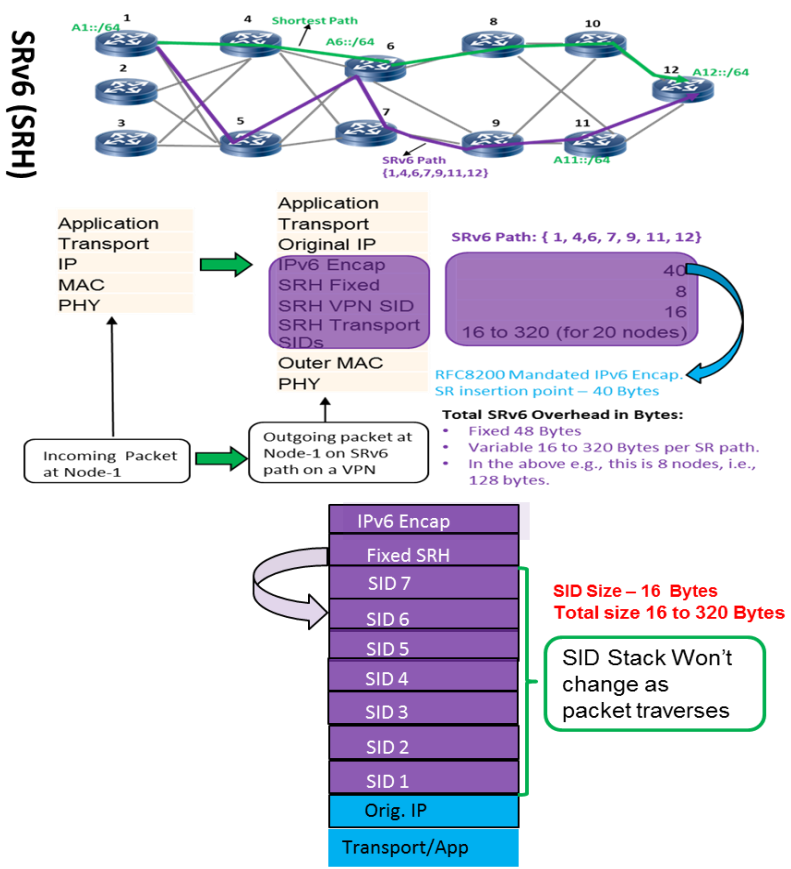


- Source Routing (Node Segment, Adj. Segment)
- ABCOPZ is expressed as {**72**, 9003, **65**}
- Node Segment is at the heart of the proposal
  - ECMP multi-hop shortest-path
  - in most topologies, any path can be expressed as list of node segments



- VPN Service (L3VPN)
- VPWS is being added by Huawei (draft TBD)
- Less number of protocols in the core
- SDN capability for transport path/on-demand TE path

SRv6 – Quick overview



- Second data plane for SR ➔ SRv6 with SRH
- SRH is a IPv6 hop-by-hop routing Extension Header (EH).
- SRH resurrects the deprecated routing header RH0 (security issues), with some solutions (e.g. ICV TLV in SRH).
- Explicit path is described in SRH with 16 bytes SIDs(IPv6 Addressed)
- Extensions to reach host for giving the path.
- Network Programming is a key corner stone (still at early stages @IETF).

Discussion

Summary & References:

1. SRH allows functions to be executed in the SIDs at intermediate nodes, while doing path steering (with available bits in the SID)
2. Refs:
   https://tools.ietf.org/html/rfc8402 (SR architecture)
   https://tools.ietf.org/html/rfc8354 (Use cases)
   https://tools.ietf.org/html/draft-ietf-6man-segment-routing-header-18 (SRH)
   https://tools.ietf.org/html/draft-ietf-spring-segment-routing-mpls-19 (SR-MPLS)

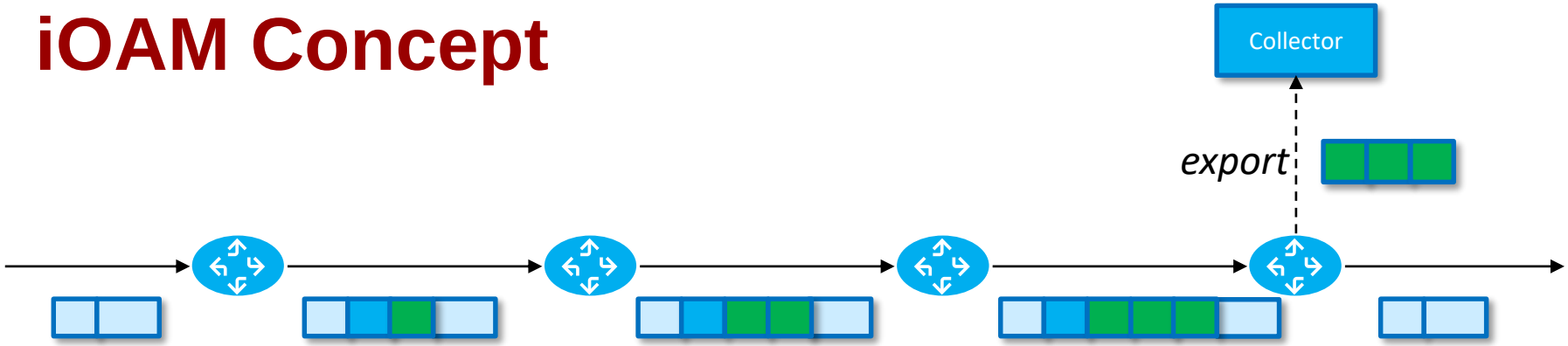
Few Challenges being worked out with SR:

1. Excessive path overhead Issues
   - Potential MTU/Fragmentation issues with large SID stack (SR-MPLS, SRH)
   - NW/Path overhead relative to actual application data (this is critical in some deployments)
2. Hardware capabilities & performance-related issues with increased SID size
3. No resource reservations for the path
   - SR only provides explicit path and TE attributes are not in scope
4. Label in SR stack can't identify the path; hence no performance measurements

In-Situ OAM (iOAM)

- Motivation
  - Getting a sense of what happens to individual packets in transit is hard
  - MSDCs need ways to collect key telemetry data to identify sources of jitter
  - Verifying that packets take a certain path (given load balancers and many levels of virtualization) can be a challenge as well
- iOAM Concept
  - Hops add telemetry information to packets as they traverse the network
  - Select from a handful of information elements: ingress-if, buffer occupancy, egress queue depth, transit delay, time stamps
  - Proof of transit: Update PoT data based on a share of a secret (SSS: Shamir's Secret Sharing): use combined key to compute a value per a polynomial function
  - Allow to set hop limits, address specific node IDs to keep packet length limited
  - Data export proposed to use IPFIX

iOAM Concept



What to do when length exceeded; is this a probe or a production packet; export data directly from node instead of adding to packet, ...

24 bits, each indicating a piece of data to include (node ID, time stamp, transit delay, queue depth, Proof-of-Transit, etc)

Each node adds its data (as long as node limit, length limits not exceeded)

iOAM Status

- Currently under standardization at IETF, mostly in IPPM working group
- Important references
  - Data fields for in-situ OAM: https://tools.ietf.org/html/draft-ietf-ippm-ioam-data-04
  - Proof of transit: https://tools.ietf.org/html/draft-brockners-proof-of-transit-05
  - In-situ OAM raw data export with IPFIX: https://tools.ietf.org/html/draft-spiegel-ippm-ioam-rawexport-01
  - Various transport encapsulations, e.g. Geneve, VXLAN: https://tools.ietf.org/html/draft-brockners-ippm-ioam-geneve-01, https://tools.ietf.org/html/draft-brockners-ippm-ioam-vxlan-gpe-01

Discussion

- MSDC optimization (e.g. identify causes of packet jitter, validate paths of production packets) as killer use case
- Allow for customization (within tight boundaries) of data being gathered to mitigate issues of packet size
- Shortcomings and what iOAM does not address
  - Huge data volume when used beyond individual packets – every packet generates records that may be larger than payload itself
  - Reduced usable packet payload, issues in supporting variable-length paths, no ability to aggregate data across paths (e.g. for bottlenecks)
  - Very low-level – no aggregation of data across packets of flows – any aggregation requires post-processing

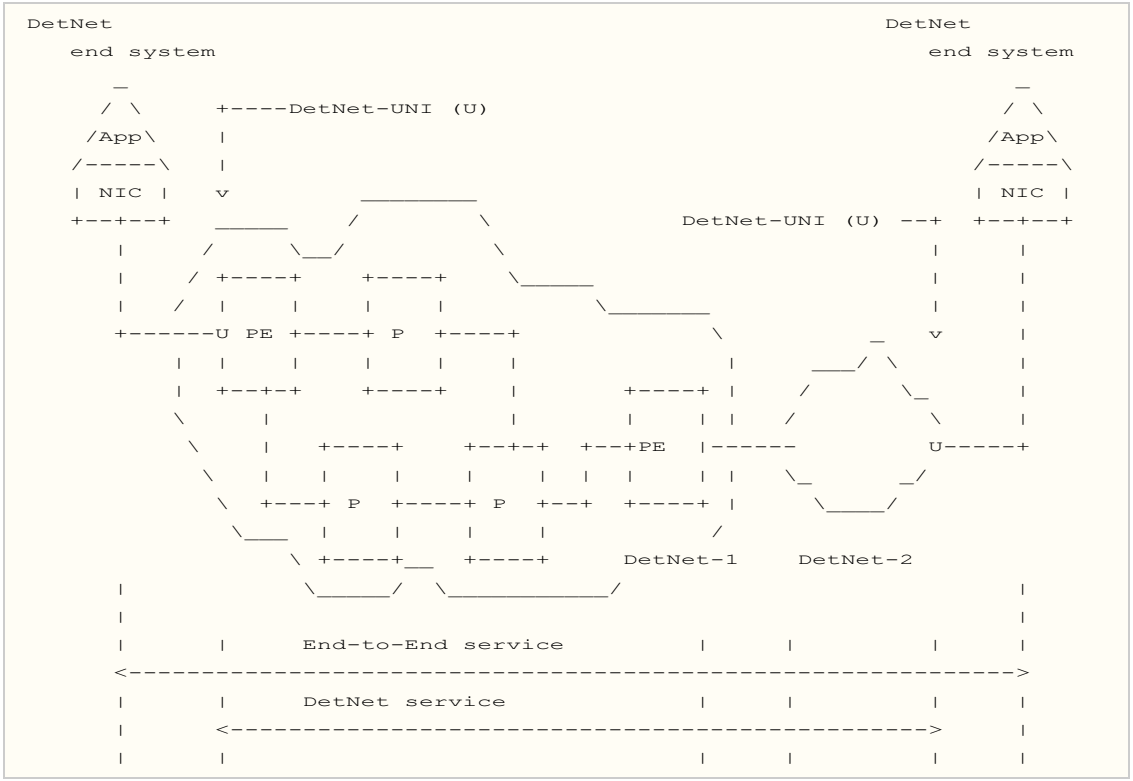
DetNet

- The Deterministic Networking (DetNet) IETF WG focuses on deterministic data paths that operate over Layer 2 bridged and Layer 3 routed segments where such paths can provide;
  - bounds on latency, loss,
  - packet delay variation (jitter) and high reliability.
- Detnet Architecture encompasses the data plane, OAM, time synchronization, management, control, and security aspects.

- Data Plane work includes how to use IP and/or MPLS to support a data plane method of flow identification and packet forwarding over Layer 3.

- DetNet WG "..explicitly excludes modifications of transport protocols, OAM, Layer 3 forwarding, encapsulations, and control plane protocols." - https://datatracker.ietf.org/wg/detnet/about

DetNet Architecture



- Goal is to provide flow level QoS characteristics across L2/L3
- Use time synchronization for providing some of the services
- Use existing control (e.g. RSVP-TE) and mapping plane (e.g. ACTN/RFC8453) technologies
- Use existing data plane technologies like MPLS and IP with existing/additional flow indicators

https://datatracker.ietf.org/doc/draft-ietf-detnet-architecture/

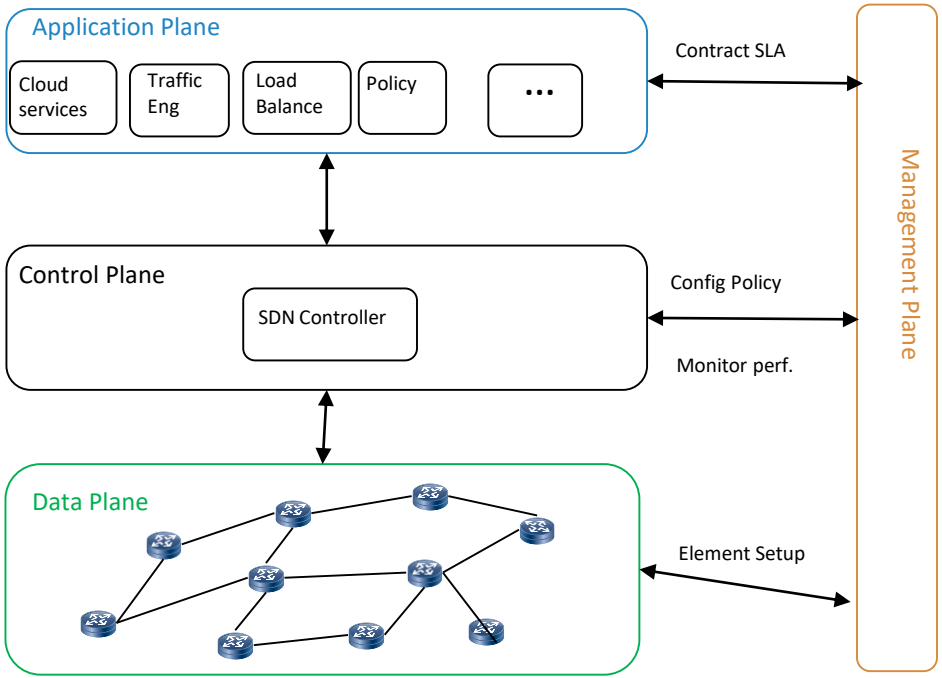
DetNet Status & Discussions

- DetNet is designed for enterprise and private wan networks where scale is limited
- There is desire to make it large scale and also look for alternate and efficient control plane and data plane solutions
- Detnet Key documents in IETF
  - Architecture: https://datatracker.ietf.org/doc/draft-ietf-detnet-architecture/?include_text=1
  - IP Data Plane: https://datatracker.ietf.org/doc/draft-ietf-detnet-dp-sol-ip/?include_text=1
  - MPLS Data Plane: https://datatracker.ietf.org/doc/draft-ietf-detnet-dp-sol-mpls/?include_text=1
- Discussion
  - BPP proposes an alternative data plane which can work with both L2 and L3
  - BPP Statelets enhancement would enable per-flow or aggregated state to apply lot of high-precision services related to the flow
  - In summary, BPP is a complementary technology and can bee seen as an enabler for DetNet requirements

Software Defined Networking (SDN)

- Origins: Academics in campus then particularly useful in DC
- Motivation: Large scale (DC) and diversity of devices and vendors make networks harder to manage:
  - Static and complex architecture: Difficult to get a whole picture which is needed for QoS, TE in a highly fluctuating environment
  - Inconsistent policies across different vendors
  - Vendor dependence
- Concept:
  - Split the management, control and data plane for better flexibility and access by application layer.
  - Centralized control
  - Program the behavior of the network using APIs

SDN Architecture - ONF



Simplified Open Networking Forum (ONF) view

Applications Plane: Apps communicate behaviors and needed resources with the SDN Controller. Apps- networking management, analytics, or business app used to run large DC.

Control Plane: Controller receives instructions or requirements from the App layer and executes on the network devices. The controller communicates back to the SDN Applications information about the network from the hardware devices (telemetry…)

Data Plane: Collection of network devices responsible for the forwarding and data processing.

Management Plane: Interacts with planes to implement contracts, policy, monitoring and device config

Discussion

- Moved from “vendor-defined” to “operator-defined” (but not user- or administrator-defined) network
- Significant controller complexity leads to its own challenges (lifecycle of controller apps, discovery, control loop latency, ...)
- Control loop latency can be an issue – will need embedded capabilities & architecture extensions to enable “smart edge”

Network Function Virtualization(NFV)

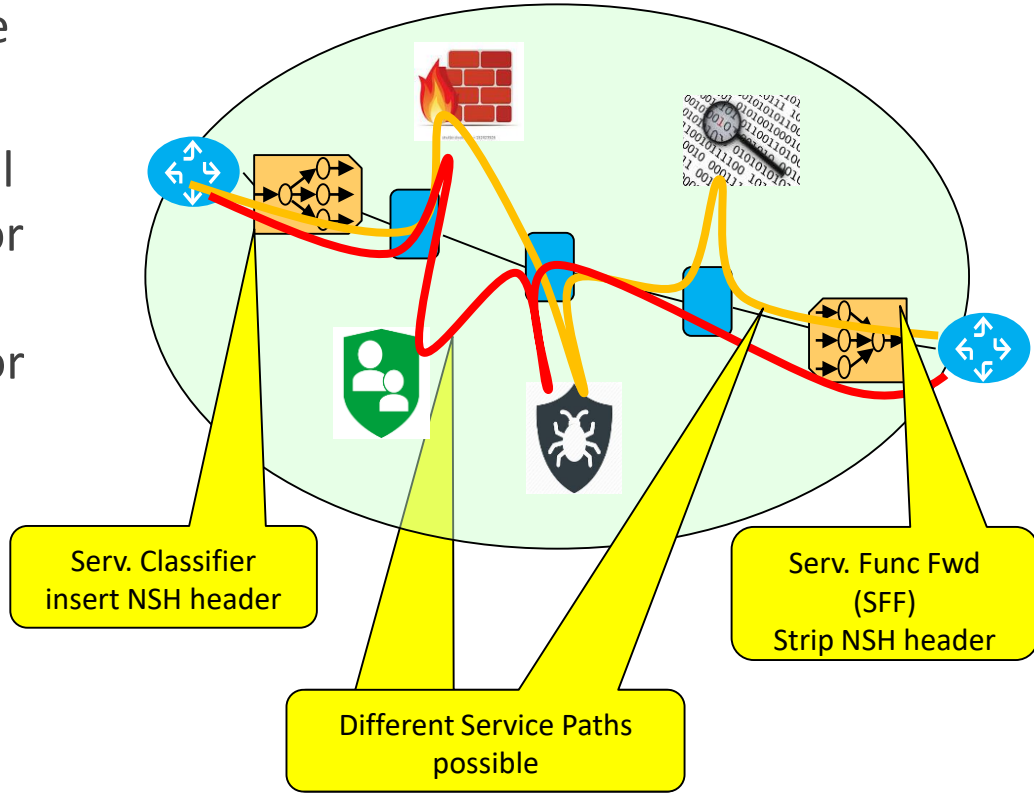
- Origins: Consortium of ISP, standardization in ETSI
- Motivation: reduce cost, TTM for new services Complex provisioning/managing HW from different vendors
  - Reduce Capex/Opex (equipment, space, …)
  - Faster deployment of innovation
- Concept: Relocate network functions from dedicated to virtualized appliances hosted on generic servers
  - Routers, Firewalls, gateways, CDN, WAN accelerators, SLA assurance
- Complementary to SDN but share similar objectives (commoditized HW, central control, abstraction of planes)

Service Function Chaining

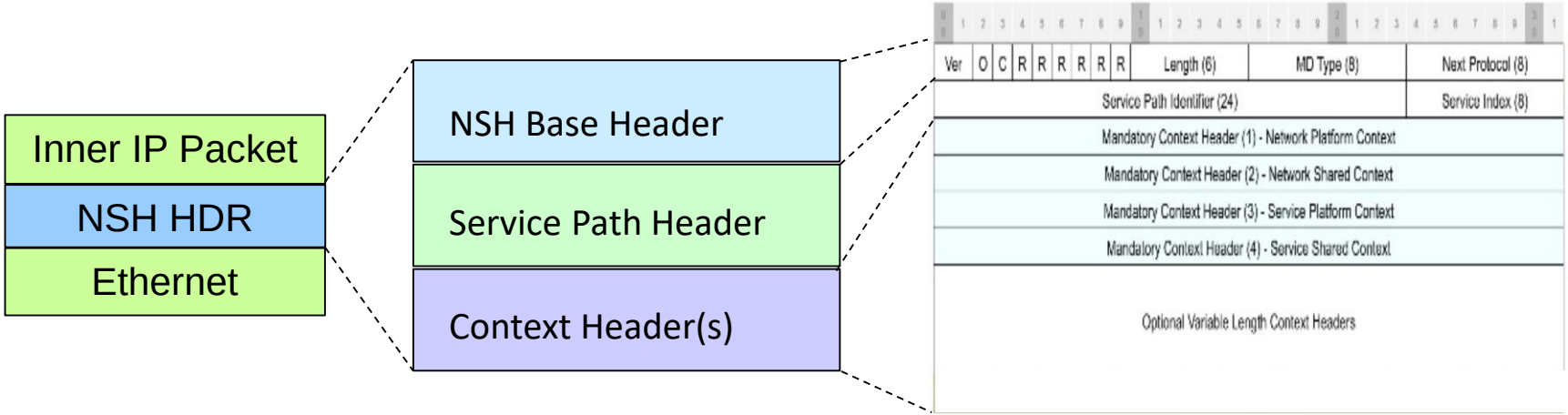
- Motivation: Create a service plane which is a service chain of connected network services (L4-7 like firewalls, virus analyzer, network address translation [NAT], intrusion protection, parental control, video optimizer).
- Concept: Relies on the principles of virtualization of SDN.
- Standardization: Network service chaining standards in several SDOs.
  - IETF – SFC wg definition of architecture to define how network flow classification can be used to route traffic between service functions (NSH).
  - ETSI - Service architecture based on network forwarding graphs to route traffic between VNF with Network Service Header (NSH).
  - ONF - SDN service chaining framework using OpenFlow.

Network Service Header

- Motivation: Implentation SFC in the packets
- Concept: A “service-plane” protocol describing service function chains or paths in the form of a header. The header can be inserted in packets or frame to steer the packets.
- Standardization: Network service chaining standards in several SDOs.
  - IETF – In SFC wg RFC 8300



NSH Architecture



Base header: Describes the service header and payload
Service Path Header: Describes Service path header and location within
Context Headers: opaque metadata (mandatory and optional)

NSH Status and Discussion

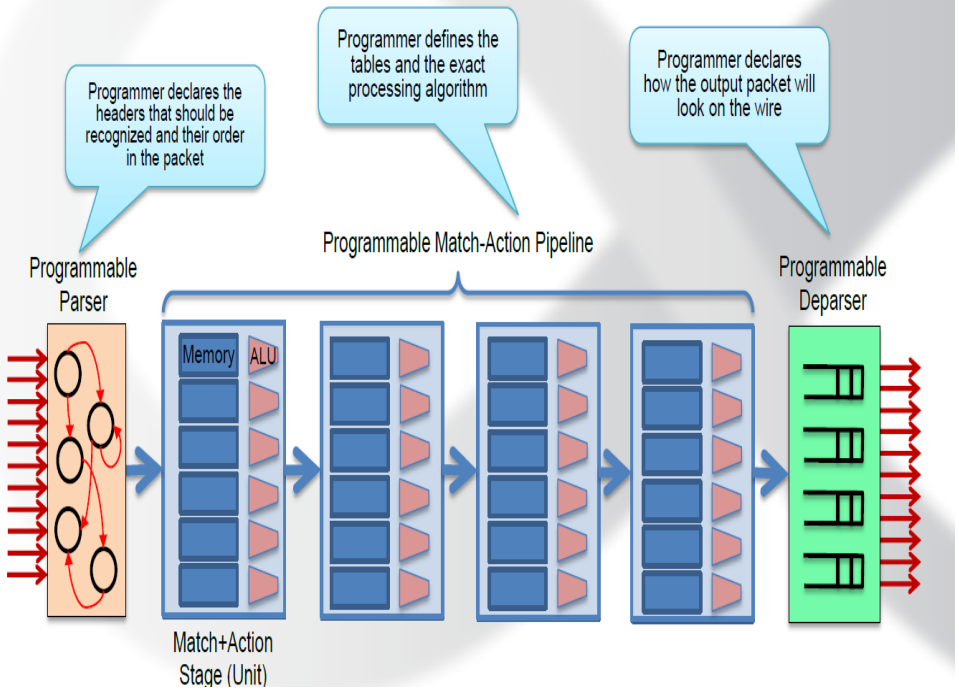
- Advantages of NSH - Defines a Service plane
  - Transport independent and topology Agnostic
  - Support all types of topologies (linear not needed anymore)
  - Simplicity
- NSH limitations
  - Centralized controller
  - Need a classifier
  - Does not handle a user defined

Discussion

- SDN, NFV and SFC with NSH: use similar abstraction and are complementary to each other.
- Downsides
  - Centralized control means a lot of post processing
  - Difficult to get the user to provision the network
  - Security

P4

PISA: Protocol-Independent Switch Architecture



- P4 ➔ Programming language to describe packet processing
- First appeared in SIGCOMM CCR 2014
- Three goals:
  - (1) Re-configurability in the field
  - (2) Protocol independence
  - (3) Target independence
- Maintained by nonprofit organization: P4.org
- Emerged as de facto standard for packet processing

Source: P4.org

P4 Status

- Flexible Match+Action ASICs (currently available programmable ASICs)
  - Intel Flexpipe; Cisco Doppler; Cavium (Xpliant); Barefoot Tofino..
- Important references
  - All @ p4.org
- What P4 does well:
  - Programmable parser; protocols and target independence
- Ongoing research in P4 community yet to be addressed
  - Supporting operations on payloads Tbits/sec
  - Virtualized instances
  - High performance stateful features
  - Programmable schedulers

BPP – P4 Relationship

- **Difference highlights**:
  - **Purpose:** P4 facilitates the implementation of protocols, including packet-parsing, protocol semantics, FIB programming. BPP is a protocol that allows to define packet and flow behavior.
  - **Scope**: BPP commands program a packet and its flow, defining its behavior as it traverses the network. P4 programs a node.
  - **Architecture**: P4 relies on controller-based architecture, communicating with network devices directly. BPP is injected via edge nodes.
- **P4 and BPP are in fact complementary:**
  - Could leverage P4 to implement BPP support
  - New requirements for P4 (that are not supported today):
    - Support for Data Items
    - Support for Dynamic Data Structures (varbit fields insufficient)
    - Support for State, infra for programmable statelet cache

Active Networks

- Area of major research in late 90'ies to early 2000's
- Idea: Inject mini-programs into networks, examples:
  - Process user data – e.g. transcoding, custom compression
  - Update QoS and ACL rules to process flows
  - Deploy monitoring rules to facilitate operations
- AN programs written in general-purpose programming language
  - Sprocket (C++ subset), Safe-Tcl, Java)
  - Compile into lower-level assembly-type language (Spanner) for execution)
- Two models:
  - Controller-based ("discrete") – ancestral lineage to SDN, P4
  - Capsule-based ("integrated") – deployed as packets traverse nodes

Active Networks Appeal

- Empower researchers, users to innovate new networking applications without having to depend on product vendors
- Break dependence on vendor-controlled firmware updates and lengthy product cycles
- Powerful use cases, e.g.:
  - Micro-CDN (dynamic caching and proxying of content)
  - Management automation (e.g. probes interrogating / analyzing device state, fault correlation apps, embedded network analytics)
  - Active QoS schemes (e.g. selective and content-aware dropping of packets using application-specific criteria)
  - Active reliable multicasting (with selective retransmission)

Active Networks Issues and Discussion

- Issues with AN
  - Lack of security (or at least perception of it)
    - Commands potentially affect behavior of switch as a whole
    - No way of isolating scope to a particular flow
    - Virtualization deemed insufficient as response
  - General operational concerns
    - Can network spin out of control; where are the circuit breakers
    - Accountability for behavior
    - Lack of development kits and test sandboxes
  - Unease with ability to inspect and modify user payload
    - At the same time, enabler for powerful use cases and concern for abuse
- As a result, never commercially deployed (despite great enthusiasm)

BPP – AN Relationship

- **BPP and AN are very different – not simply "AN 2.0"**
- **Difference highlights**:
  - **Purpose**: BPP commands affect a packet and its flow, defining its behavior as it traverses the network. AN programs a node.
  - **Scope**: AN programs affect the node as a whole. This has caused numerous concerns, from security having network behavior spin out of control. BPP commands affect only their own packet and flow. Inherently, this makes AN security issues non-issues in BPP.
  - **Payload handling**: AN programs can inspect and modify user data (use cases include, for example, payload transcoding). For BPP, user data is off-limits, providing inherently much greater security and privacy.
- AN's lack of isolation/limitation of scope of programmed behavior, exposure of user payload, and related security concerns are non-issues with BPP

	BPP	AN
Typical Use Cases	High-precision services, SL guarantees, flow optimization, “networkless network”	Device reprogramming, stream transcoding, micro-CDN, general operations
Scope	Flow and packet behavior	Device at large (not limited to specific flows)
User payload	User payload off limits: no inspection, no manipulation of user payload, greater privacy, greater security	Inspecting, manipulating, modifying user payload is fair game and central to many use cases
Programming model	Conditional commands (size: 10s of octets) as part of BPP Block executed as packet is processed for that packet	Capsules contain full-fledged programs (size: KBytes) for node as a whole (and can be applied against any flows/packets)
Isolation	BPP commands inherently affect only own packet and flow; cannot reprogram device or other flows	AN programs can affect device and other flows, not just your own; use VMs to limit scope/ extent of device reprogramming.
Instruction set	Catalogue of predefined parametrizable conditions & actions, no general-purpose programming	General-purpose programming: Sprocket, Safe-Tcl, Java; loops, recursion
Support for flow behavior prgm.	Built-in support (e.g. drop detection, packet hashes, KPI measurements)	Requires custom programming; AN programs generally not aimed at a single flow
Packet processing optimiz. support	Integral part (command interdependencies, serialization constraints fully specified)	No special support
Error handling support	Built-in support (e.g. flagging of error condition, drop option)	Custom programming to handle errors required
Metadata and statelets	Inherent support to facilitate declarative (vs imperative) guidance, SLO support, and much more	Not supported

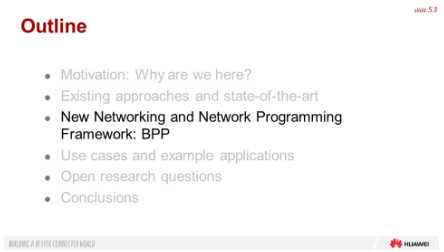
Outline

- Motivation: Why are we here?
- Existing approaches and state-of-the-art
- New Networking and Network Programming Framework: BPP
- Use cases and example applications
- Open research questions
- Conclusions

Preamble

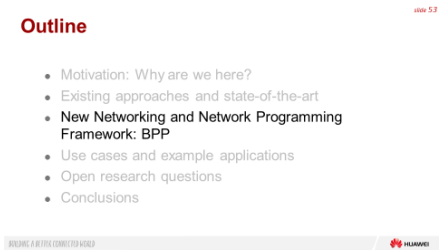
- The following is still in the concept stage
  - PoC is under development but no SDKs available yet
  - Not a polished system – we do not have all answers yet – early results
- Exemplifies a new approach to the challenges outlined earlier
  - Rethinks how networks can be programmed
  - Holistic concept, not a piecemeal approach
- We will point out research challenges
  - Both with our approach and more general

BPP: A New Networking and Network Programming Framework



- BPP Framework overview
- BPP protocol: Blocks, Commands, Metadata
- Stateful extensions
- BPP programming model
- BPP engine, node infrastructure, deployment model

BPP: A New Networking and Network Programming Framework

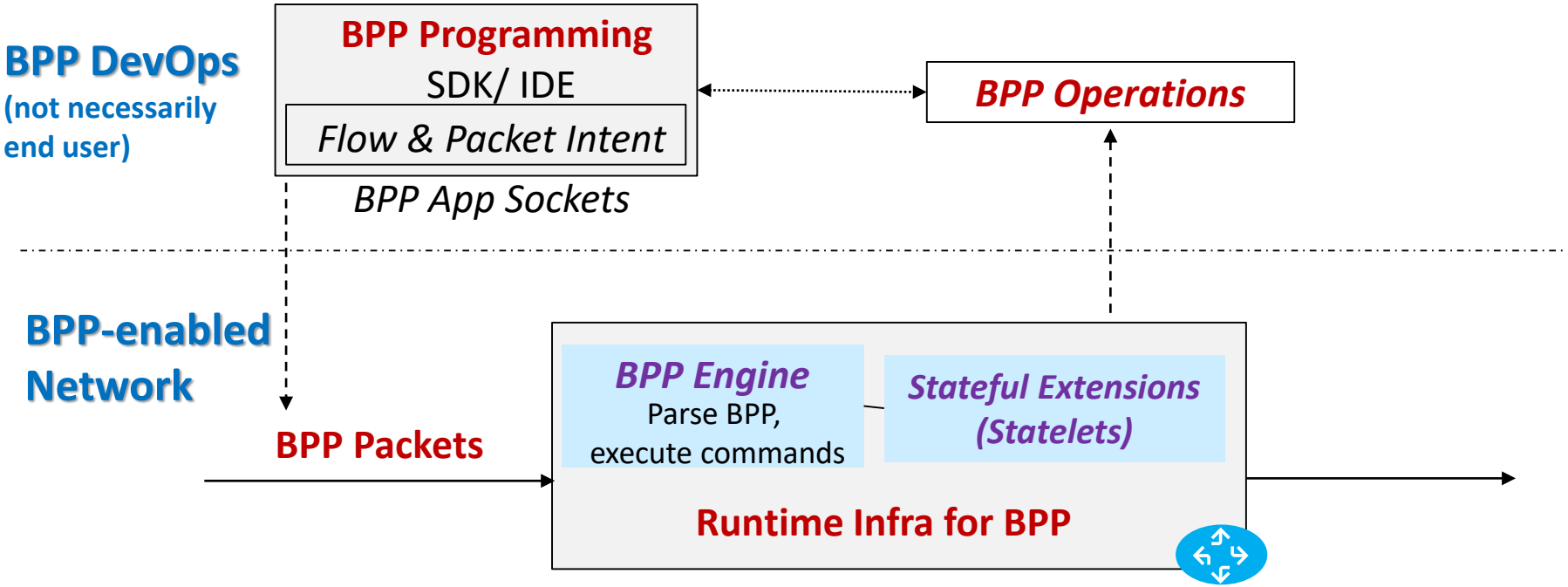


- BPP Framework overview
- BPP protocol: Blocks, Commands, Metadata
- Stateful extensions
- BPP programming model
- BPP engine, node infrastructure, deployment model

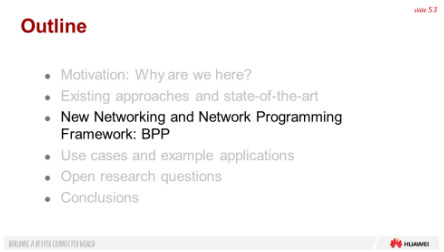
BPP Cornerstones

- **BPP Packets (and Protocol)**
  Carry the information that guides their processing (and processing of their flow)
- **BPP Stateful Extensions**
  Option of dynamically programmable network state
  flow cache (statelets) to interact with BPP Packets
- **BPP Infrastructure**
  Interpret, process, act on information in BPP Packets - exploit advances in processing HW
- **BPP Programming and Operations**
  Deploy, monitor, run BPP Apps; convey user intent,
  enable users to define network behavior – SDKs & APIs

BPP Framework

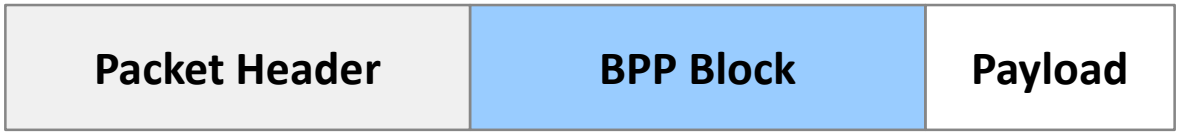


BPP: A New Networking and Network Programming Framework



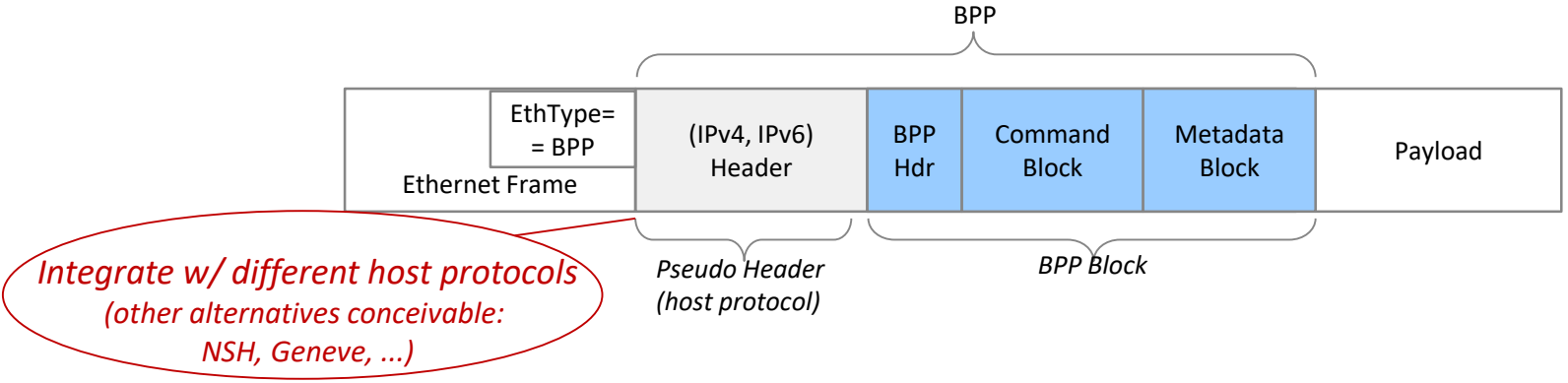
- BPP Framework overview
- BPP protocol: Blocks, Commands, Metadata
- Stateful extensions
- BPP programming model
- BPP engine, node infrastructure, deployment model

BPP Block and BPP Packet Structure



- Insert BPP Block into data packets (between header and payload)
- Different integration options depending on deployment scenario

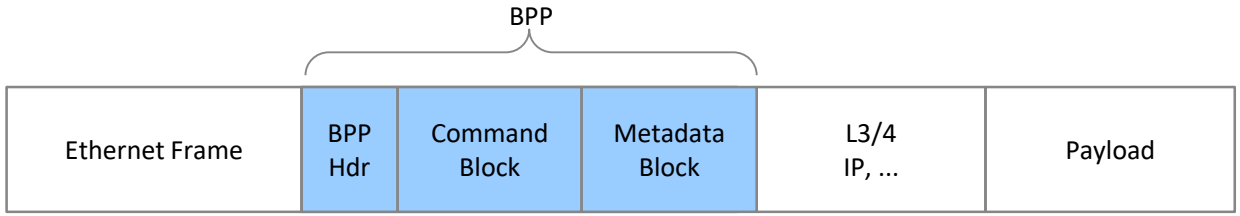
BPP Block and BPP Packet Structure
L 3.5 option



When would you deploy L3.5?

- End-to-end applications
- Operational use cases for L3 networks (e.g. Operational Flow Profiling etc)

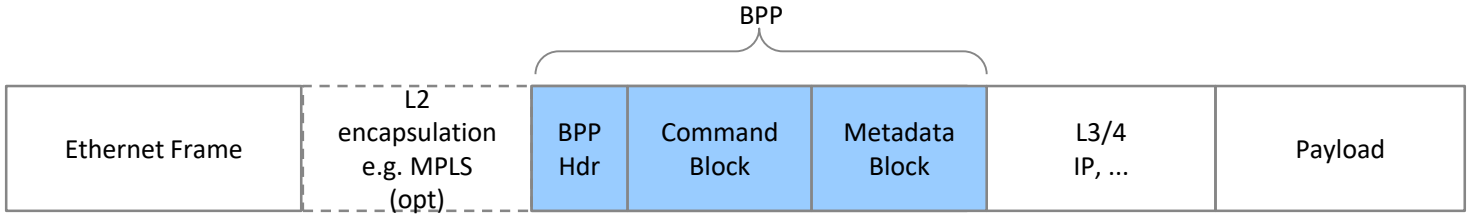
BPP Block and BPP Packet Structure
L 2.5 Option



When would you deploy L2.5?

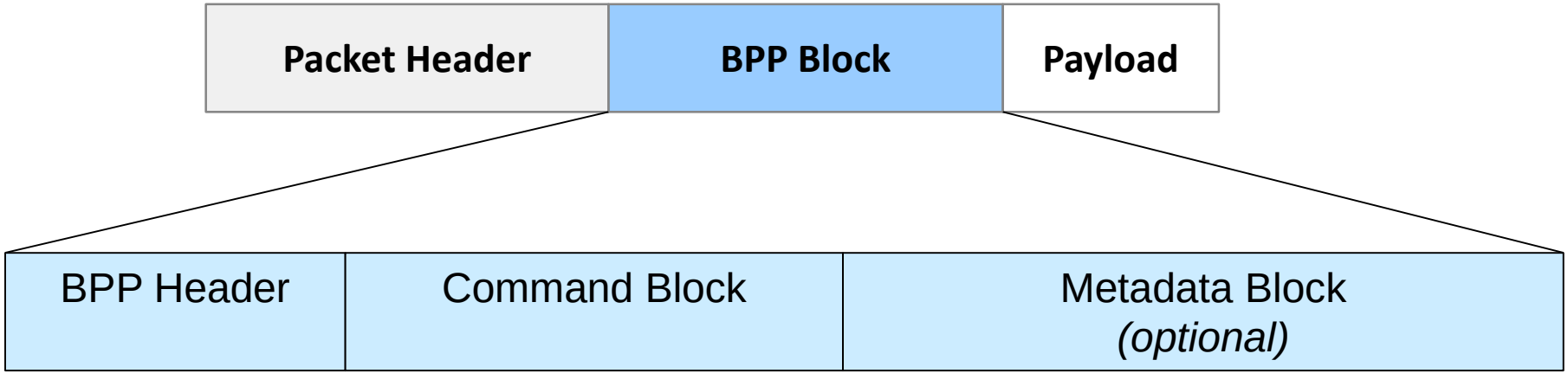
- Edge-to-edge within SP
- Access Networks / access aggregation
- L2 networks and implementation on switches
- High-precision applications: no need to inspect / act on IP processing
- TTL may still be needed, which is supported in BPP extension header

BPP Block and BPP Packet Structure
L 2.5 Option

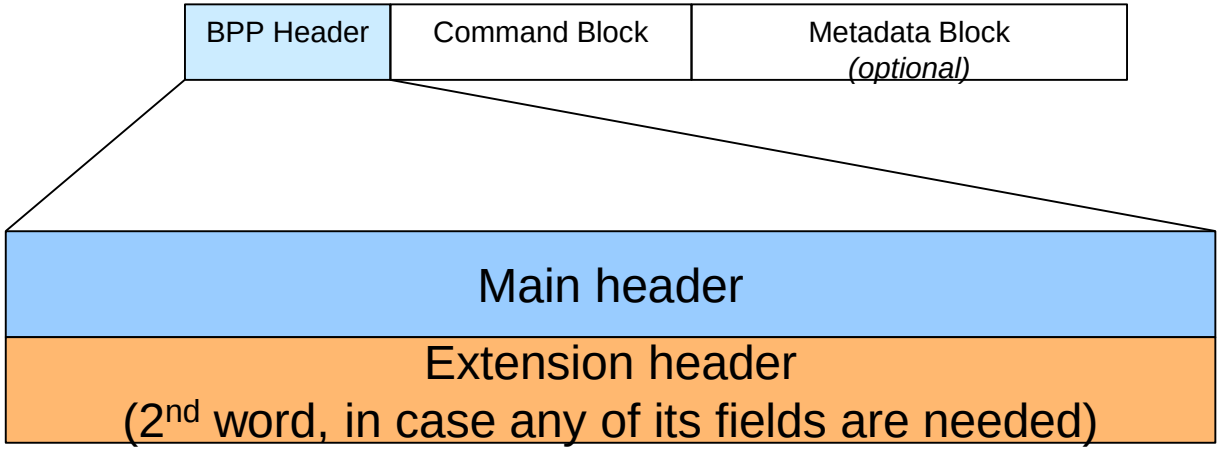


- BPP as MPLS Extension Header
- Currently MPLS is limited by additional functionality w.r.t metdata
- This option enables commands and metadata with MPLS/SR-MPLS forwarding
- Still under further investigation

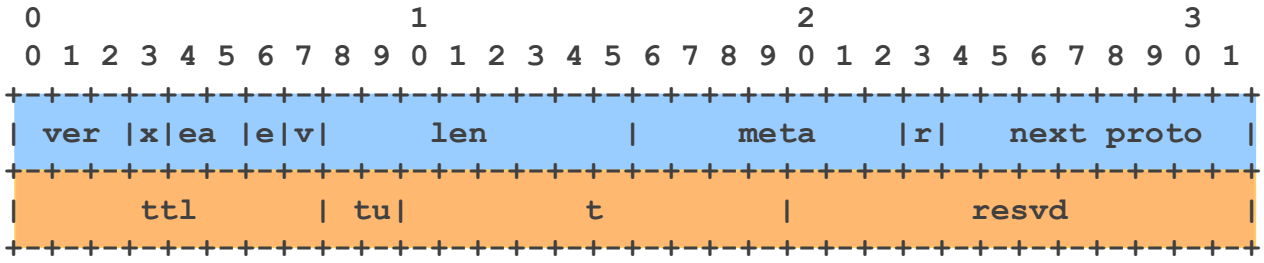
BPP Block Structure



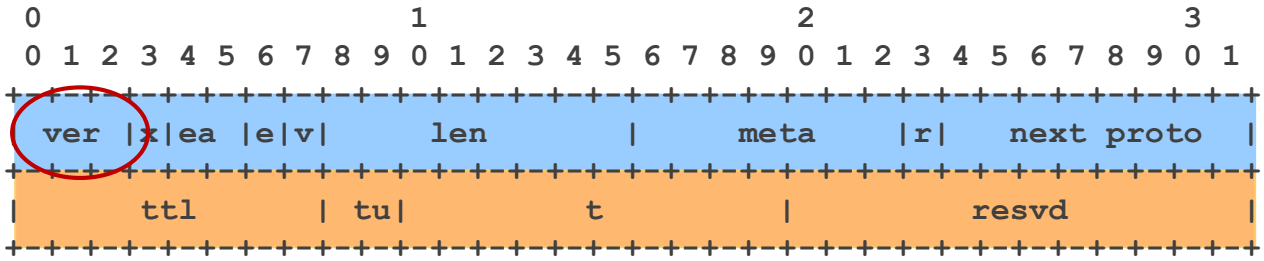
BPP Header


BPP Header
Command Block
Metadata Block
(optional)
Main header
Extension header
(2nd word, in case any of its fields are needed)

BPP Header

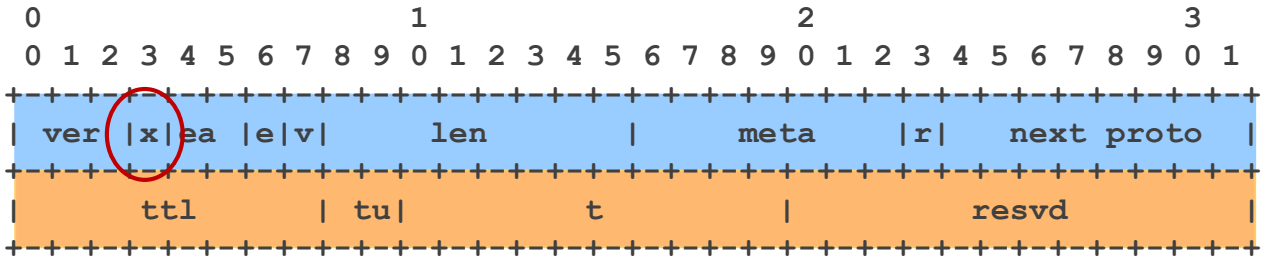

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
ver
x
ea
e
v
len
meta
r
next proto
ttl
tu
t
resvd

BPP Header



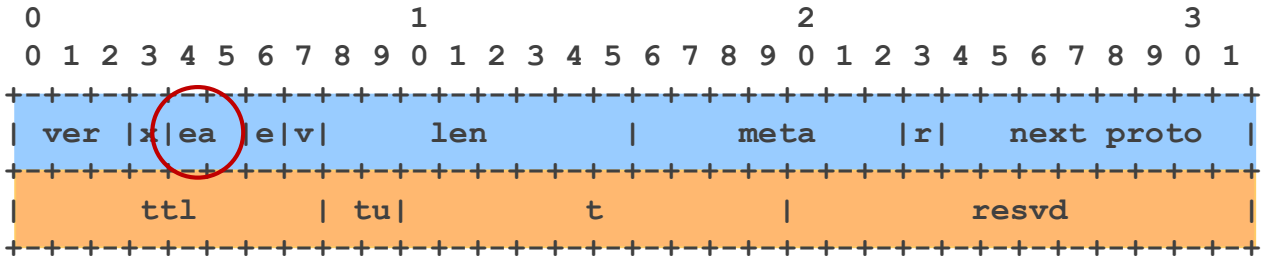
- Version: BPP version (0 initially)

BPP Header



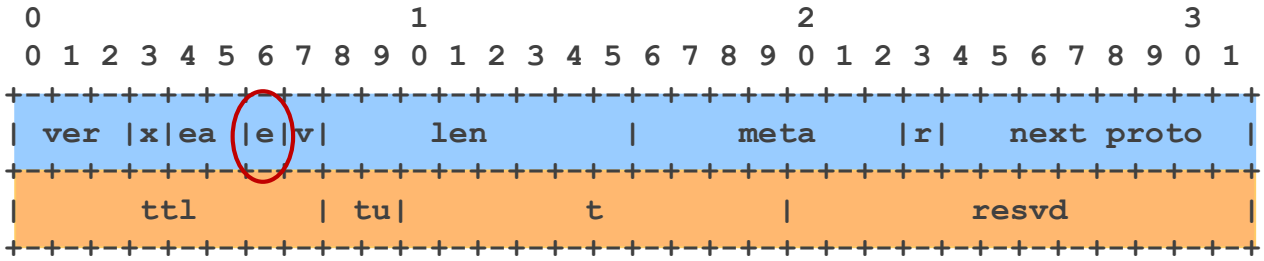
- x: Extension header present
  - Set to 1 if yes, 0 if no (header length in that case only 4 octets)
  - Cannot be changed by subsequent hops

BPP Header



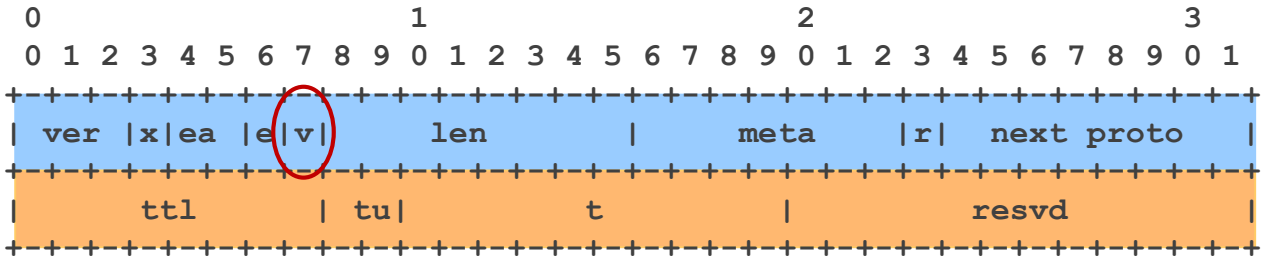
- ea: Error Action
  - Specify what to do when an error occurs when processing BPP commands
    - 00: ignore (i.e., continue as if no error had occurred)
    - 01: drop packet
    - 10: skip (skip any further BPP processing; subsequent nodes do not need to process BPP block)
    - 11: reserved

BPP Header



- e: prior errors on path
  - 0 when packet is originated
  - Node may set this to 1 once an error occurs (cannot be reset to 0)
  - Facilitate troubleshooting
  - Certain algorithms may benefit from allowing downstream nodes to be aware of earlier errors

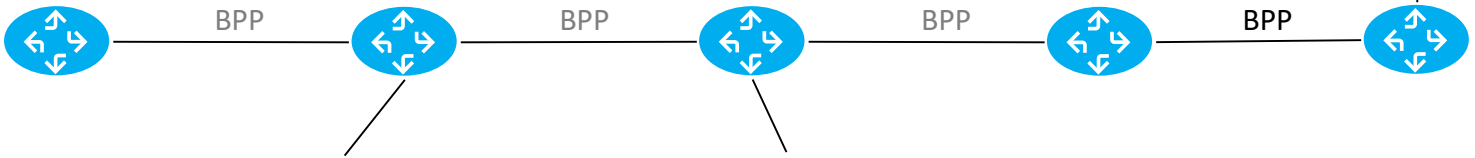
BPP Header



- v: verbose error reporting (1) or not (0)
  - Record details of any errors to facilitate diagnostics:
    - Which command / condition / action / parameter
    - Which type of error (invalid data item, time exceeded, read or write error, unsupported action / capability, ...)
    - At which node
  - Record on node (as part of statelet, and/or in separate log)
    - Option to add error info to metadata ffs (size considerations)
  - Export as part of syslog or using other means, details TBD

Example: Error handling

Monitor at egress node
to assess proper functioning.
may request “verbose” in case troubleshooting needs to occur

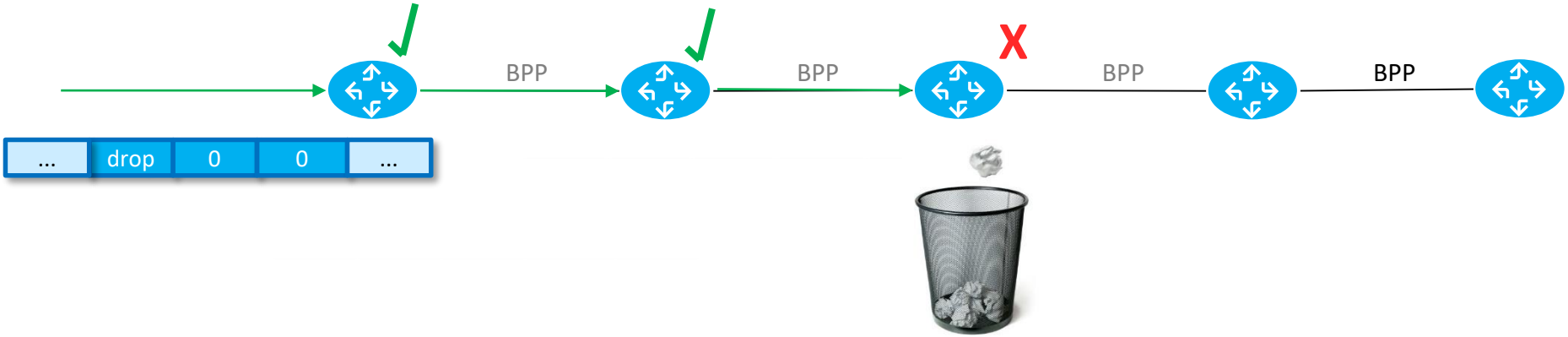


Error occurs during processing
e.g. timing exceeded,
malformed command,
data item unavailable,...

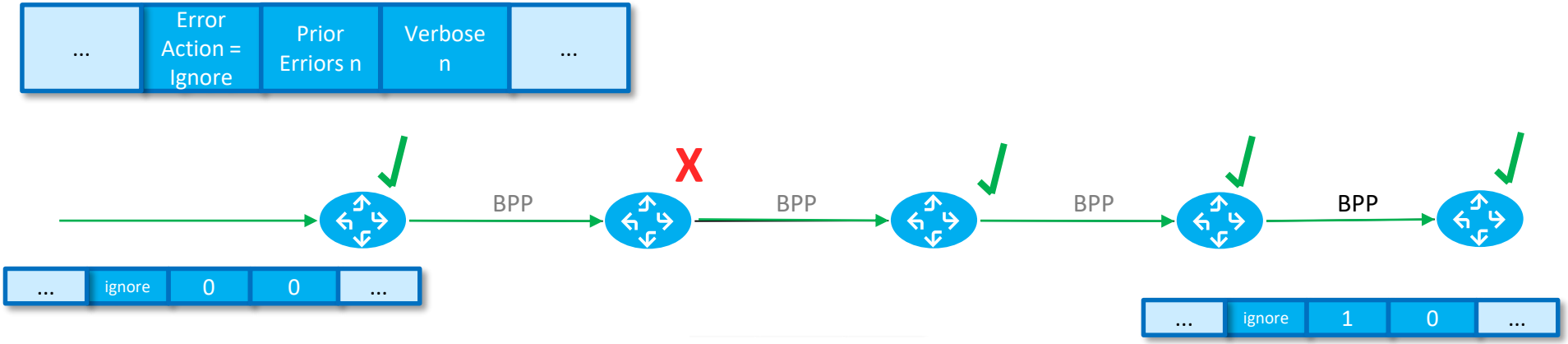
Commands at subsequent nodes can be performed conditional on no prior error (error action = skip)

- Perform error action
- Set prior errors flag
- Log error details into statelet, packet MD if v-flag is set

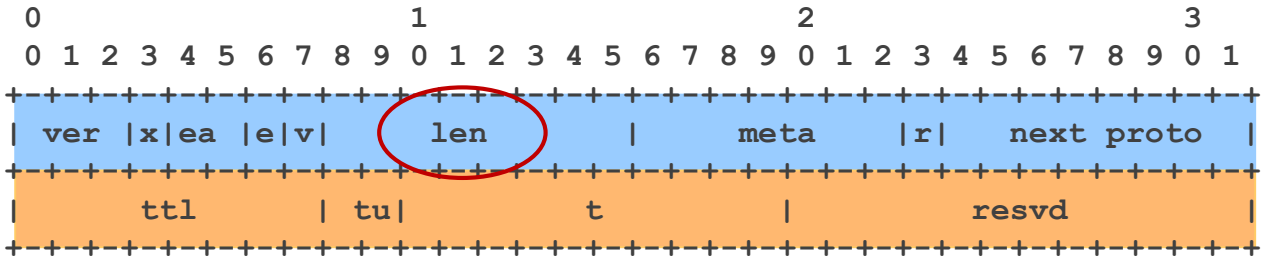
Example: Error handling



Example: Error handling

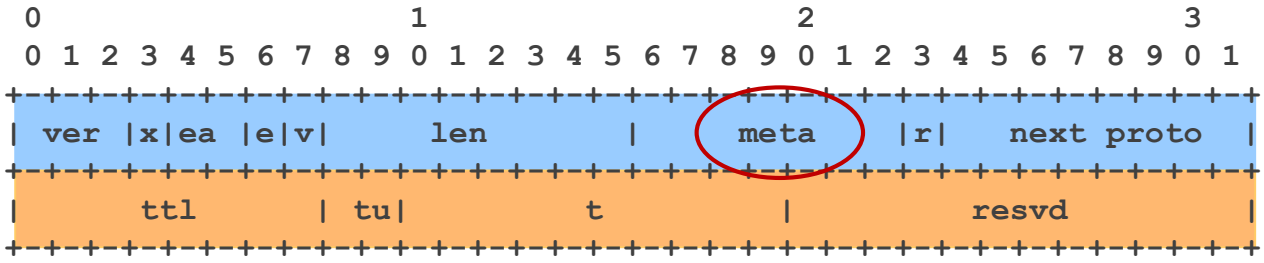


BPP Header



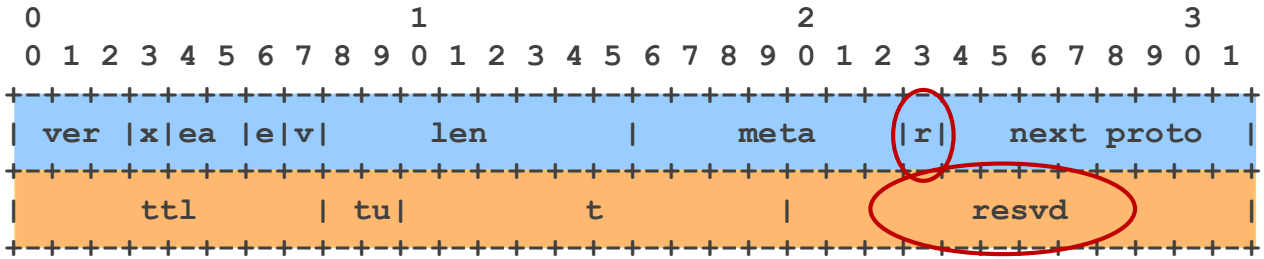
- len: Length of the BPP Block (in octets)
- Discussion: Fixed vs variable size?
  - More rigid structure will facilitate implementation
    - Earlier considerations allowed for less length options: S/M/L/XL
    - Even treatable as sister protocols: BPP-S, BPP-M, BPP-L, BPP-XL, each fixed size
    - Each sister protocol follows the same grammar / design pattern
  - Variable size offers more flexibility, less “waste”
  - More implementation experience is needed

BPP Header



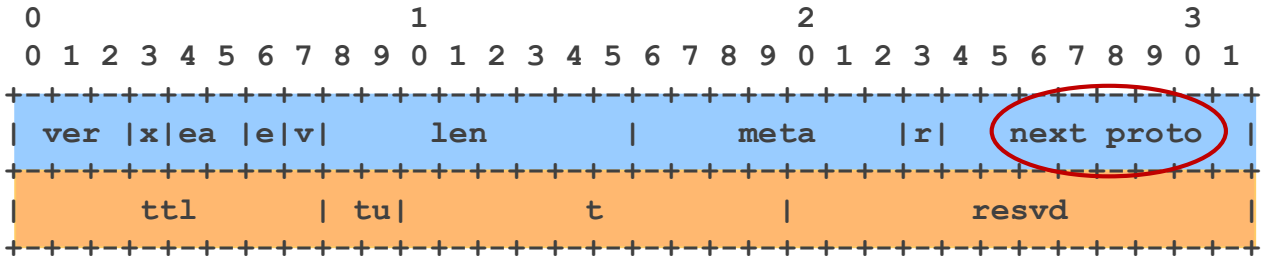
- Indicates offset of metadata, if any
  - Specified in octets from beginning of BPP Block
  - Set to 0 if no metadata.
  - Notes:
    - Word alignment not required
    - No context dependency on extension header being present or not
    - metadata offset cannot exceed len

BPP Header



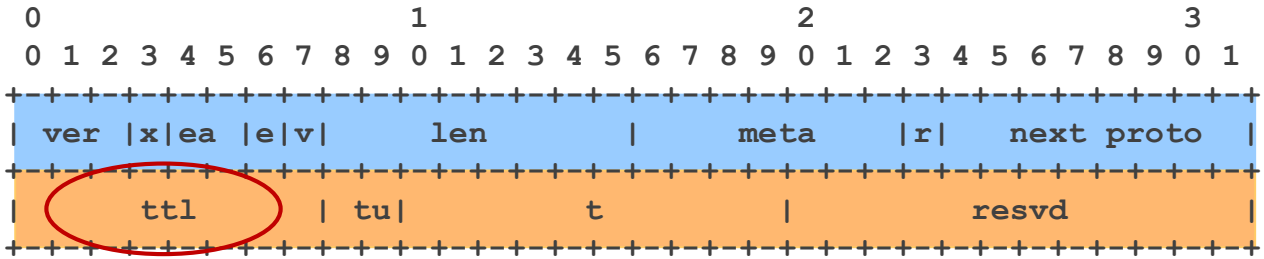
- r / resvd: Reserved for future usage; ignore / set to 0

BPP Header



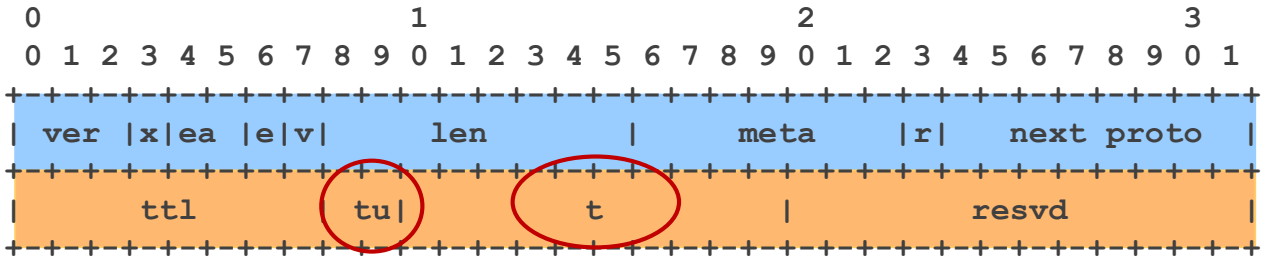
- Indicates next protocol (i.e. subsequent protocol header)
  - corresponds to protocol / next header field

BPP Header



- ttl: Time to Live
  - Analogous semantics as IP TTL:
    - Decremented at every BPP hop
    - Packet is discarded when TTL is 0
  - Useful in L2.5 deployments, redundant when used with L3.5 deployments

BPP Header

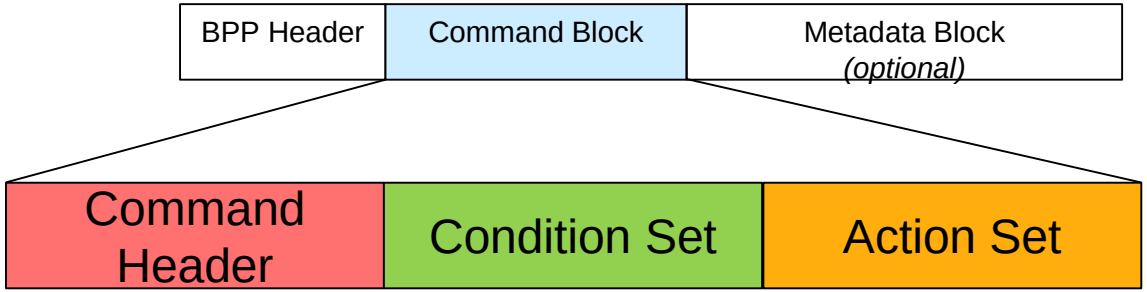


- Timing constraint
  - BPP commands processing is aborted if not completed within timing constraint, with corresponding error flags set
  - Timing is specified through two fields:
    - tu: unit of resolution:
      00: .1ns (100ps); 01: 10 ns, 10: 1 µs, 11: 100 µs
    - t: 10 bits to express the actual value (0: no constraint)
    - Allows to specify constraints from 100 ps to 100 ms
      (the latter obviously intended for control applications, not line rate)

Command Block

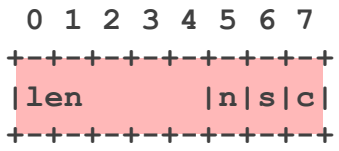
- Command Block consists of one or more commands
- No separate Command Block Header
- Commands contain flag to indicate whether followed by another command (or by metadata, if any)
- Commands must not exceed beyond metadata offset and BPP Block

Command Structure



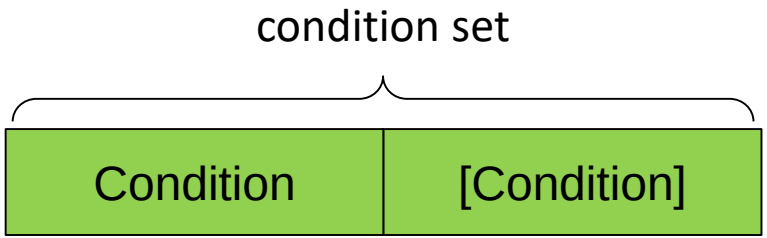
- Command Header with housekeeping information
- Optional condition set must evaluate to “true” for actions to be applied (should not include more than 2 conditions)
- All actions in action set are applied

Command Header



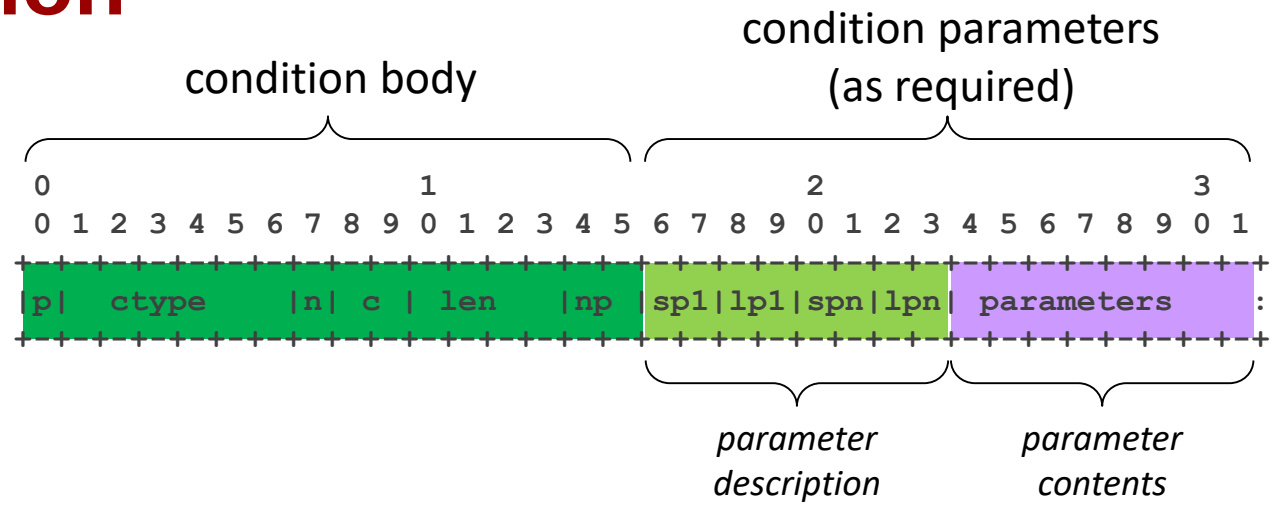
- len: length of commands (in octets)
- n (next): is there a subsequent command after this
  - if not, the command is followed by metadata, if any
- s (serialization): does next command need this command to first finish
  - ignore if n is 0
- c (condition): is there a condition set
  - 1: command is conditional; CHeader is followed by condition set
  - 0: command is unconditional; CHeader is followed by action set

Condition set

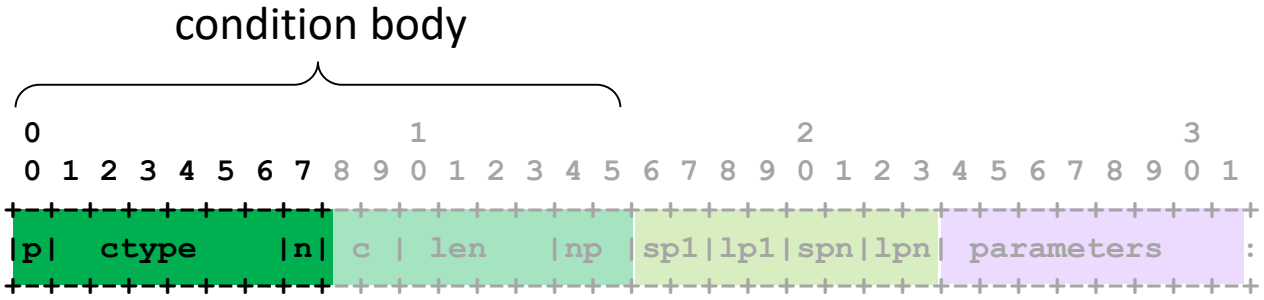


- Set of one or more conditions (recommended: not to exceed 2)
- No separate condition set header
  - condition indicates if another condition follows and how to combine (and / or)

Condition

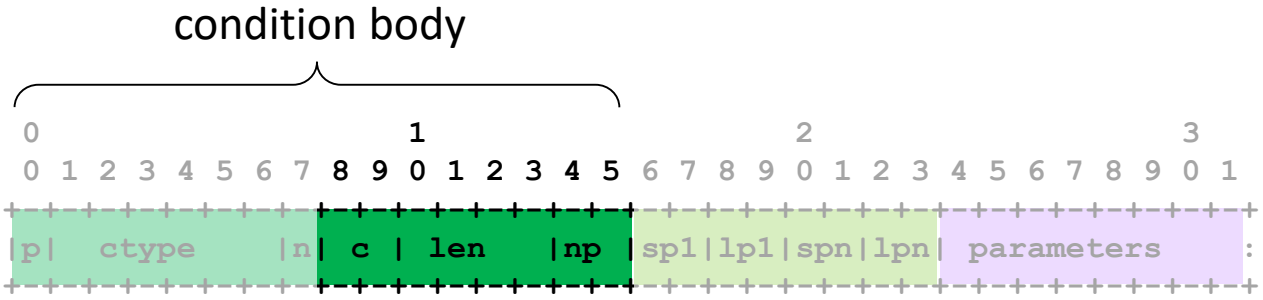

condition body
condition parameters
(as required)
0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
|p| ctype |n| c | len |np |sp1|lp1|spn|lpn| parameters :
parameter description
parameter contents

Condition



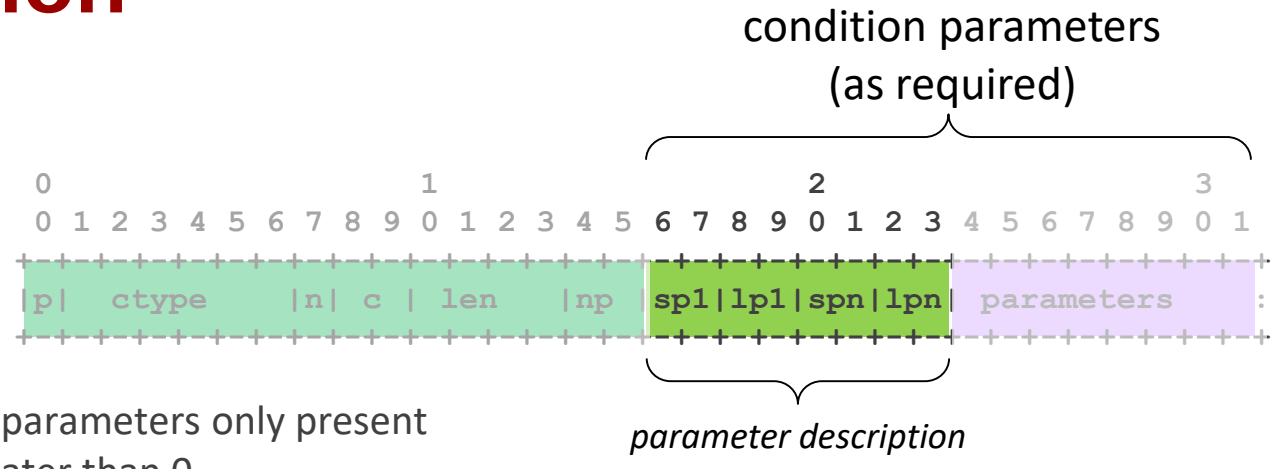
- p: is ctype proprietary (1) or part of built-in set (0)
- ctype: type of condition
  - Comparators: LE, LT, EQ
  - Existence checks: EXISTS
  - Set operators: ISELEMENT, ISSUBSET
  - Convenience functions: ISZERO
- n: negate (Boolean “not”) condition

Condition



- c: is there a subsequent condition, and how to chain
  - 00: no more condition
  - 10: next condition, AND applies
  - 11: next condition, OR applies
- len: length of condition (incl body)
- np: number of parameters
  - Most conditions have 1 or 2 parameters
  - np is implied by ctype, but avoids context dependency

Condition



- Condition parameters only present if np is greater than 0
- Parameter description is a "template" for parameters 1: sp_1, lp_1 and n (i.e., 2..3): sp_n, lp_n
  - sp_x: size of the parameter
    - 01: 1 octet; 02: 2 octets; 10: 4 octets
    - 00: parameter has a parameter header in which length is encoded
  - lp_x: location/category of parameter (overridden by parameter header if sp_x is 00)

00: raw value	01: metadata offset	10: statelet offset	11: data item/ data ID

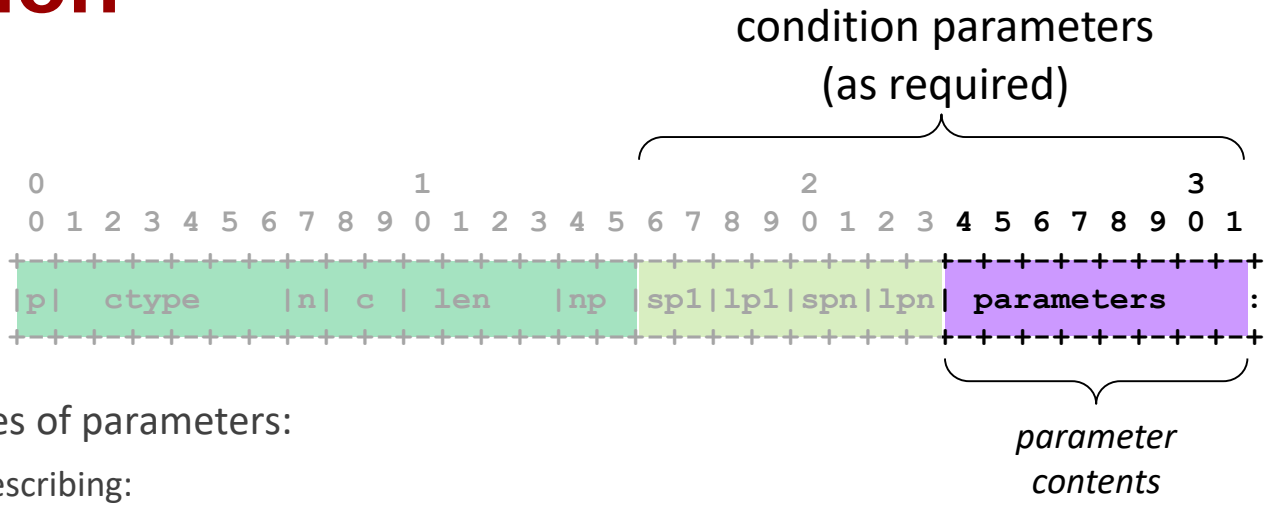
Data Items

- Data Items can refer to
  - Packet properties: Packet Length
  - Header fields: e.g. TTL, src IP, dest IP, ...
  - Well-defined BPP Data Items
    - Statelet Data Items: Previous Packet Metadata (e.g. time, seqno)
  - Flow telemetry:
    - IOAM data items: egress queue depth
    - ingress interface, egress interface
    - ingress interface stats, egress interface stats
  - Flow statistics
    - IPFIX/Netflow Information Elements
  - Other operational data (incl node ID)
  - Other data items learned from the control plane

Registry considerations

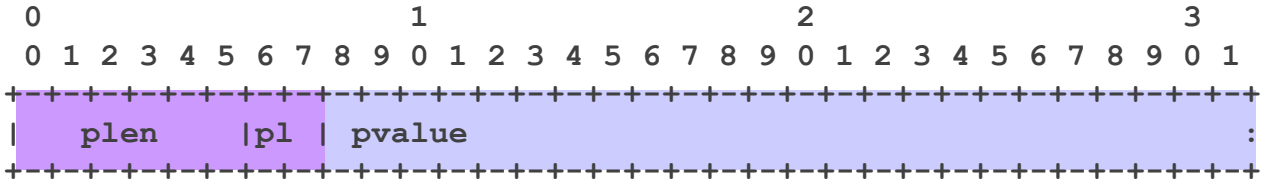
- Leverage different registries / identifier types
  - IPFIX IEs, IOAM, OID
  - BPP
  - Enterprise-ID (proprietary)
- Identify data items per tuple
  - Data Item Category ID
  - Data item ID (within category)
- BPP registry defines items that must be supported
  - All others are optional
  - May lead to some duplicates
  - Data item “misses” to be indicated by corresponding error flag (“unknown data item”)

Condition



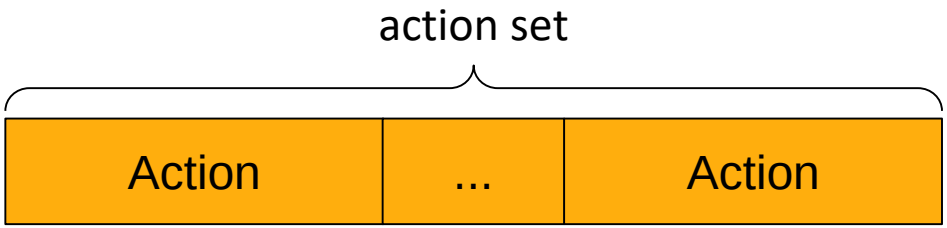
- 2 categories of parameters:
  - Self-describing:
    parameter contents includes parameter header
    describing length, location, etc
  - Template-defined: sp_x and lp_x define size and location; parameter contents includes raw value only (not even delimiters) – motivation: save encoding space
  - Data type (e.g. interpretation as uint8/16/32 etc) implied by condition type

Self-describing parameters



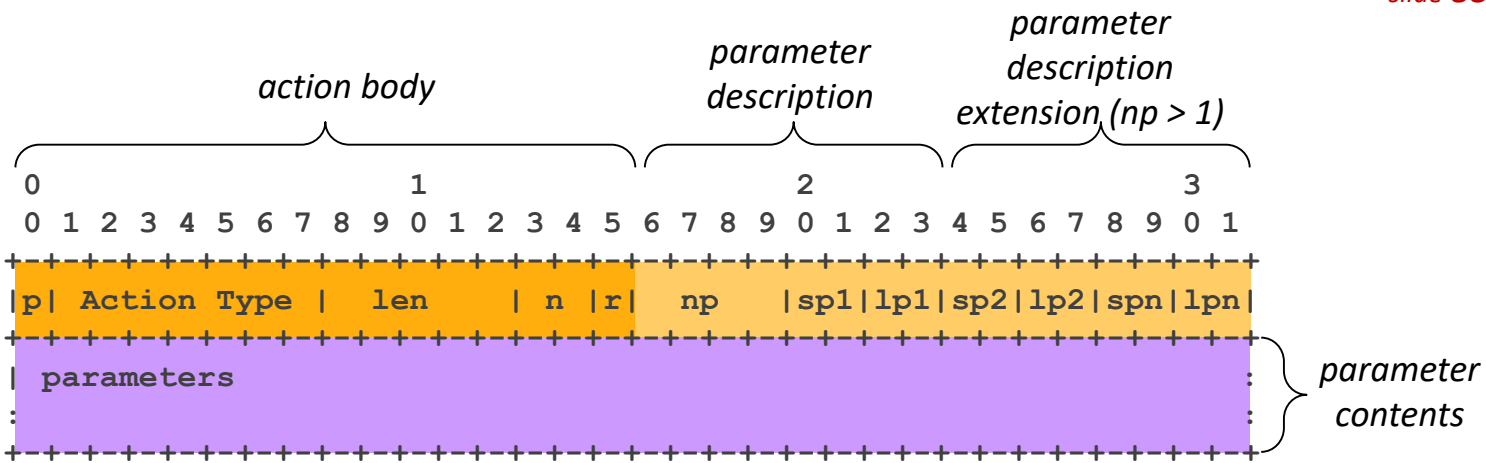
- Parameter header indicates length and category/location
  - plen: length in octets (incl parameter header)
  - pl: parameter category/location
    00: raw value, 01: metadata offset, 10: statelet offset, 11: data item
- No data type, hence no full-fledged TLV
  - Comparator and actions impose expected data type on parameter value
  - Programming model does not require type safety (so trade it off for overhead)
    - BPP ingress edge responsible to construct well-formed BPP Block (including soundness of commands)
    - Mismatches will only affect the own packet / flow

Action Set



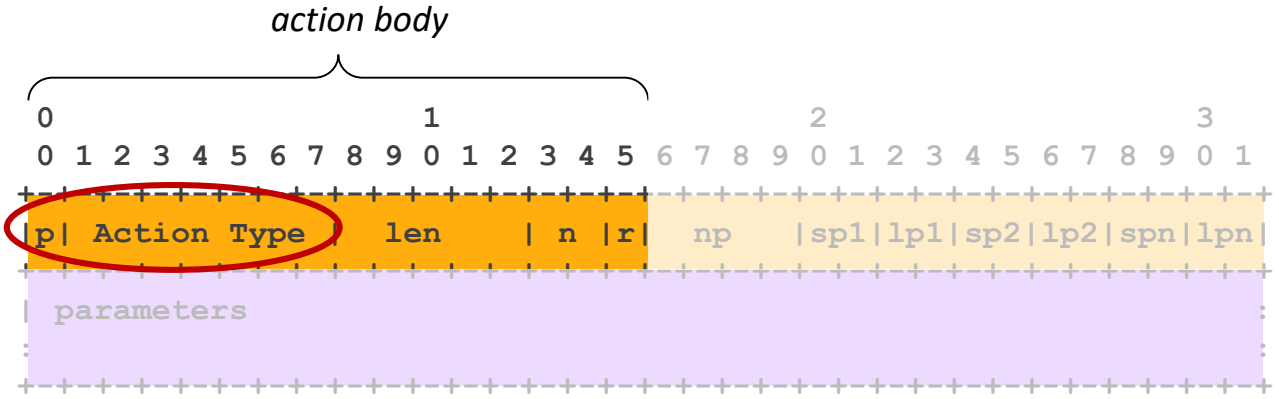
- Set of one or more actions
- No separate action set header
  - analogous to condition set
  - action header indicates if another action follows and if it needs to be serialized

Action



- p: is Action Type proprietary (1) or part of built-in set (0)
- Action Type: Type of Action
  - Catalogue to follow
- len: Length of the action (in octets, including parameters)
- n: is there a subsequent action
  - 00: no subsequent action
  - 10: subsequent action, serialized
  - 11: subsequent action, can be parallelized

Action



- p: is Action Type proprietary (1) or part of built-in set (0)
  - Management of registries of proprietary action types & their deployment at network devices is responsibility of the network provider
  - Most likely will involve a controller or management application to deploy custom actions (and manage their lifecycle) consistently across the network
- Action Type: Type of Action

Action catalogue: Primitives that act on packet as a whole

- Drop
  - Drop the packet
- Strip-BPP
  - Remove BPP Block from packet
- Swap (par1, par2)
  - Items referred to by par1 and par2 will swap their values
  - Parameters can refer to metadata parameters, statelet items, and header fields
- Checksum-update
  - Needed in case case header fields were changed
- Break
  - Stop any further BPP command processing

Action catalogue: Primitives that act on packet as a whole (contd.)

- Actions related to buffering
  - Start-Buffer
    - Initialize buffer and buffer packet instead of forwarding packet immediately
    - Packet is forwarded/played out in order it was buffered when rel-buffer command is received
  - Cont-Buffer
    - Buffer packet instead of forwarding packet immediately
    - Buffer has already been initialized
  - Rel-Buffer (p4arameter)
    - Parameter indicates how many packets are expected to be in the buffer
    - If number does not match, error flag is set (option to wait for missing packets in case of parameter indicating greater number than the actual)

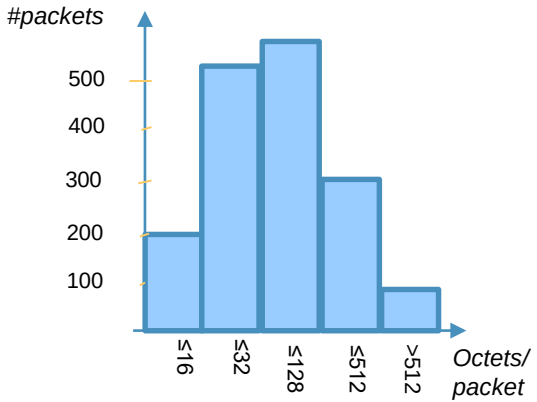
Action catalogue: primitives that manipulate metadata

- Increment / decrement (parameter)
  - Parameter refers to metadata in packet or statelet
  - Interpreted as uint; size per parameter description
- Increment-by / decrement-by (par, val)
  - increments *par* by *val*
  - par cannot refer to a built-in data item
  - val can refer to metadata in packet or statelet, a data item, or an actual value
  - par and val interpreted as uint; par over- or underflows result in error
- Set (par1, par2) *(assignment operator)*
  - This is the assignment operator; sets par1 to the value of par2
- Set-to-0 (parameter)
  - Frequently used convenience function – set parameter to 0

Action catalogue: Advanced actions

- Special applications may require advanced actions
- Example: Match-queue (lb, ub, pd)
  - Select suitable packet queue according to timing parameters
  - Use for timing-aware forwarding applications (→ use case)
  - lb and ub refer to lower and upper bound of acceptable end-to-end delay
  - pd refers to packet delay experienced so far, used to offset lb and ub
- Custom actions can be defined
  - Use “p” flag to indicate proprietary extension
  - Deployment of action instrumentation in BPP nodes as part of software deployment framework

Action catalogue: primitives that manipulate metadata



- Histogram update
  - Update one in a set of counters to maintain a histogram of statistics
  - Use to aggregate statistics (e.g. OFP use case → later)
  - Set of commands with the following pattern:
    histo-***x***-uint***y***-incr (target, bb[x-1], histogram)
    - x: number of buckets in the histogram
    - y: size of the counters
    - parameter "target": data item whose value determines which counter will be incremented
    - parameter "bb[x-1]": a set of x-1 parameters that specify bucket boundaries;
      bucket counter to be updated determined by comparing boundaries with target
    - histogram: a set of x counters of size y
- For future consideration: Set operations (e.g. adding an element to a set)

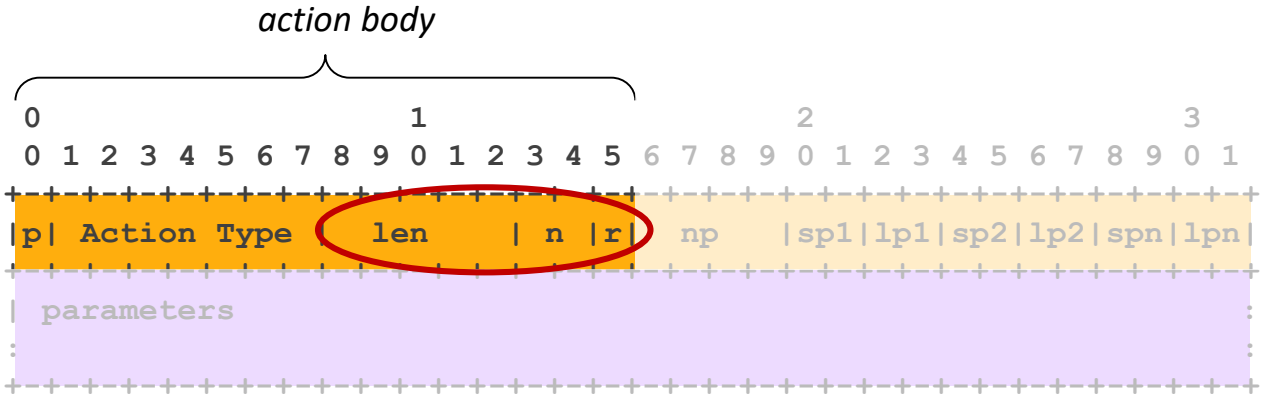
Action catalogue: other considerations

- Primitives that act on state: see stateful extensions
- For future consideration: actions that act on packets
  - Actions to tag a packet
  - Actions to mark a packet
- For future consideration: primitives that trigger actions on device, e.g.
  - Allocation and reservation of resources
  - OAM actions (e.g. start additional measurements)
  - Data export
- For future consideration: primitives that act on flows, e.g.
  - Deduplication
  - Reordering
  - Drop detection

Note: certain dependencies on header fields / wrappers apply

Notes:
- *flow primitives may require retaining considerable flow state, with various implications including buffering and blocking*
- *May also impose need for predefined metadata, e.g. seq no.*

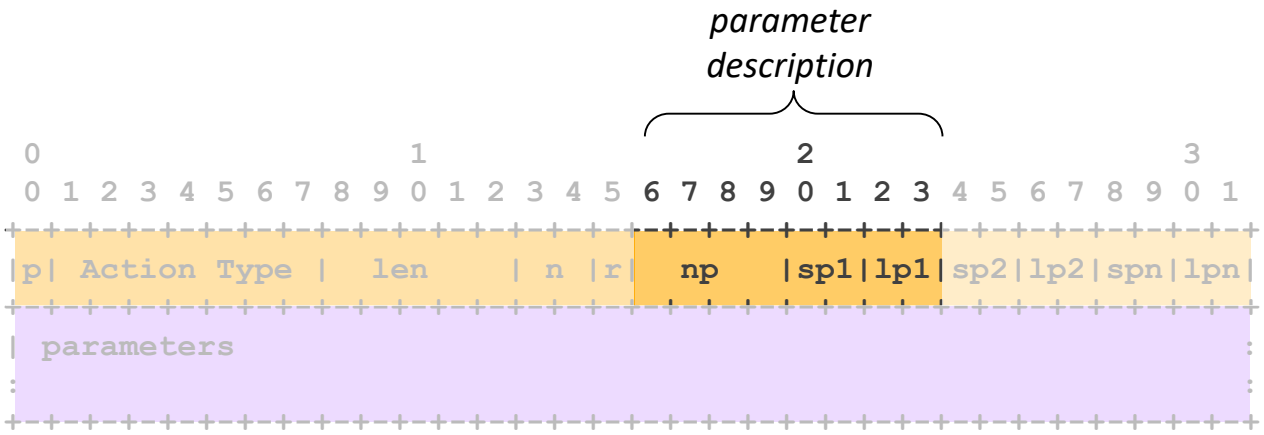
Action



- len: Length of the action (in octets, including parameters)
- n: is there a subsequent action
  - 00: no subsequent action
  - 10: subsequent action, serialized
  - 11: subsequent action, can be parallelized
- r: reserved

Key to facilitating performance optimizations

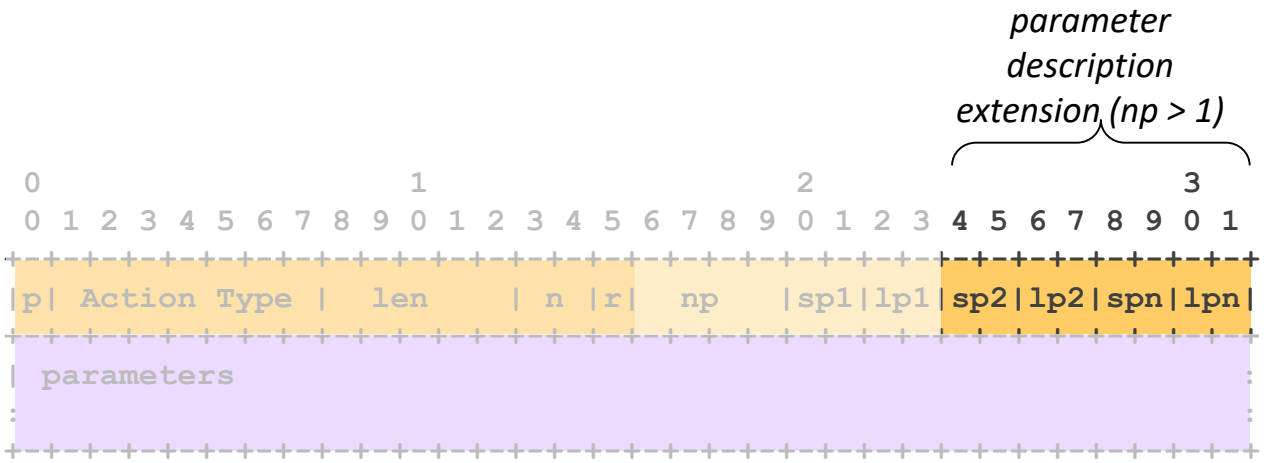
Action



- np: number of parameters
  - Actions can have a greater number of parameters
- sp_1: size of parameter 1 (analogous to conditions) (n.a. if np==0)
  - 01: 1 octet; 10: 2 octets; 11: 4 octets
  - 00: parameter has a parameter header in which length is encoded
- lp_1: location/category of parameter 1 (analogous to conditions) (n.a. if np==0)

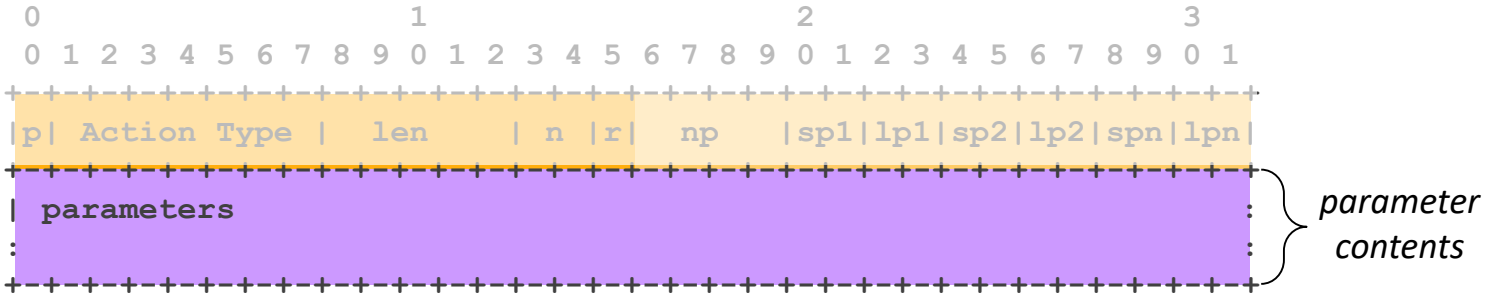
00: raw value	01: metadata offset	10: statelet offset	11: data item/ data ID

Action



- Parameter description extension only needed if action has more than 1 parameter (save one octet otherwise)
- sp_1, lp_2 analogous to sp_1, lp_1 for parameter 2
- sp_n, lp_n analogous for all other parameters (3..n), if applicable
  - If differentiation required, can have defined by parameter header – set sp_n to 00

Action



- Parameters analogous to condition parameters
  - Self-describing:
    parameter contents includes parameter header
    describing length, location, etc
  - Template-defined: sp_x and lp_x define size and location; parameter contents includes raw value only (not even delimiters) – motivation: save encoding space
  - Data type (e.g. interpretation as uint8/16/32 etc) implied by action type

Metadata

- Metadata to guide processing, such as tags or labels
  - Information about the packet (e.g. tags, classifiers, assertions)
  - May contain declaratives (e.g. what to accomplish, not how)
  - Examples: security material, classification tags, identity metadata, SLO data…
- Alows to deploy context as part of a statelet (incl. cached commands)
  - See stateful extensions
- Output from BPP processing
  - From different nodes: tracing, telemetry, ...
  - Aggregated: end-to-end measurements+telemetry

Metadata

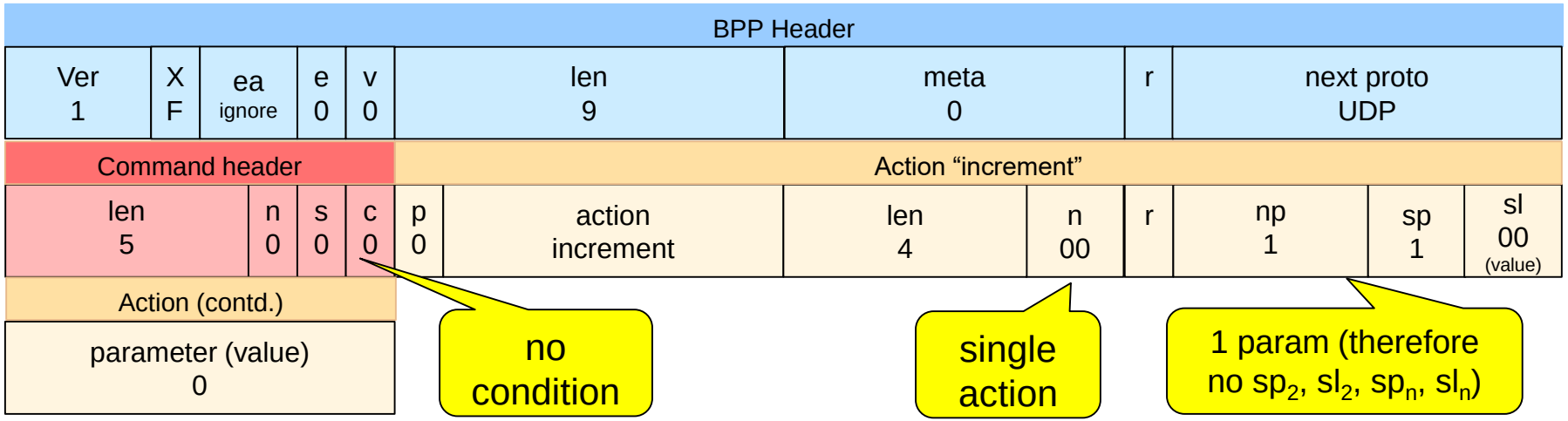
- Unstructured Metadata deemed initially sufficient
  - Semantics, meaning of metadata up to BPP Commands & Programming
    - Offset and length per BPP Command Parameters
    - Casting of raw metadata contents into data type per BPP Commands
  - In effect “raw packet storage”, with interpretation to BPP commands and applications
- Structured Metadata ffs
  - E.g. TLV, Attribute / Value pairs, etc
  - Needed when other programs need to interpret, consume, act on metadata
    - E.g. recording, exporting of BPP metadata at the edge
    - Interpretation of metadata by other programs

Example 1

- Count the number of hops being traversed
- Algorithm:

specify how to initialize at “hop 0”

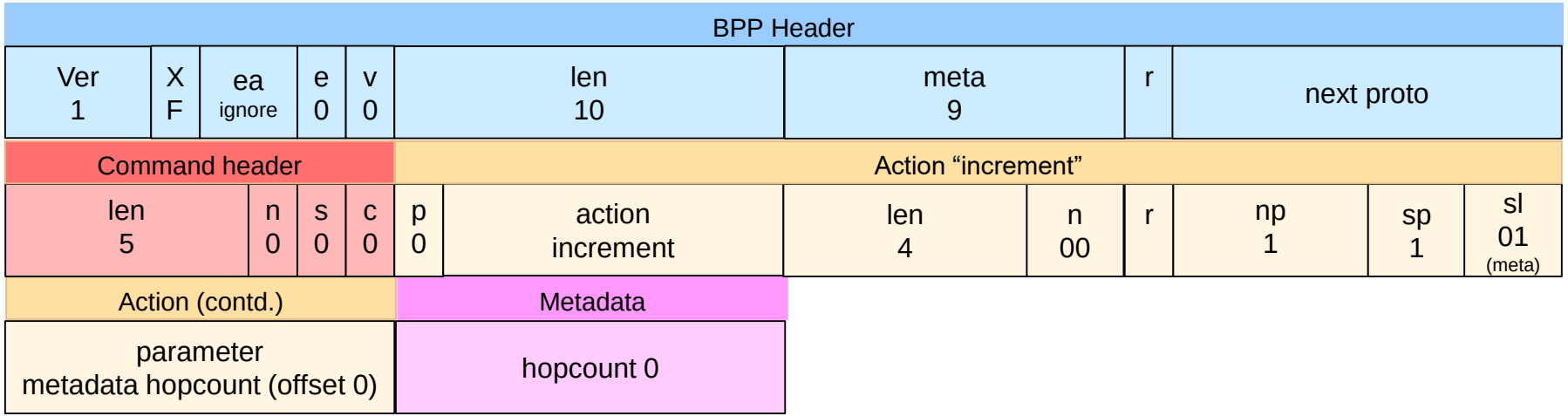
```
Cmd1:   cond -;
        action: increment(par.val "hopcount"(size 1, init 0));
```



Example 1 (preferred alternative)

- Carry hop count as metadata
  - allow other commands to refer to it
  - avoid need to modify commands

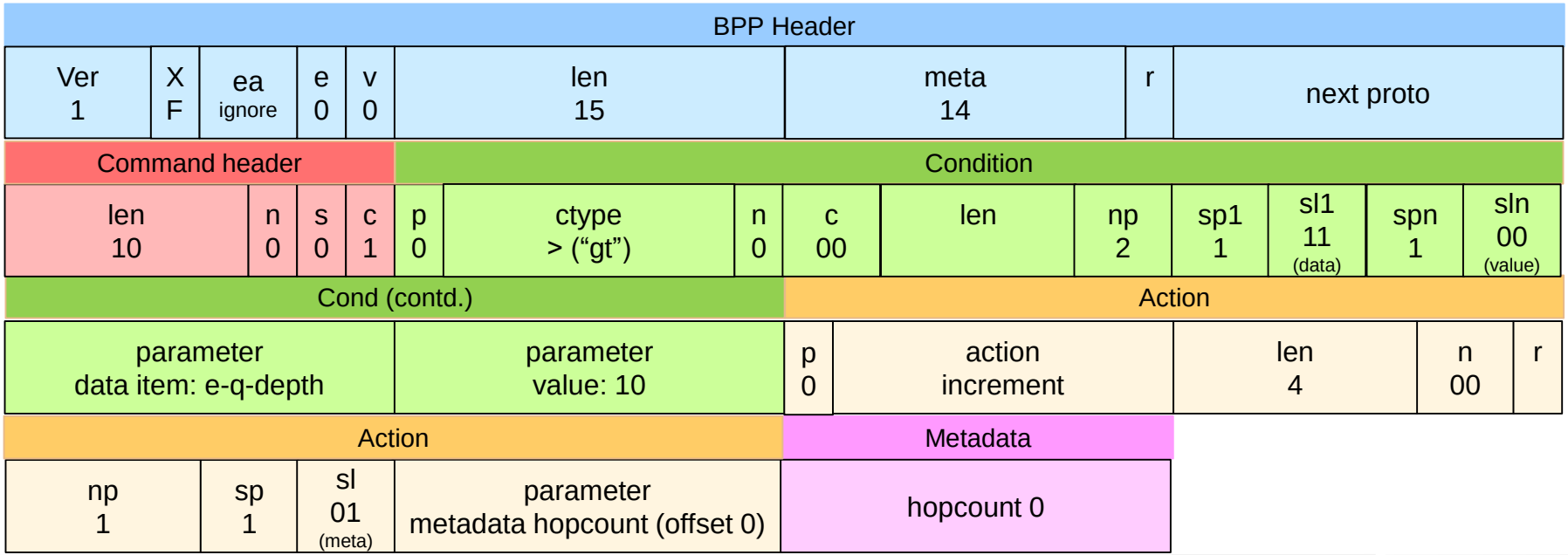
```
Cmd1:
  cond -;
  action: increment
    (par.meta "hopcount"
      (offset 0, size 1, init 0));
```



Example 2

- Count hops on which egress queue depth exceeds 10 *(this adds a condition)*

```
Cmd1:
  cond: gt(par.data e-q-depth,
           par.value 10)
  action: increment
    (par.meta "hopcount"
      (offset 0, size 1, init 0));
```



Example 3

- Add egress queue depth of nodes along the path

```
Cmd1:
  cond -;
  action: increment-by
    (par.meta "total-q-d"
      (offset 0, size 1, init 0),
     par.data "e-q-depth");
```

(larger size could be chosen)

BPP Header								
Ver 1	X F	ea ignore	e 0	v 0	len 12	meta 11	r	next proto

Command header				Action “increment-by”								
len 7	n 0	s 0	c 0	p 0	action increment-by	len 6	n 00	r	np 2	sp1 1	sl1 01 (meta)

Action (contd.)						Metadata
sp2 1	sl2 11 (data)	spn 0	sln 0	parameter metadata total-q-d (offset 0)	parameter data item: e-q-depth	total-q-d 0

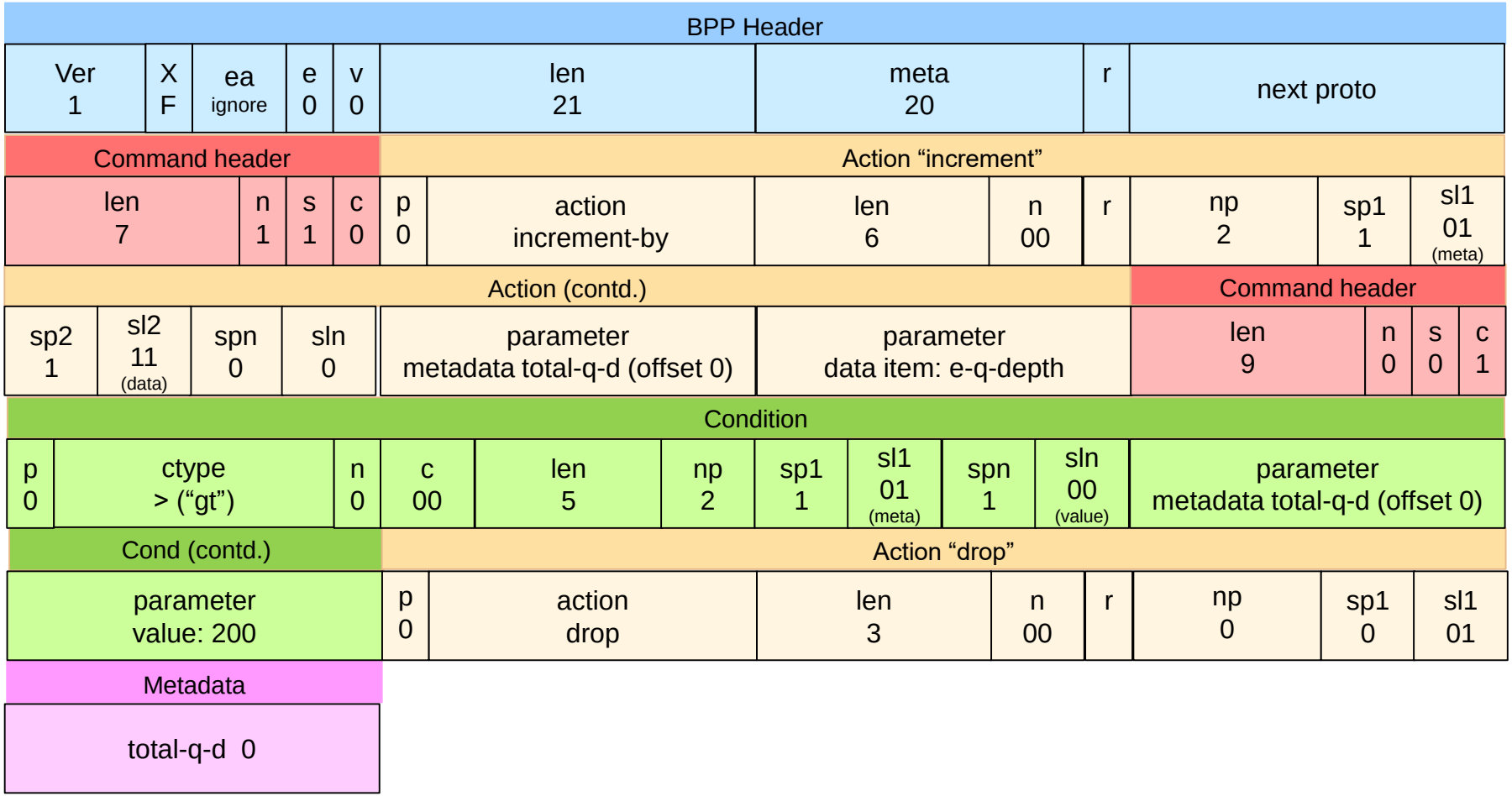
Example 4

- Add egress queue depth of nodes along the path and drop packet if sum of queue depths exceeds 200

need to perform update first before comparing

```
Cmd1:
  serialize-next;
  cond -;
  action: increment-by
    (par.meta "total-q-d"
      (offset 0, size 1, init 0),
     par.data "e-q-depth");
Cmd2:
  cond: gt(par.meta "total-q-d" (offset 0, size 1), par.value 200);
  action: drop;
```

used in comparison so no need to specify initialization here;

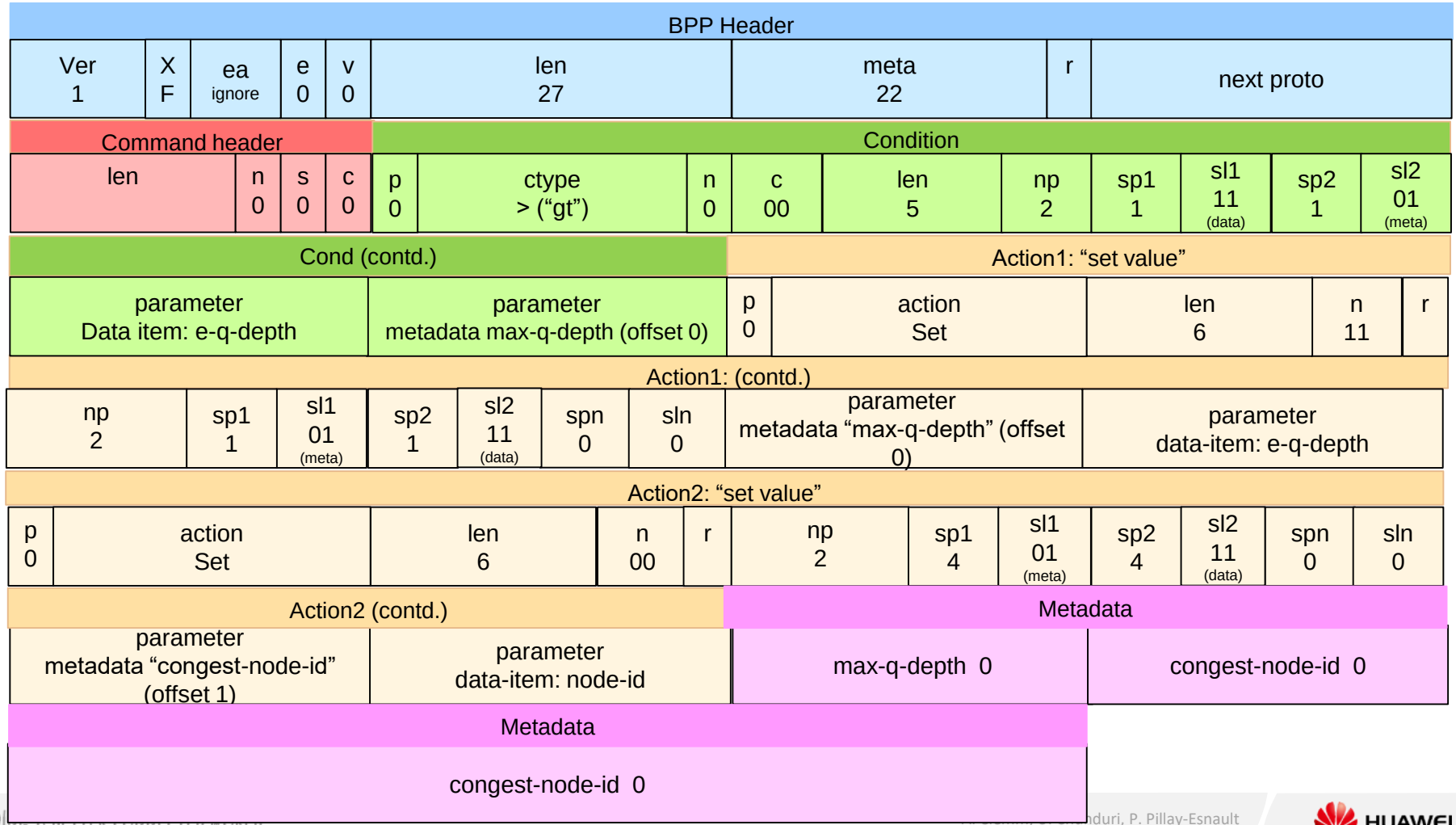

BPP Header
Ver 1
X F
ea ignore
e 0
v 0
len 21
meta 20
r
next proto
Command header
Action “increment”
len 7
n 1
s 1
c 0
p 0
action increment-by
len 6
n 00
r
np 2
sp1 1
sl1 01 (meta)
Action (contd.)
Command header
sp2 1
sl2 11 (data)
spn 0
sln 0
parameter metadata total-q-d (offset 0)
parameter data item: e-q-depth
len 9
n 0
s 0
c 1
Condition
p 0
ctype > (“gt”)
n 0
c 00
len 5
np 2
sp1 1
sl1 01 (meta)
spn 1
sln 00 (value)
parameter metadata total-q-d (offset 0)
Cond (contd.)
Action “drop”
parameter value: 200
p 0
action drop
len 3
n 00
r
np 0
sp1 0
sl1 01
Metadata
total-q-d 0

Example 5

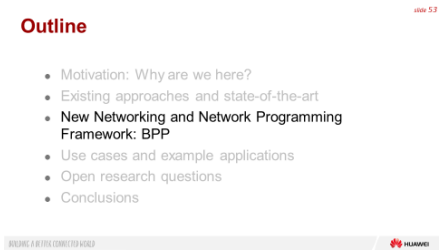
- Record Most Congested node (the largest queue depth) along the path

```
Cmd:
  cond: gt(par.data "e-q-depth"),
           par.meta "max-q-d"(offset 0, size 1));
  action: set
    (par.meta "max-q-d", offset 0, size 1, init 0),
     par.data "e-q-depth");
  action: set
    (par.meta "cong-node", offset 1, size 4, init 0),
     par.data "node-id");
```

can be parallelized


BPP Header
Ver 1
X F
ea ignore
e 0
v 0
len 27
meta 22
r
next proto
Command header
len
n 0
s 0
c 0
Condition
p 0
ctype > (“gt”)
n 0
c 00
len 5
np 2
sp1 1
sl1 11 (data)
sp2 1
sl2 01 (meta)
Cond (contd.)
parameter Data item: e-q-depth
parameter metadata max-q-depth (offset 0)
Action1: “set value”
p 0
action Set
len 6
n 11
r
Action1: (contd.)
np 2
sp1 1
sl1 01 (meta)
sp2 1
sl2 11 (data)
spn 0
sln 0
parameter metadata “max-q-depth” (offset 0)
parameter data-item: e-q-depth
Action2: “set value”
p 0
action Set
len 6
n 00
r
np 2
sp1 4
sl1 01 (meta)
sp2 4
sl2 11 (data)
spn 0
sln 0
Action2 (contd.)
parameter metadata “congest-node-id” (offset 1)
parameter data-item: node-id
Metadata
max-q-depth 0
congest-node-id 0
Metadata
congest-node-id 0

BPP: A New Networking and Network Programming Framework



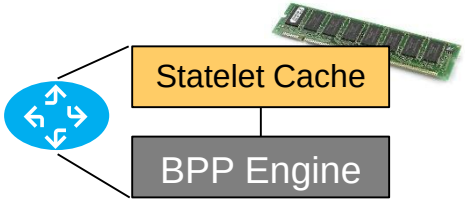
- BPP Framework overview
- BPP protocol: Blocks, Commands, Metadata
- Stateful extensions
- BPP programming model
- BPP engine, node infrastructure, deployment model

Stateful extensions

- Some applications can benefit from state, examples:
  - Applications operating on a flow:
    - Deduplication (e.g. of redundant packets due to resilience or performance reasons)
    - Real-time flow analytics (cp. Netflow): e.g. retain statistics for threat analytics
    - Volume-based accounting
  - Service-level measurements (e.g. inter-packet arrival delays)
  - Reservations
  - Accounting information
- BPP itself can benefit as well:
  reduce BPP packet size by caching BPP data carried in every packet
  - Selected metadata (e.g. SLO data)
  - BPP commands applicable to every packet on a flow

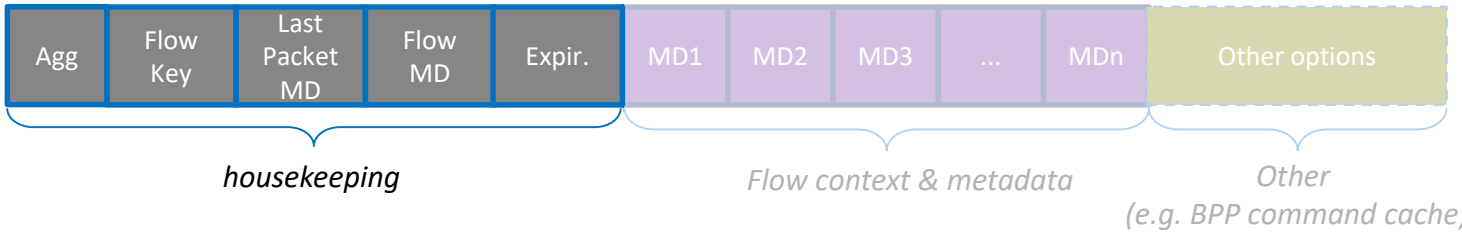
Statelets

- Statelets: on-device cache memory to retain programmable flow state
  - Example: reservations, measurements, SLOs
  - Bound to BPP flows packets via statelet (flow) key
- Glorified Programmable Flow Cache (analogous Netflow, IPFIX)
  - Analogous addressing via flow key
  - Difference: BPP commands can interact with statelet contents; contents can be programmed
- Interact with BPP packets
  - Read from, write to, compare, increment/aggregate statelet metadata
  - Options: non-blocking (update after forwarding), cached commands
- State mgmt. considerations apply
  - Reclaim state cache no longer used
  - Special facilities to deal with packet drops, path changes, export, reinitialization



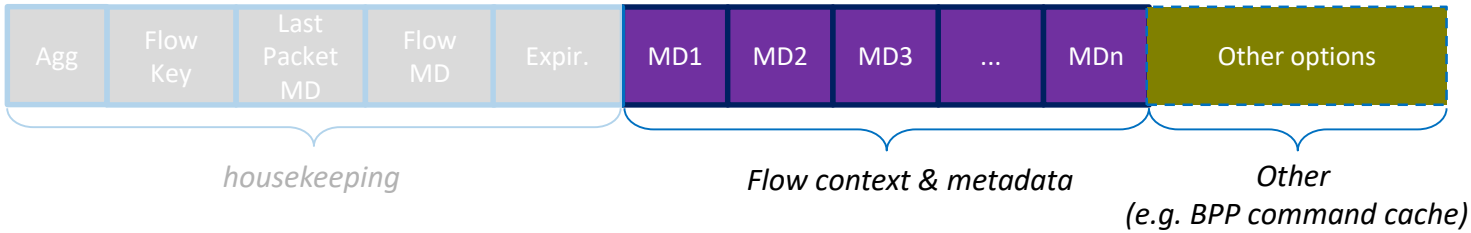
Statelet structure

- Housekeeping fields:
  - Flow key: associate packets with “their” flow and statelet (n-tuple, flow ID as option)
  - Aggregation of flows is possible (agg indicates key fields)
  - Last packet metadata
    - Time received (use for measurements e.g. inter-packet arrival delay)
    - Other ffs (e.g. hash, sequence # for flow operations – order & deduplication apps)
  - Flow metadata (e.g. BPP processing errors)
  - Expiration (inactivity timer can be tied to time of last packet)



Statelet structure

- Flow context and metadata
  - Freely programmable by BPP commands, referred in command parameters & packet metadata
  - Nonstructured (like metadata); structured option for further study
  - Metadata can further be structured into a portion that is protected vs one that is not (→ discussion further below)
- Other options
  - BPP Command cache
  - Accounting information (requires additional security infrastructure)



BPP Actions

- Initialize-statelet (size, expiration, expir-action, contents, reroute-protect)
  - size: amount of user data to allocate (excludes metadata)
  - expiration: inactivity time (in seconds)
  - expir-action: one of (do nothing, export)
  - contents: data to initialize statelet with (written from offset 0)
  - reroute-protect-range: monitor for path changes *(for future support)*
    - Indicates statelet data that will be replicated in case of path changes, indicating the range (length) until which to protect
    - 0: no reroute protection (0 octets to protect)
  - command-cache *(for future support)*
    - Number of octets reserved for command cache (0: no command cache)
    - Command cache will be automatically reroute protected
    - Any commands in the command cache are applied before BPP Block Commands

BPP Stateful Extensions (contd.)

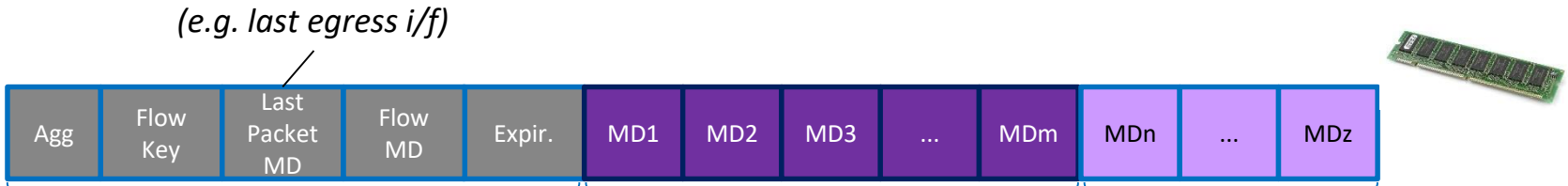
- More BPP Actions
  - Remove-statelet
  - Export-statelet-data *(format for further study)*
- Predefined data items:
  - Last Packet metadata items, e.g. last-packet-received
  - Statelet size
  - Statelet reroute-protected data
- Predefined condition:
  - statelet-exists (checks for presence of statelet)
- Support for parameter location/category “statelet” for action + condition parameters
  - location/category “statelet” (lp_x) 10: statelet offset

Selected issues with state

- State management
  - Reclaiming state that is no longer needed
    - Requires system infrastructure analogous to flow caches
    - “Glorified Netflow Cache”
  - Mitigation of resource scarcity
    - Failure to allocate statelet, failure to access statelet data result in BPP error condition
- Path changes
  - “Orphaned” state – collected eventually when state expires
  - Missing state (from non-initialized devices) → reroute protection

Note: Complementing BPP with a controller to manage deployment of statelets with cached data (commands, SLO) conceivable but not explored here

Example: Reroute Protection

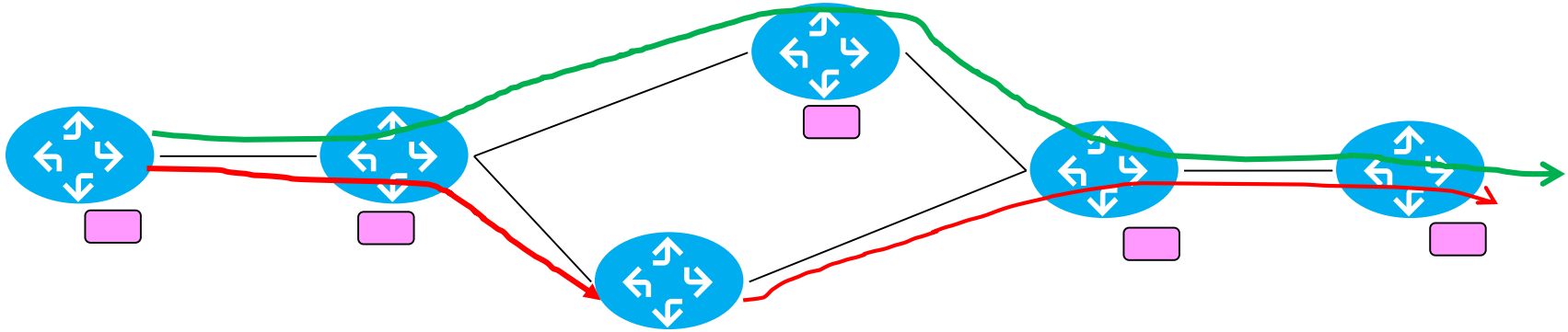


housekeeping

Reroute-protected state:
(reroute-protect range)
e.g. flow context, SLOs, command cache
indicate protected metadata portion from offset
(0: no protection)

Other state – node-dependent
(recoute protection not applicable):
e.g. statistics, ops telemetry

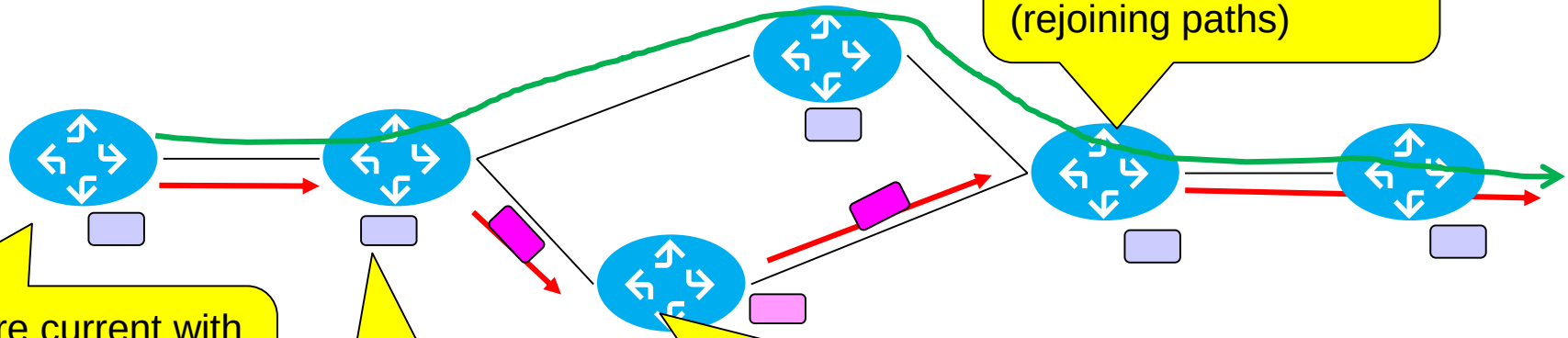
Reroute protection



no statelet!

Where is my command cache?
Where is my other cached metadata, e.g. SLO?

Reroute protection



Do not initialize when statelet already exists (rejoining paths)

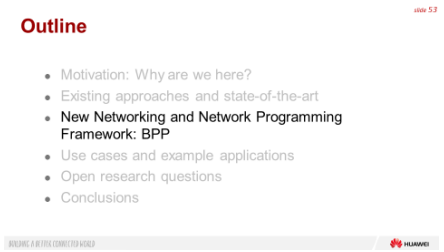
compare current with previous forwarding decision (egress i/f)

change detected (or first time): replicate protected statelet state to downstream node (piggyback; fragmentation considerations apply)

initialize statelet before applying other commands (since they may refer to statelet)

replication process repeats (this is a new forwarding decision)

BPP: A New Networking and Network Programming Framework



- BPP Framework overview
- BPP protocol: Blocks, Commands, Metadata
- Stateful extensions
- BPP programming model
- BPP engine, node infrastructure, deployment model

BPP Programming

- A Command Block contains a sequence of commands
- A BPP “Algorithm” thus involves writing a sequence of commands
  - Specify the order of commands, and where serialization must occur
  - For each command, specify:
    - The conditions under which to execute the command (incl Boolean operators)
    - The set of actions that should be performed
    - An indication of the order of actions, and where serialization must occur
    - The parameters that go with each
- Additional constraints (timing!) can be specified
- An SDK will involve support for a simple programming syntax and generating the BPP Block (including proper lengths)

BPP Programming Model – Design Decisions

- Path programs: commands performed at every node along path
  - BPP is intended to program a packet's and flow's behavior across the network.
    It is not intended to program a device how to process a particular packet.
  - That said, conditions allow to address individual nodes, roles
    (apply only if absolutely required – introduces multiple dependencies, assumptions, need for discovery)
- Command based
- No general-purpose programming language
  - No command hierarchies, e.g. declaration of functions
  - No loops, no recursion in order to facilitate pipelining

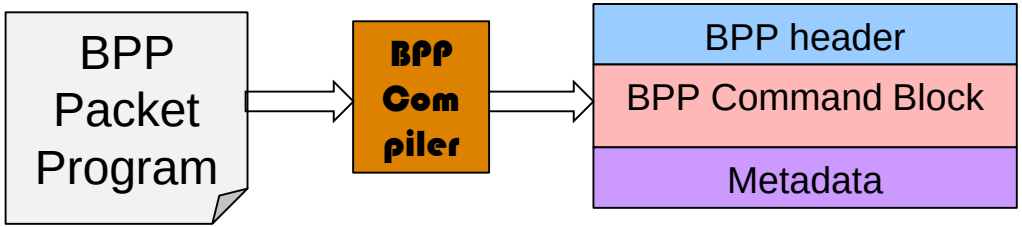
BPP Programming Model – Design Decisions (2)

- Concurrency model
  - Explicit definition of interdependencies facilitates optimization, avoids serialization
- No typing of parameters, type-safety of commands
  - Well-forming of BPP Blocks and integrity of commands and parameters are the responsibility of the BPP ingress node
  - Low overhead, more compact encoding
  - Ill-formed BPP Blocks are to be discarded (e.g. exceeding length boundaries, etc)
  - Any errors affect only the same packet and flow
- No scratch-pads or declaration of variables for intermediate results
  - No need for additional memory allocation
  - Arithmetic, functions directly performed on parameters per commands
- No backchannel/signaling
  - Results/outcomes of commands not visible to “source”

BPP Programming Model – Design Decisions (3)

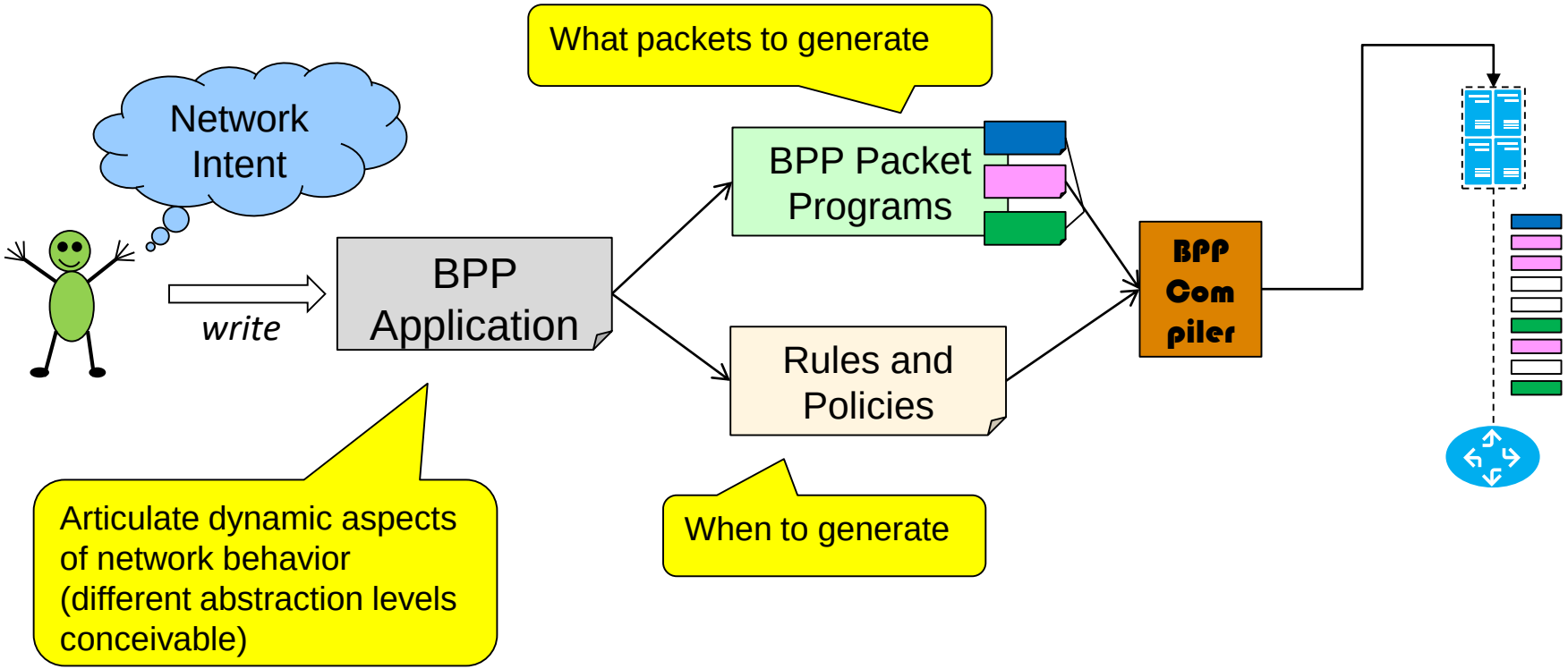
- Support for timing constraints
  - Support for timing constraints can be added
- No actions to allow generation of additional packets via commands *
  - Avoid attack vector
  - Caveats (*):
    - State replication (e.g. in event of path changes) for stateful extensions under consideration
    - Flow operations for further study (e.g. to deduplicate)
    - Fragmentation and reassembly for further study (e.g. for elastic BPP Blocks)

BPP SDK considerations

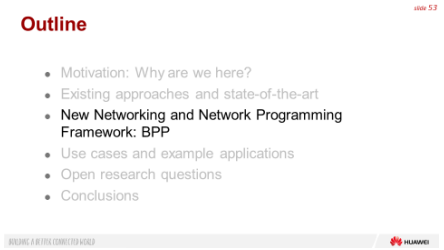


- Converting a “BPP Packet Program” to a BPP Block is necessary but not sufficient...
  - Network services and streams have dynamic aspects
  - → need different BPP commands / packet programs at different times
  - → need rules / FSMs / policies when to generate what

BPP SDK considerations (contd.)



BPP: A New Networking and Network Programming Framework



- BPP Framework overview
- BPP protocol: Blocks, Commands, Metadata
- Stateful extensions
- BPP programming model
- BPP engine, node infrastructure, deployment model

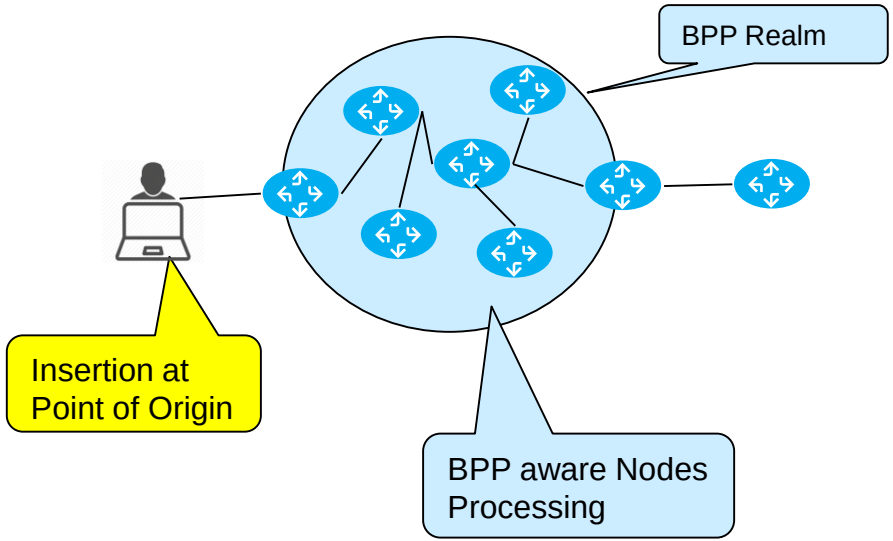
BPP Packets Headers Creator

Requisites:

- BPP engine exists and it has been provisioned
- BPP User/App “profile”:
  - Part of the BPP engine config
  - Used to apply the appropriate BPP header.
  - Examples:
    - It may contain information on what instructions to apply to particular packets
    - Some application have strict requirements
    - User access characteristics
  - Packets are constructed with BPP headers according to known “rules”

BPP Packets Origination Point

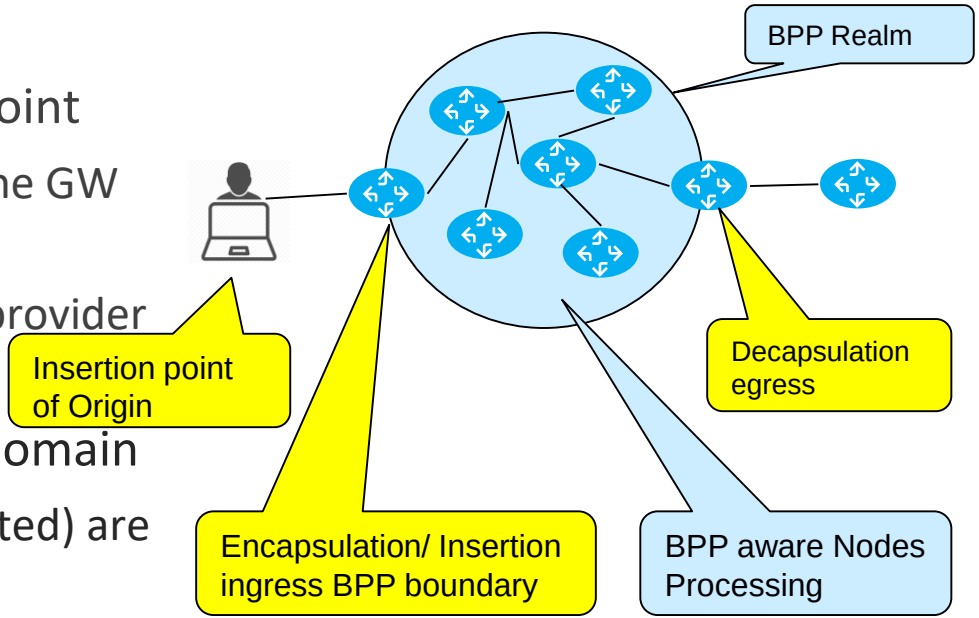
- BPP Headers may be added at
  - Insertion at Origin
  - Encapsulation on a BPP boundary
  - Insertion in a trusted domain
  - Decapsulation at egress



- Insertion at point of Origin
  - User has a BPP aware device
    - App directy builds BPP Hdrs
    - BPP Engine determines the appropriate BPP header and creates the BPP hdr for packets
  - User sends normal packets, BPP aware agent process the packet and add the header

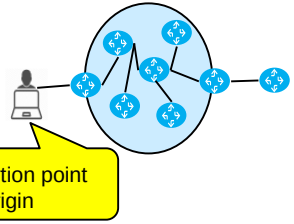
BPP Packets – Origination Point

- Encapsulation at a BPP boundary point
  - User packets become payload at the GW device of user edge BPP boundary
  - User packets become payload on provider edge BPP boundary
- Header Insertion within a trusted domain
  - Packets within the BPP realm (trusted) are have insertion of the BPP header
- Processing within the BPP domain



BPP Packets – Insertion at origin

- User may be input simplified
  - may be based on Profile/Context
  - Service Access characteristics – high speed for some services



BPP SDK/FW + Intent Edge

Policies
Modelized Reference

App/User Rules input:
Through Context/Profile/
Graphical interface
Eg: SLO for this flow

Payload

App

BPP Engine - Apply BPP rules/Policies

Modelized Reference
Context based on user
Type of services

SDK

Generate BPP Commands

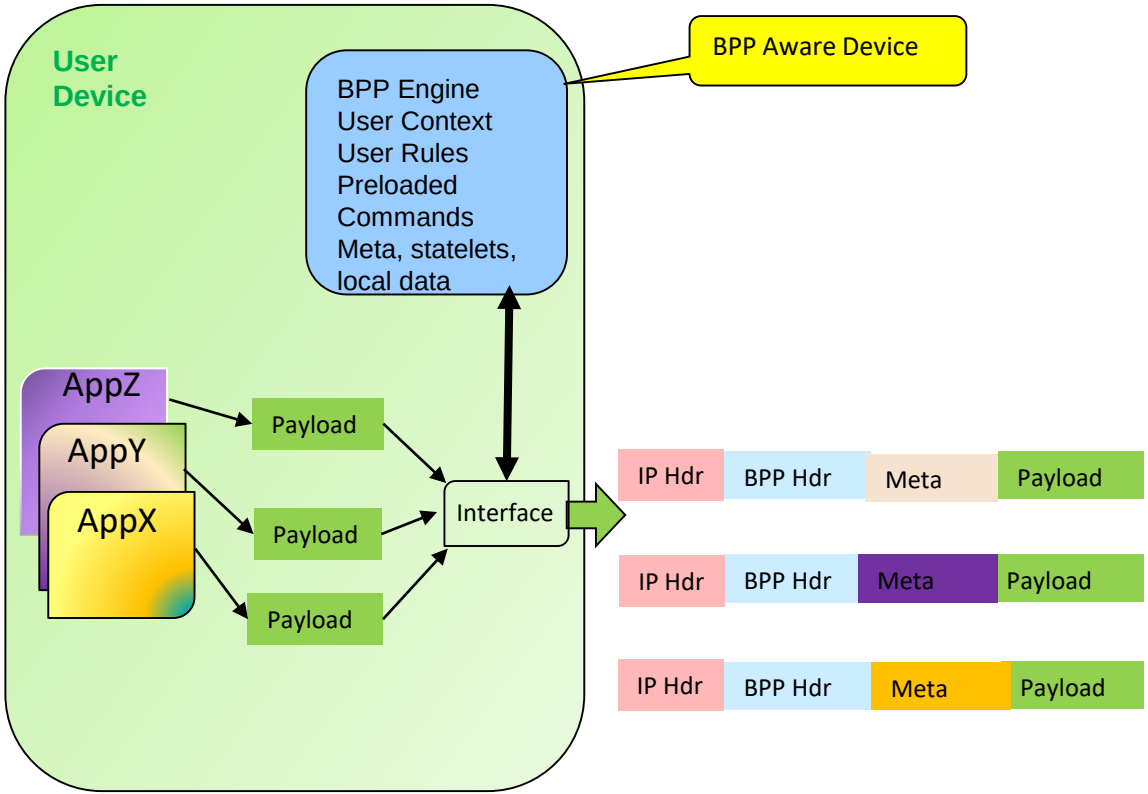
Status/Monitor

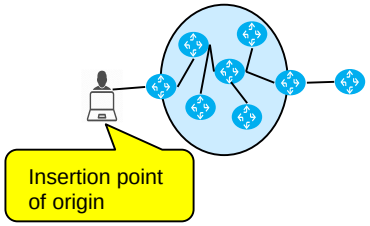
User Device

L3/L2 | BPP Header | Payload

Packet on wire

BPP Header Insertion at Point of Origin

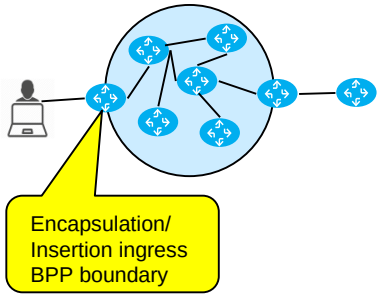




User device is BPP Aware
- BPP Engine has context
- Knowledge of the profile
- Assess what BPP Hdr to apply
- Create the BPP Hdr
- Builds the final packet

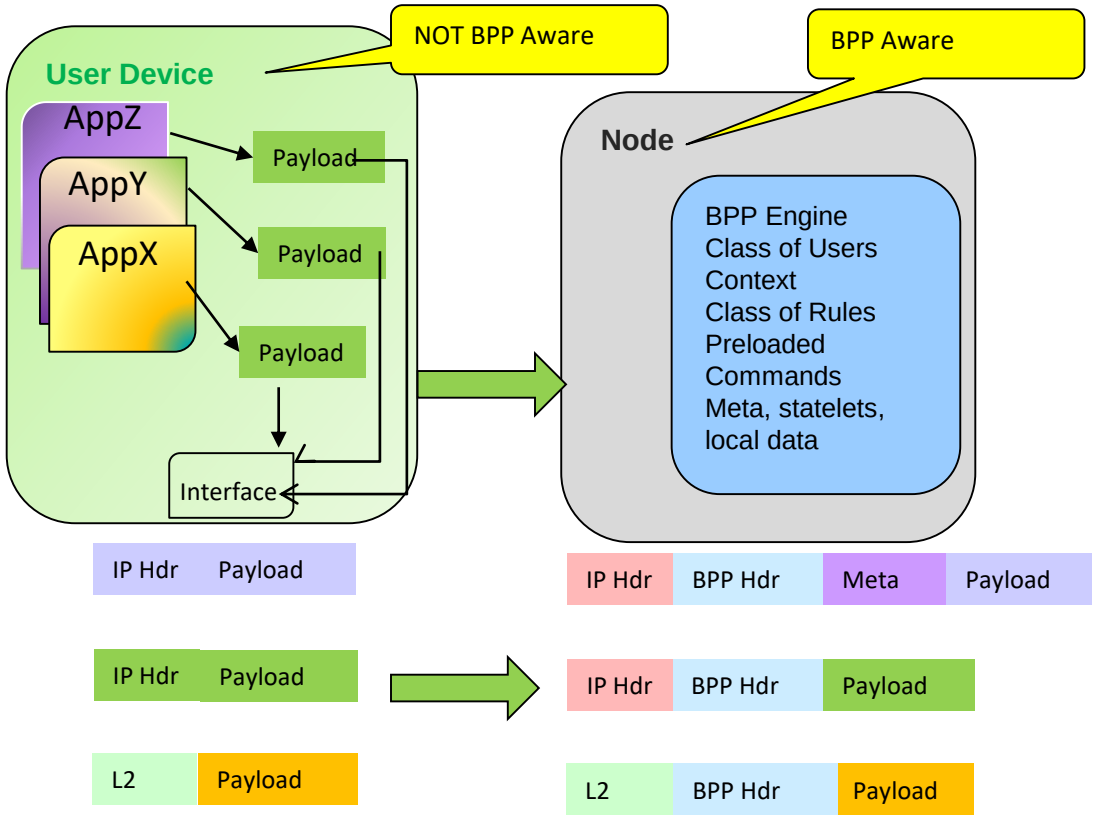
- Allows User/Apps to be unaware of commands

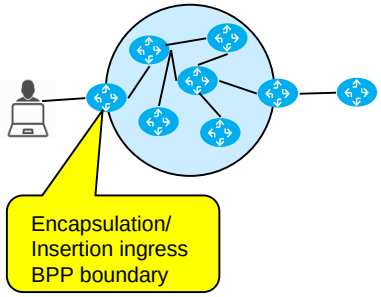
BPP Insertion/Encap Provider Edge



- BPP Engine
- Provisioned with intent of User/Admin
- Look up rules(user) or policies(network) to apply BPP Hdr

BPP Header Encap/Insertion PE





BPP Engine may have user context

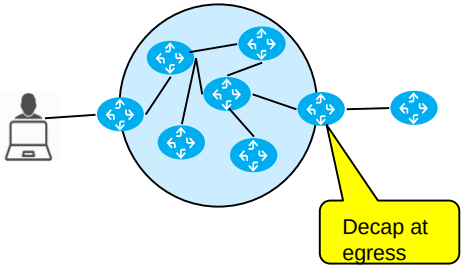
- Assess what BPP Hdr to apply
- Create the BPP Hdr
- Encap the packet

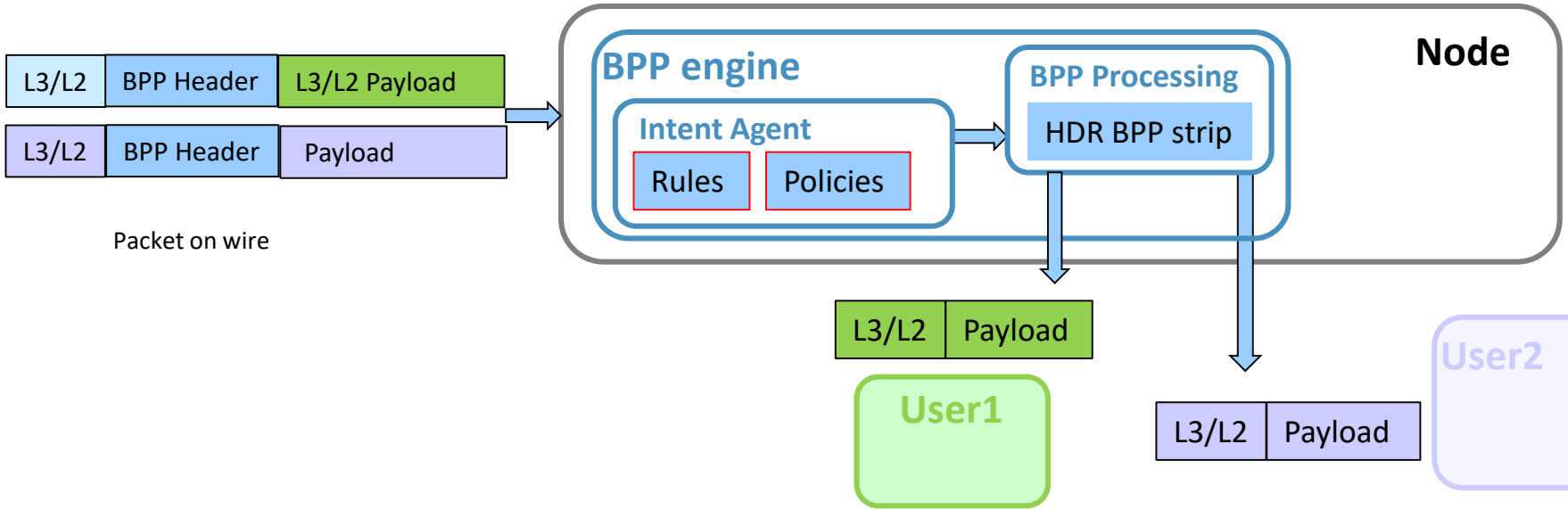
- Allows User/Apps to be unaware of commands

BPP Packets – Decapsulation

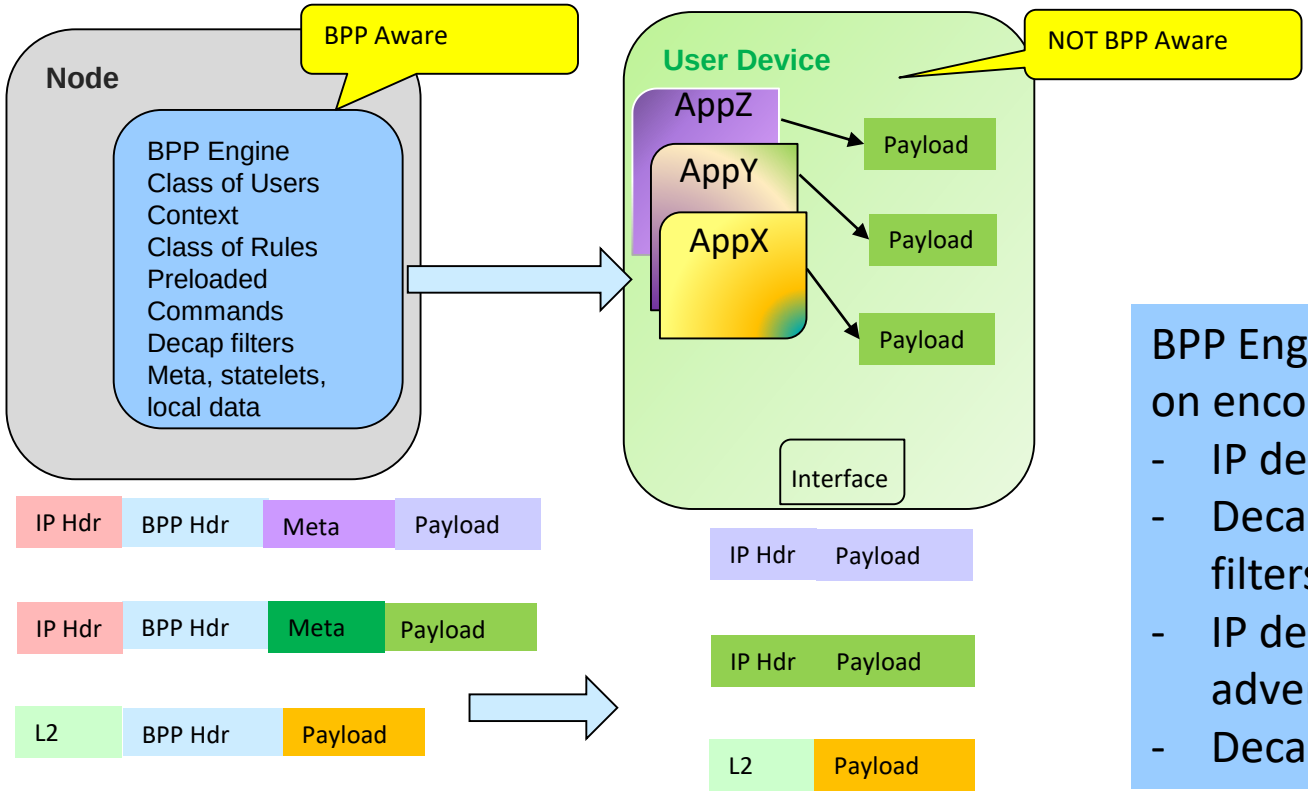
Decapsulation can be ”programmed” at the BPP egress router

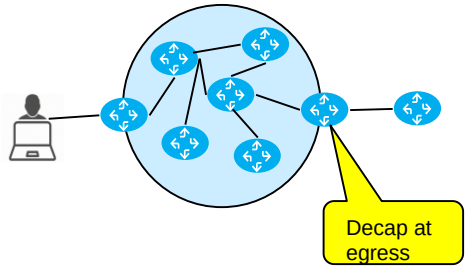
- By decapsulating on arrival at the egress destination (encap dest)
- Configuration of decapsulation filters on the egress BPP
- By command in the BPP header





BPP Header Decapsulation/Stripping

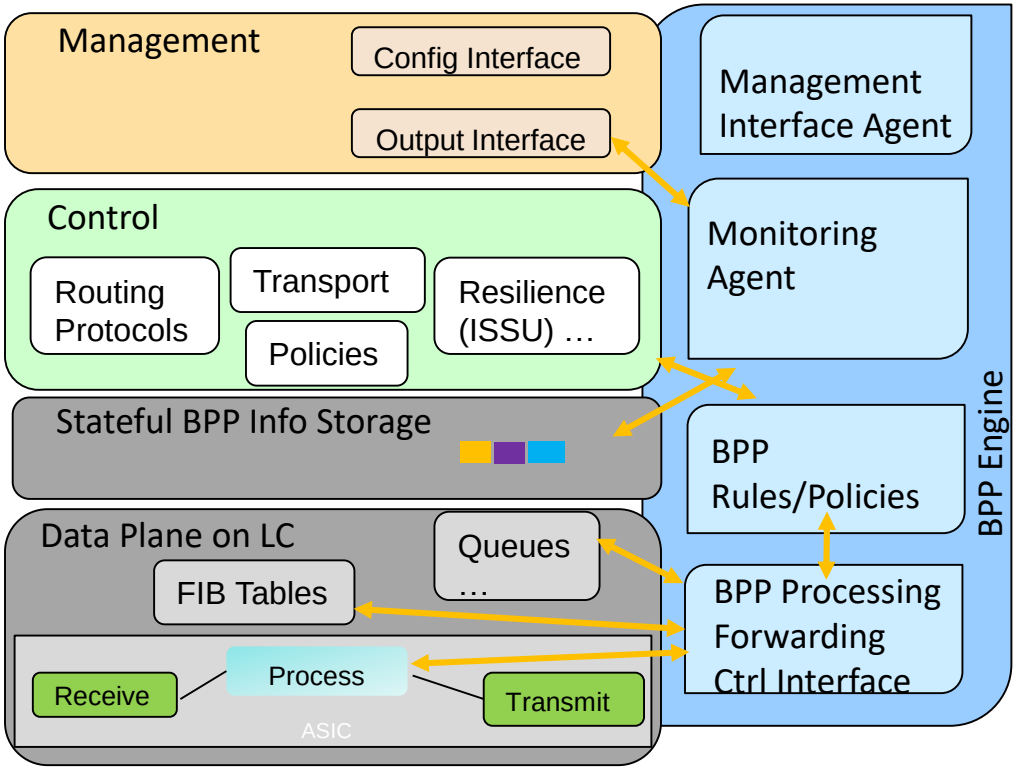




BPP Engine may strip depending on encoding

- IP destination is egress node
- Decapsulation policy and filters programmed
- IP destination is part of its advertisement
- Decap the packet

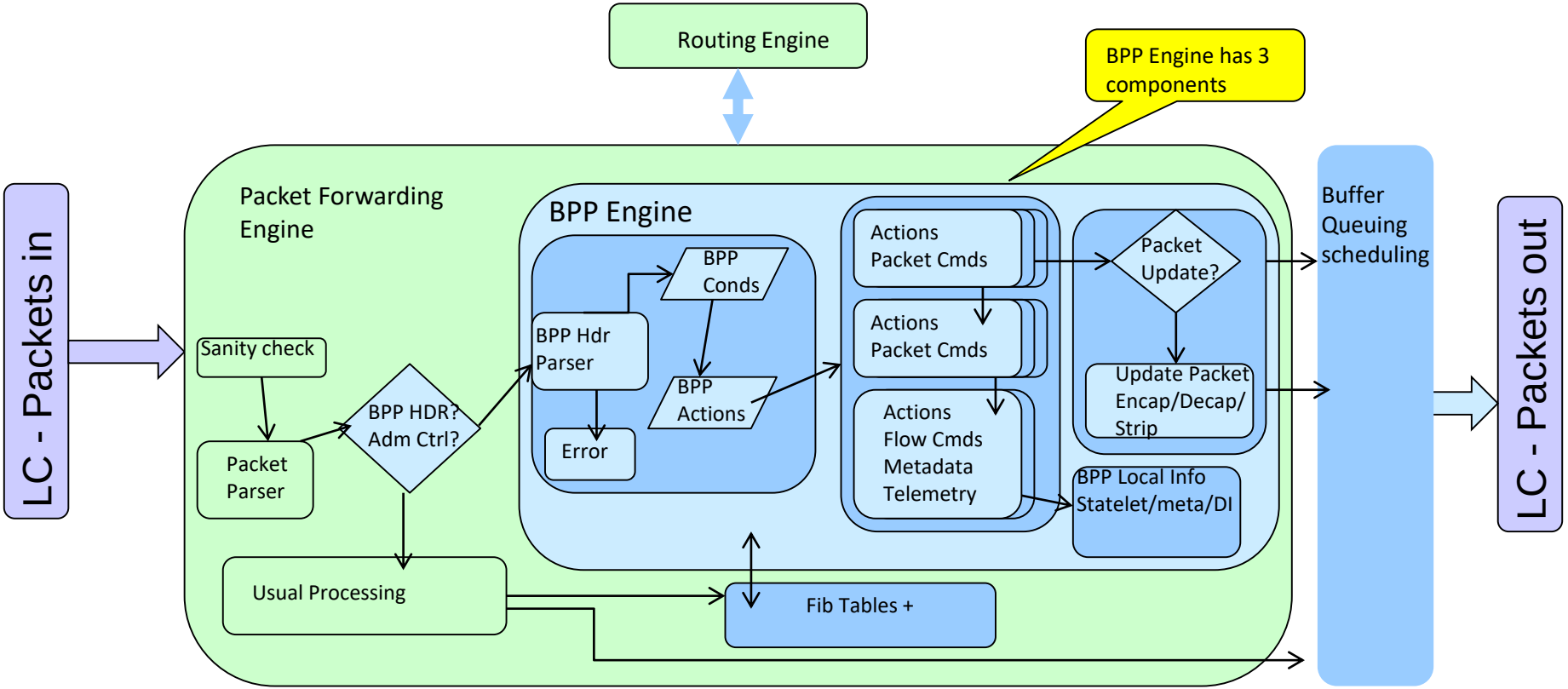
BPP Engine - Interactions on Node



BPP Engine Interactions and interfaces on the router ecosystem:

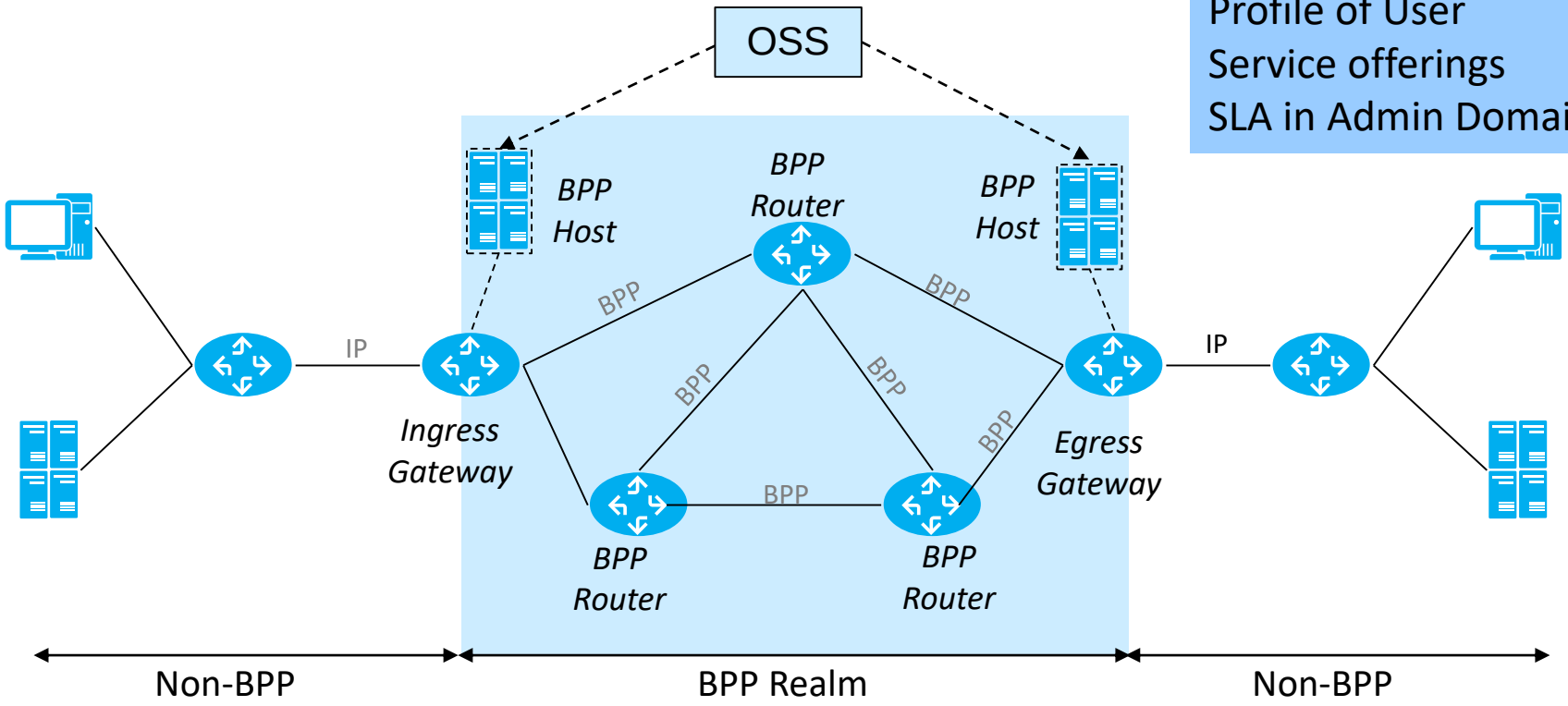
- BPP Rules/policy agent (Provisioned by the SDK)
- BPP Command processing in the forwarding engine)
- Monitoring Agent (for telemetry and flow management)
- Management Interface

BPP Engine - Packet Processing



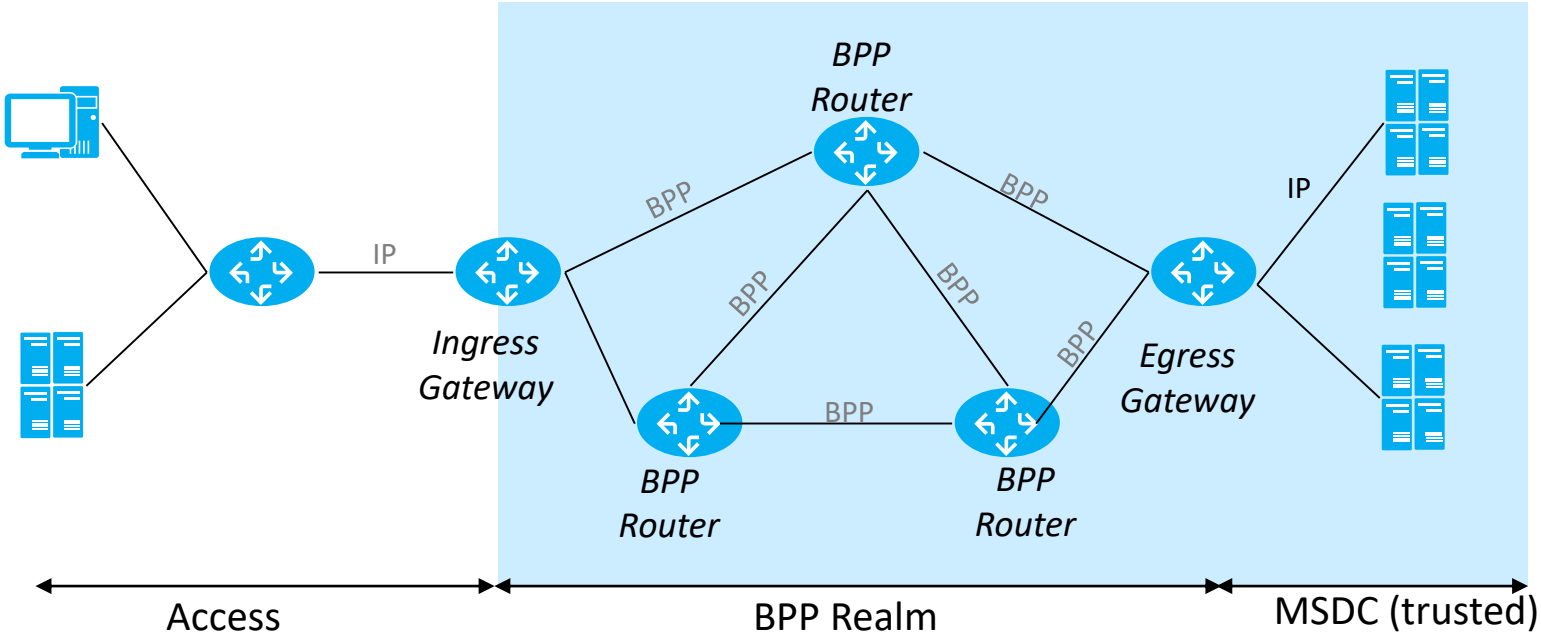
BPP Deployment Scenarios

Typical ISP scenario
Profile of User
Service offerings
SLA in Admin Domain

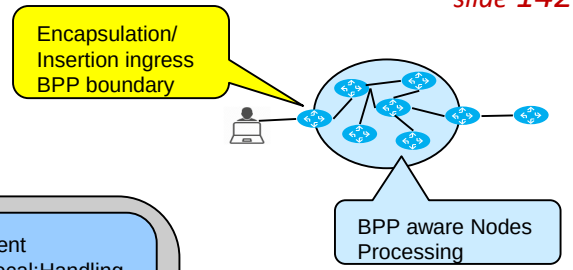


BPP Deployment Scenarios

Campus
SLA in Admin Domain

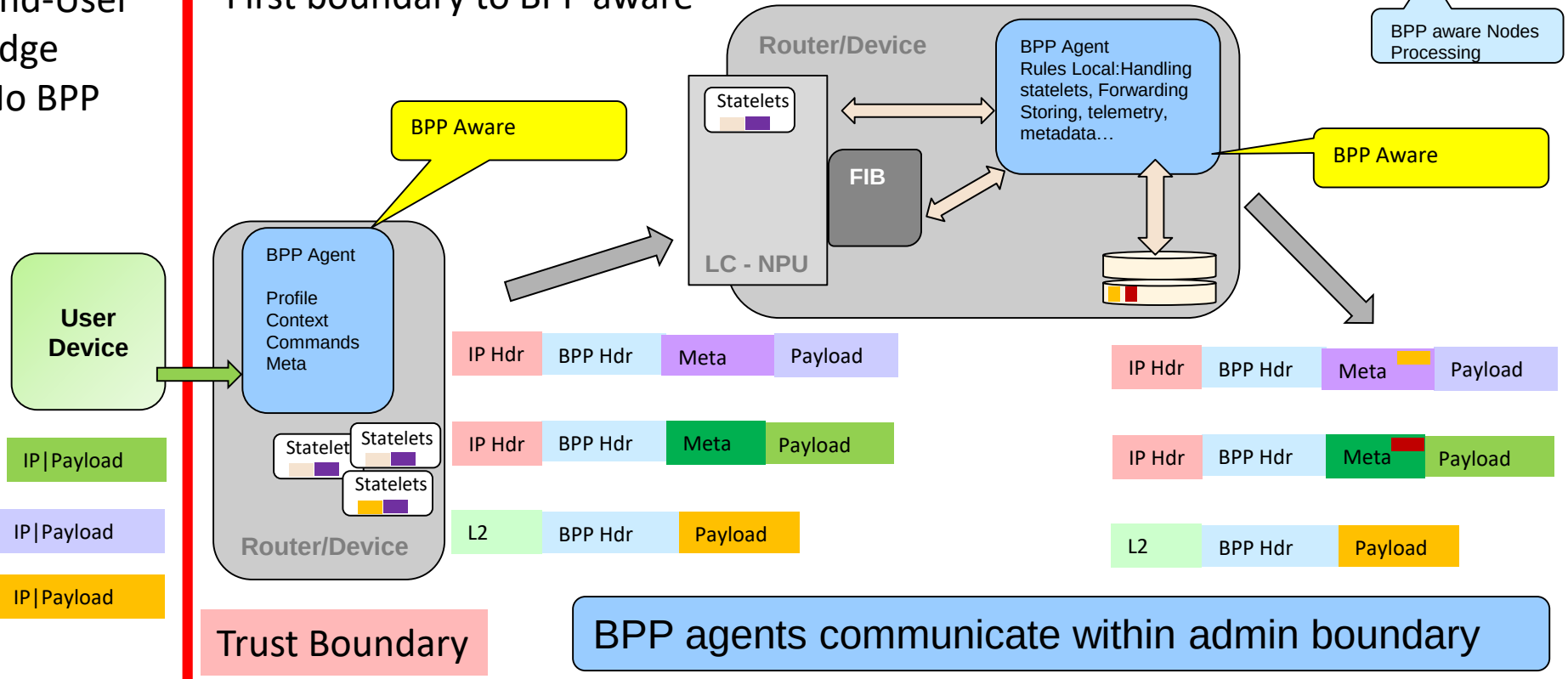


BPP Packet Flow handling



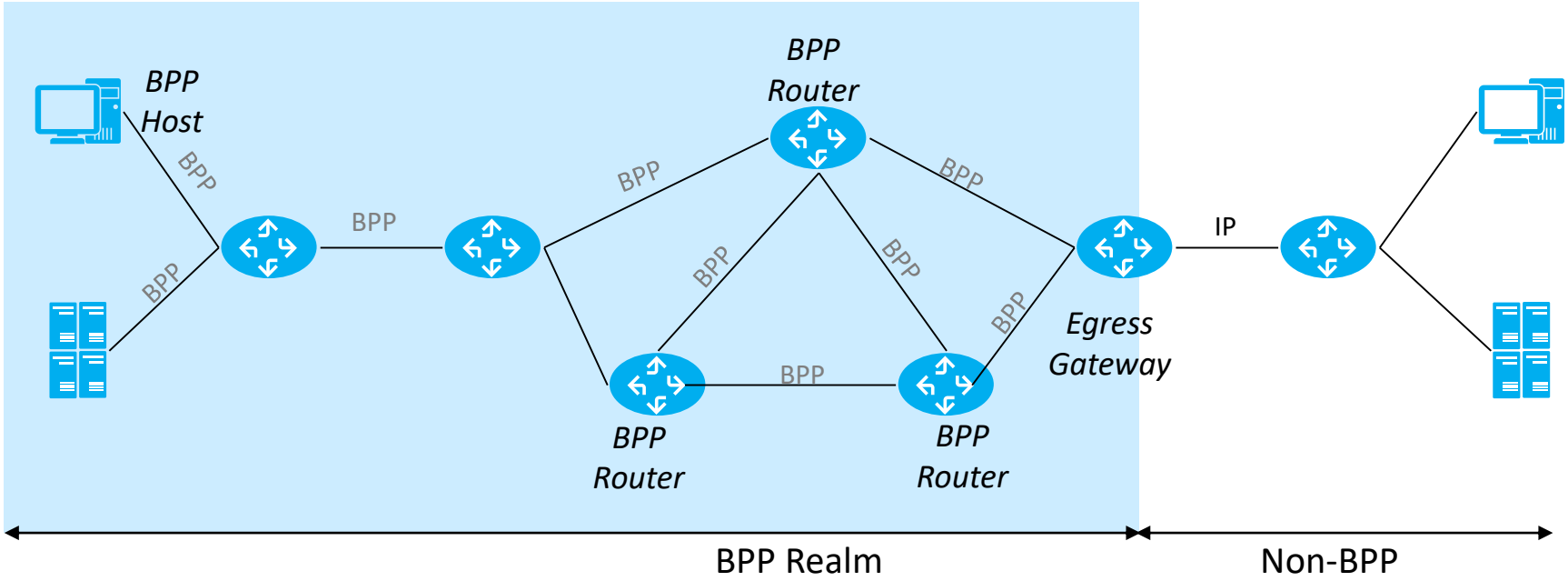
End-User
Edge
No BPP

First boundary to BPP aware

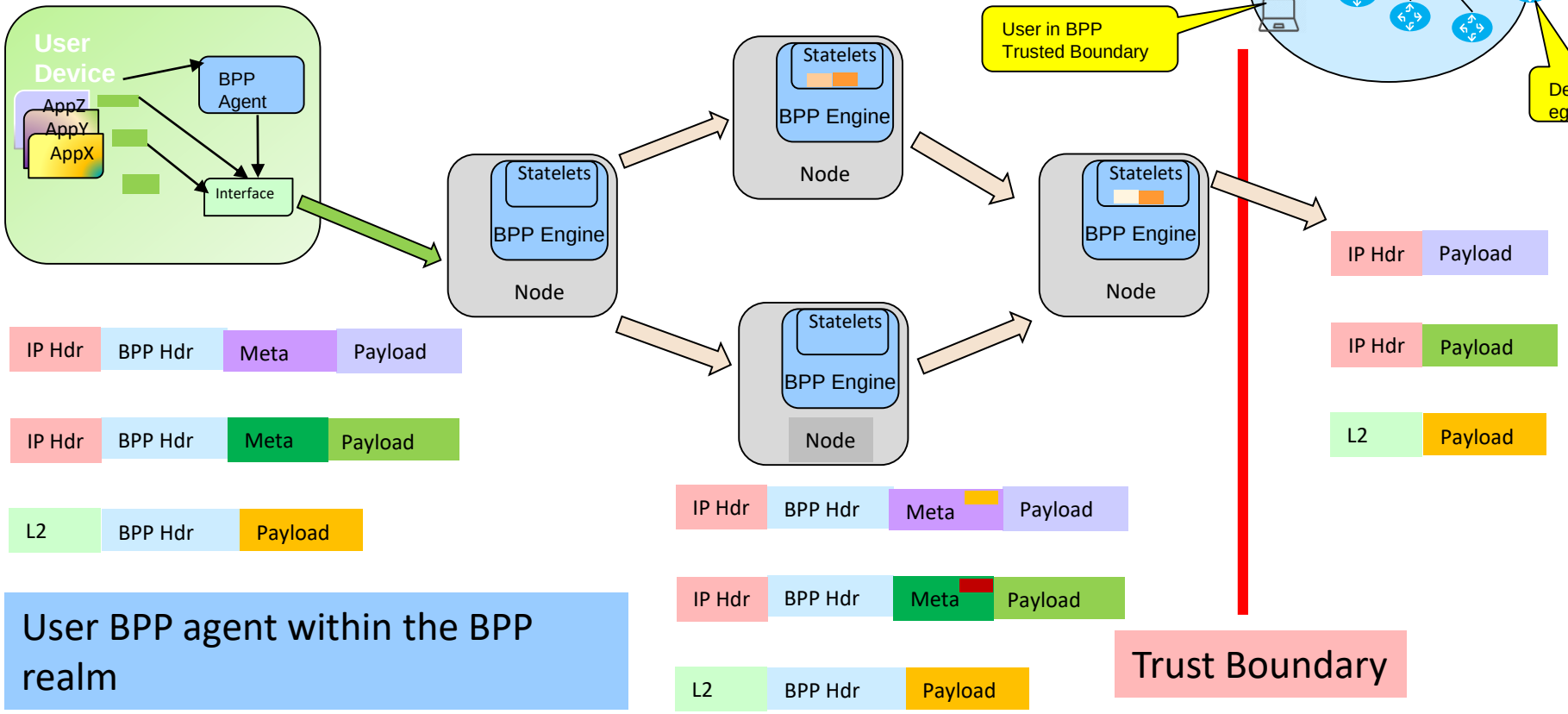


BPP Deployment Scenarios

Desirable; however, typically different trust boundaries apply



BPP Packet Flow handling



Outline

- Motivation: Why are we here?
- Existing approaches and state-of-the-art
- New Networking and Network Programming Framework: BPP
- Use cases and example applications
- Open research questions
- Conclusions

Things you can do with BPP (examples)

- Make context-aware forwarding decisions based on observed service levels and extrinsic factors (e.g. current i/f utilization, queue occupancy)
- Carry SLO in metadata
- Measure packet delay, inter-packet delay variation & compare against SLO
- Maintain custom statistics about a flow
- Drop a packet if it is late beyond repair (e.g. for Industrial Internet)
- Manage reservation of resources along a path;
  maintain, record reservations as part of statelet
- Service-level dependent accounting
- Aggregate telemetry data for a flow

Use Cases and Example Applications

- Time-Centric Forwarding for High-Precision SLOs
- A Greenfield Service for Data Mobility
- Operational Flow Profiling

Use Cases and Example Applications

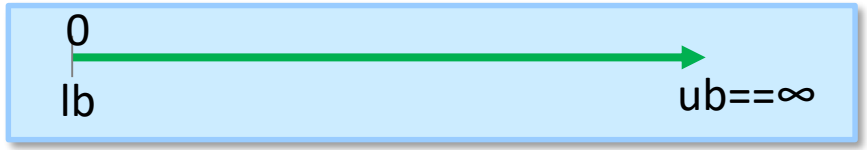
- Time-Centric Forwarding for High-Precision SLOs
- A Greenfield Service for Data Mobility
- Operational Flow Profiling

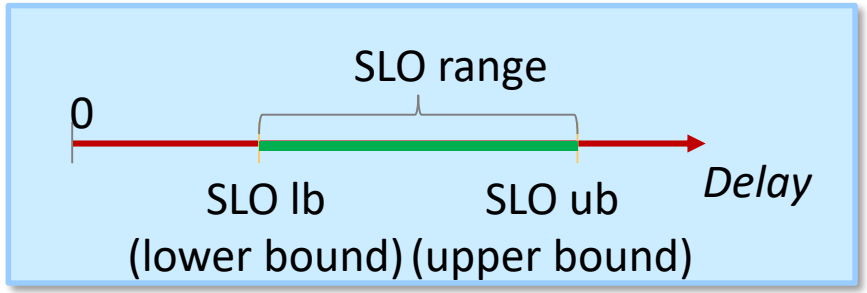
Delay SLO

- A Service Level Objective is a performance goal for a service under certain well defined constraints
- Service levels are observed end-to-end (meaningful for a service)
  - Service level is not to be confused with performance of an individual node
  - Of course, each node “contributes” to the service level
- Note: A Service Level Guarantee involves more than an SLO:
  - Focus here: Delivering according to SLO
  - Assessing whether a guarantee can be given
  - Assessing whether an SLO was met
  - Remediation actions when the guarantee is violated

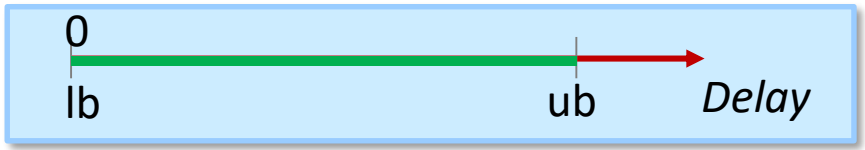
Delay SLO

Best effort (no guarantee)

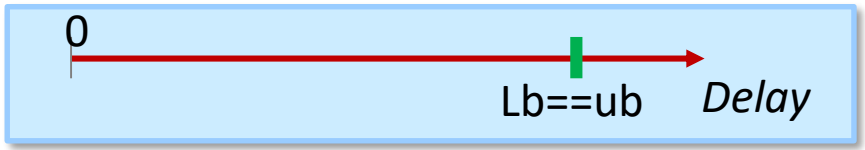




In-time guarantee: lb==0



On-time guarantee: lb==ub



Motivation for network in-time SLOs

- VR/AR: stringent limits to max. motion-to-photon time
  - Dizziness, reduced QoE at longer delays severely reduces user acceptance
- Tactile Internet: stringent limits to delay for haptic feedback
  - Lack of sensation of being in control, sluggish control would make many applications infeasible
- Industrial controllers: stringent limits to feedback control loops
- V2X
- Remote-controlled robots and drones
- And many more

Motivation for network on-time SLOs

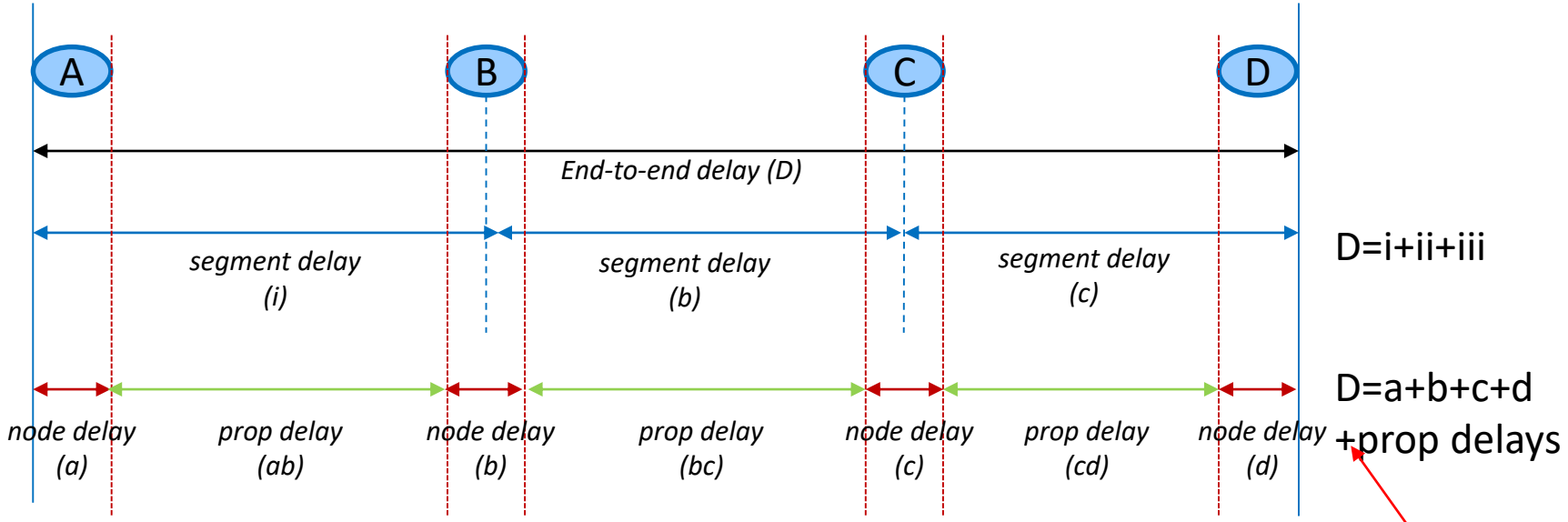
- On-time is a lot “stronger” than “in-time”
- On-time can be emulated at the receiver if an in-time guarantee is provided and the receiver/ application has buffer space
  - Network on-time guarantee only needed when app buffer cannot be assumed
- Why bother in the first place? Possible reasons:
  - Fairness: Do not give anyone an unfair advantage, examples
    - multiparty applications and marketplaces
    - Examples: trading, gaming (incl involving tactile Internet)
  - Synchronization, examples:
    - Robot collaboration (lift packet by two remotely controlled robots)
    - High-precision measurements (e.g. remote polling at exact intervals)

Component SLOs

- Decomposable Service Levels yield decomposable SLOs
  - Each component (segment, hop) can get a “component SLO”
  - Sum of component SLOs yields end-to-end SLO
  - Component SLOs determined through SLO budgeting
  - Observation:
    - Component SLOs are fairly “rigid” and yield unnecessary violations of end-to-end SLOs
    - Precise budgeting / allocation may be difficult
    - High-precision “nailed up” paths & resources are brittle
      No oversubscription can mean low utilization and high expense
      Little capability to react to unforeseen events in timely manner
    - One component’s SLO violation could be offset by another component exceeding its SLO

→ A more flexible, elastic approach is needed

Decomposition of Service Levels



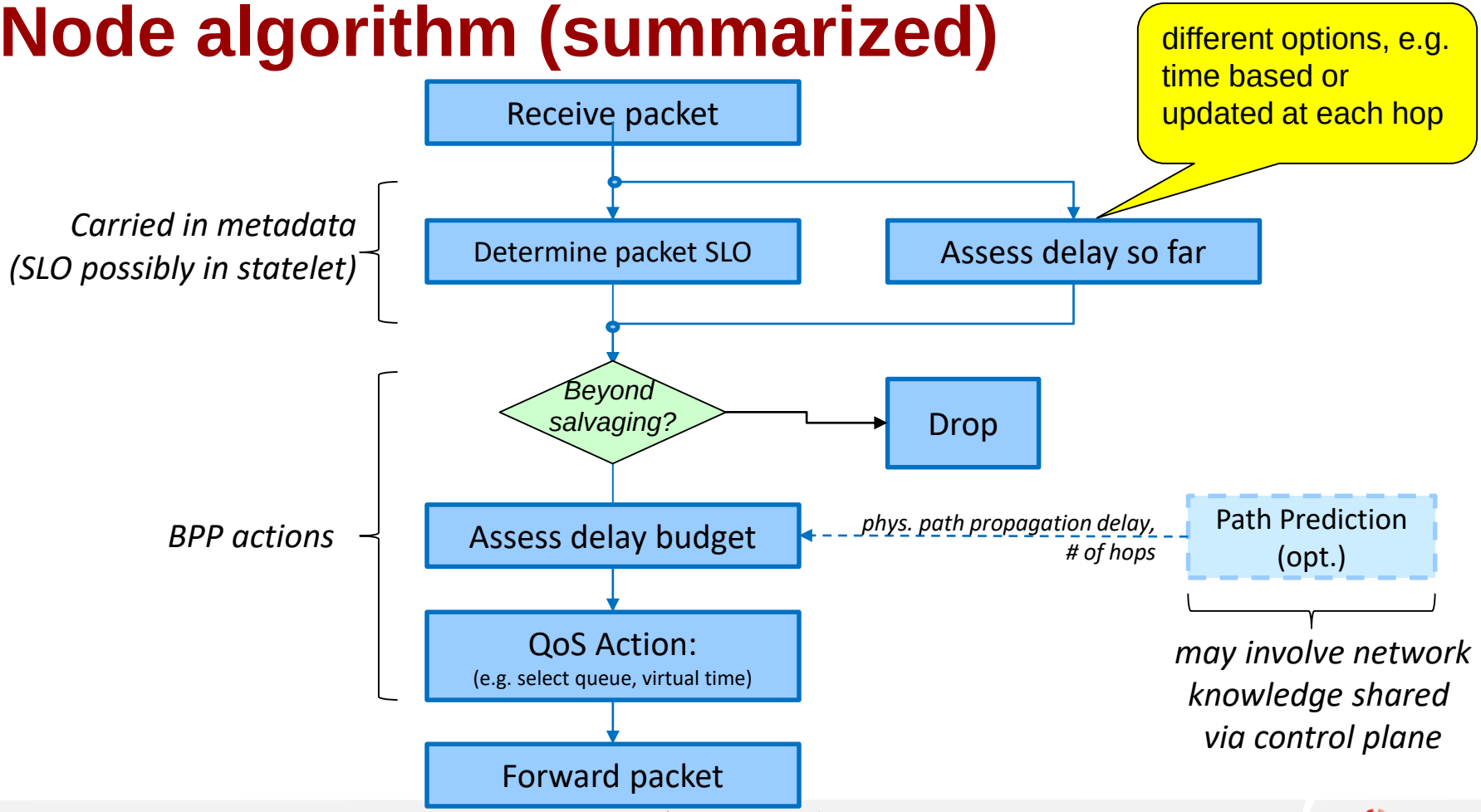
Some Service Levels can be easily decomposed: e.g. delay

- End-to-end delay is the sum of all component delays
- Note, not true for all service levels:
  - E.g. throughput: end-to-end throughput is minimum of component throughputs
  - E.g. jitter: No easy arithmetic

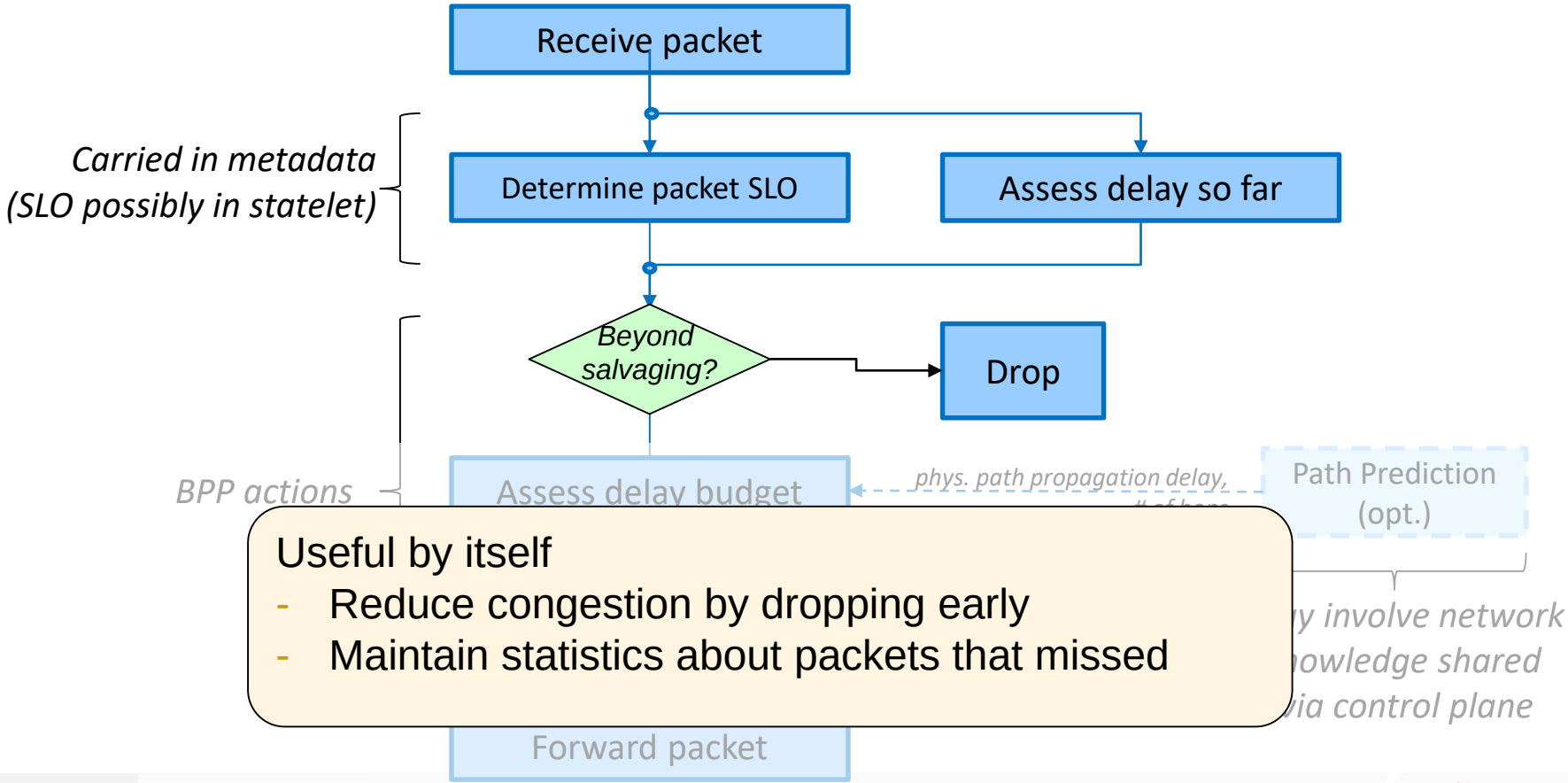
Time-Centric Forwarding

- Basic idea: make forwarding decision based on delay expectation
  - Packet is early: make it slow
    - Slow path, slow queue, buffer when feasible, etc
  - Packet is late, with a chance to still meet SLO: speed it up
    - Fast path, fast queue, etc
  - Packet is late beyond recovery: drop it
    - Reduce congestion, improve service levels for other packets
- Distributed algorithm across nodes (horizontal)
  complements vertical scheduling/ queueing
  - Elasticity: delays in one segment can be recovered by speedup in another
  - Meet precision SLO targets even in presence of non-determinism
  - No need for pre-allocated component SLOs, ability to recover from slight jitters

Node algorithm (summarized)



Validation of SLO compliance



BPP Mapping

```
Cmd1: serialize-next;
  cond: -;
  action: incr-by
    (par.meta ("packet-delay" offset 8 len 4),
     par.data ("egress-link-delay"));
Cmd2: serialize-next;
  cond: lt (par.meta ("slo-ub" offset 0 len 4),
            par.meta ("packet-delay" offset 8 len 4);
  action: serialize-next drop;
  action: break;
Cmd3:
  cond: -;
  action: serialize-next match-queue
    (par.meta ("slo-lb" offset 0 len 4),
     par.meta ("slo-ub" offset 4 len 4),
     par.meta ("packet-delay" offset 8 len 4));
  action: incr-by
    (par.meta ("packet-delay" offset 8 len 4),
     par.data ("local-packet-queue-time"));
```

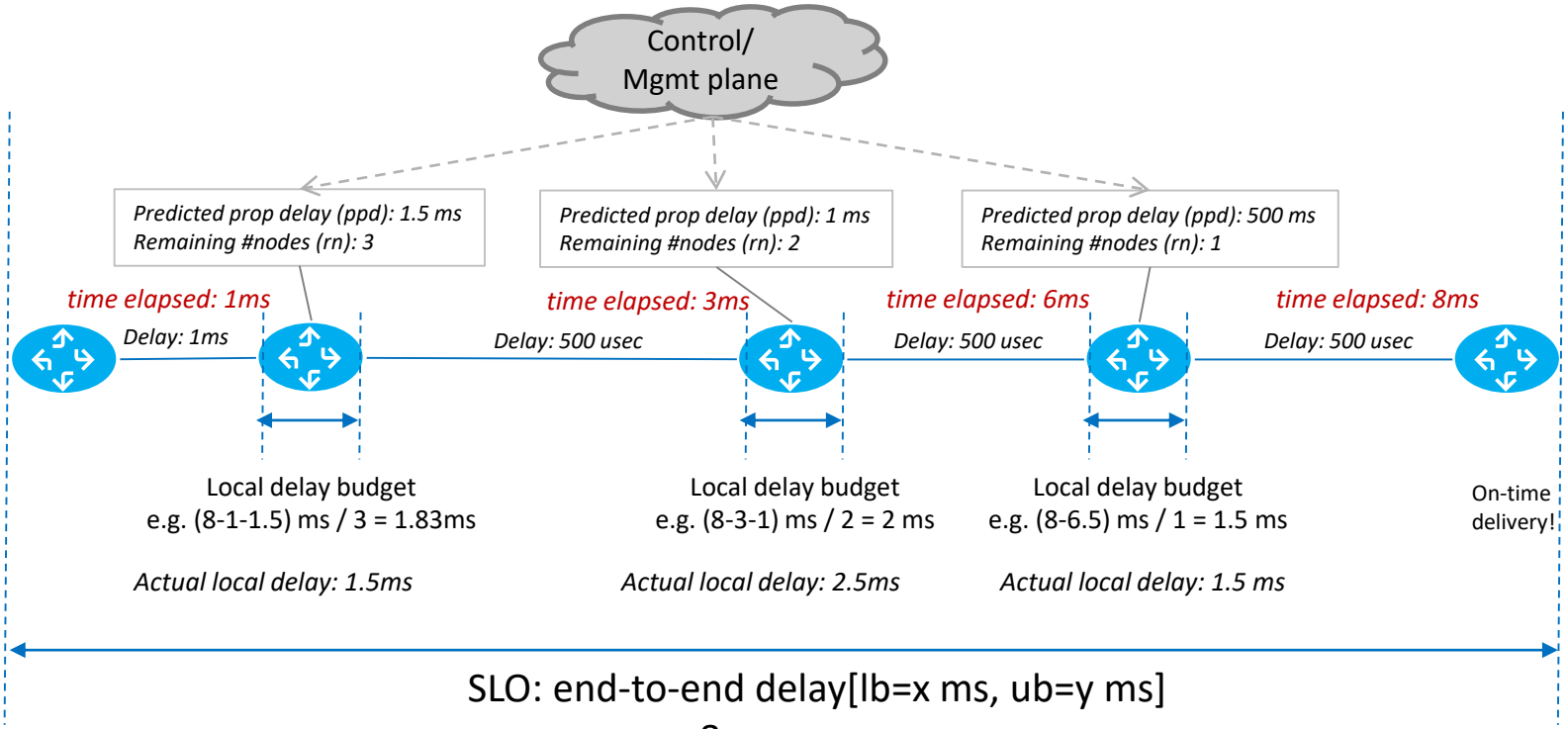
Packet delay experienced so far carried as metadata updated at each hop

known and provisioned via control plane

special-purpose action

known data item computed as part of match-queue operation

Example


Control/
Mgmt plane
Predicted prop delay (ppd): 1.5 ms
Remaining #nodes (rn): 3
Predicted prop delay (ppd): 1 ms
Remaining #nodes (rn): 2
Predicted prop delay (ppd): 500 ms
Remaining #nodes (rn): 1
time elapsed: 1ms
time elapsed: 3ms
time elapsed: 6ms
time elapsed: 8ms
Delay: 1ms
Delay: 500 usec
Delay: 500 usec
Delay: 500 usec
Local delay budget
e.g. (8-1-1.5) ms / 3 = 1.83ms
Local delay budget
e.g. (8-3-1) ms / 2 = 2 ms
Local delay budget
e.g. (8-6.5) ms / 1 = 1.5 ms
On-time
delivery!
Actual local delay: 1.5ms
Actual local delay: 2.5ms
Actual local delay: 1.5 ms
SLO: end-to-end delay[lb=x ms, ub=y ms]
e.g. x == y == 8 ms

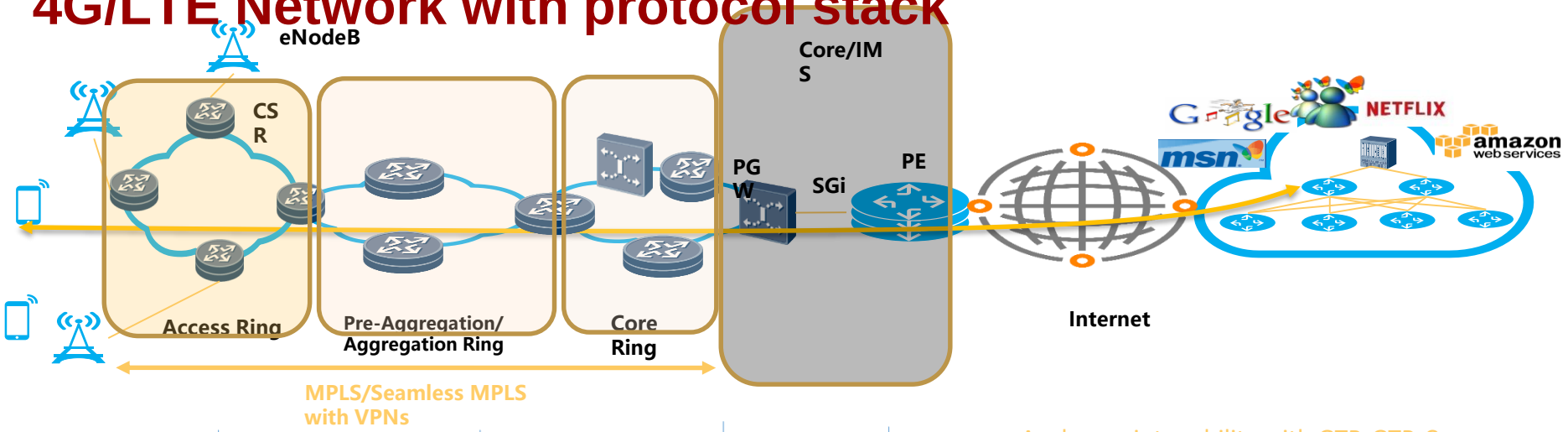
Discussion

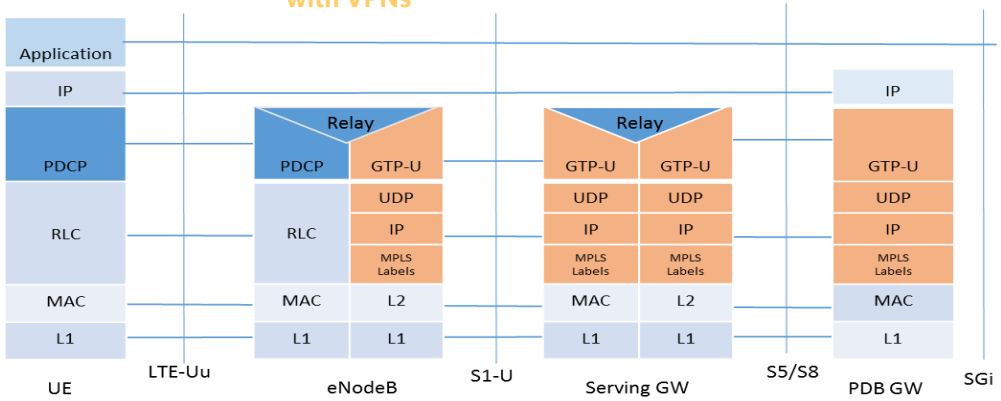
- Ensures delivery of packets according to SLO; benefits include:
  - Applicable to VBR, not just CBR
  - Works for “first packet” – no need for session setup / handshake
  - Early discard as an option
  - Drop packet early when SLO will be violated – reduce congestions, endpoint load
  - Inherent SLO validation – count packets that are late
- Full service level guarantees need also:
  - Assessing (in advance) whether a service level guarantee can be given
  - Admission control and assessing of service constraints (e.g., not-to-exceed rates)
- Local Delay Budget determination involves heuristics
  - More accurate predictions are useful but not necessary:
    Elastic property accommodates for “poor” predictions (to an extent)
  - Multiple tradeoffs and optimization possibilities, e.g. buffer space vs delay lower bound

Use Cases and Example Applications

- Time-Centric Forwarding for High-Precision SLOs
- A Greenfield Service for Data Mobility
- Operational Flow Profiling

4G/LTE Network with protocol stack

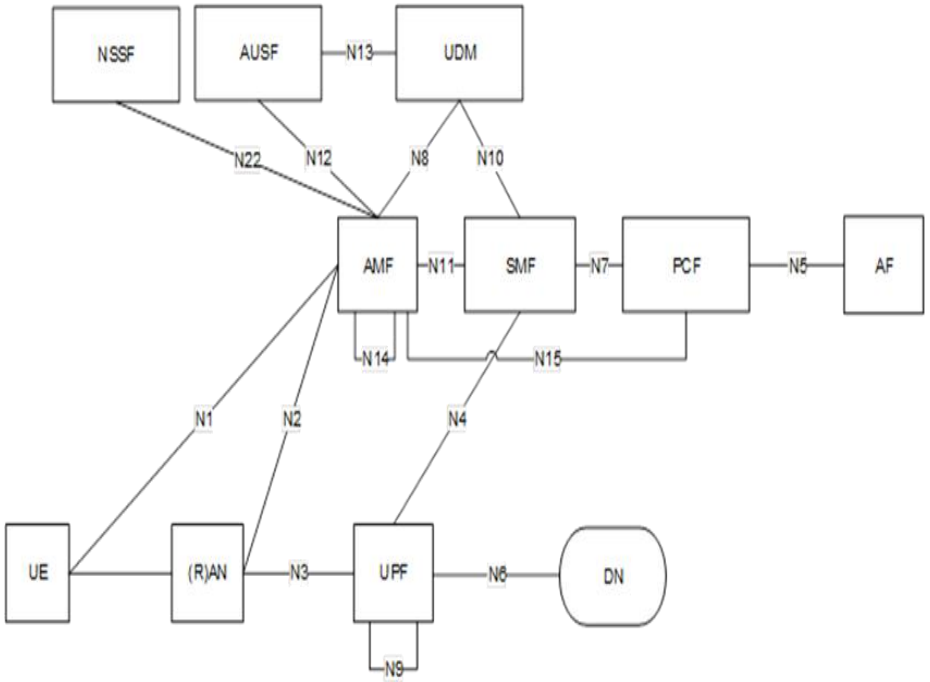




- Anchor point mobility with GTP, GTP-C overhead
- Not optimized for some traffic (video/multicast)
- Multiple areas (seamless MPLS complexity)
- Difficult TE/FRR solutions (RLFA etc..)

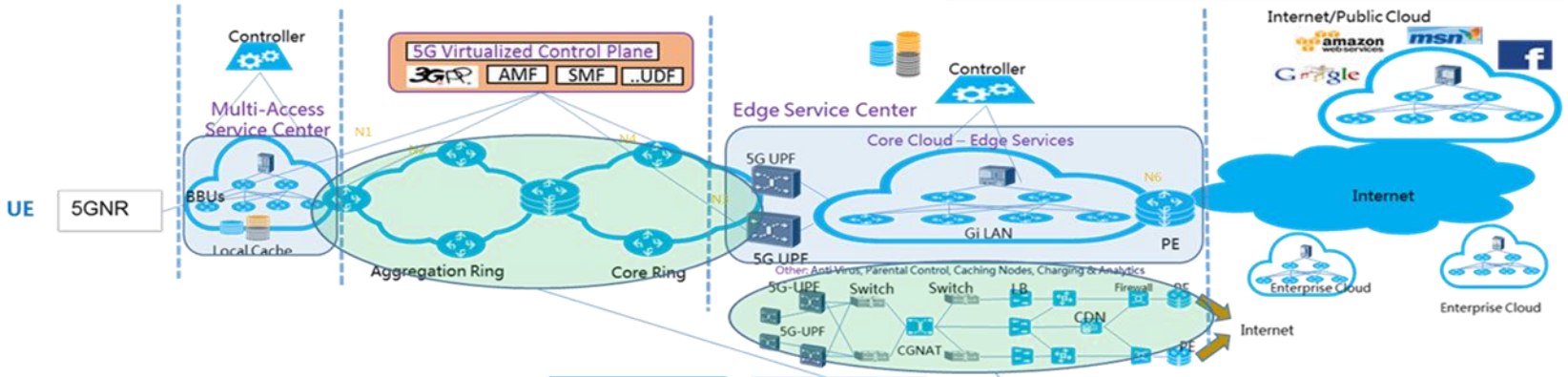
BPP with 5G and Beyond

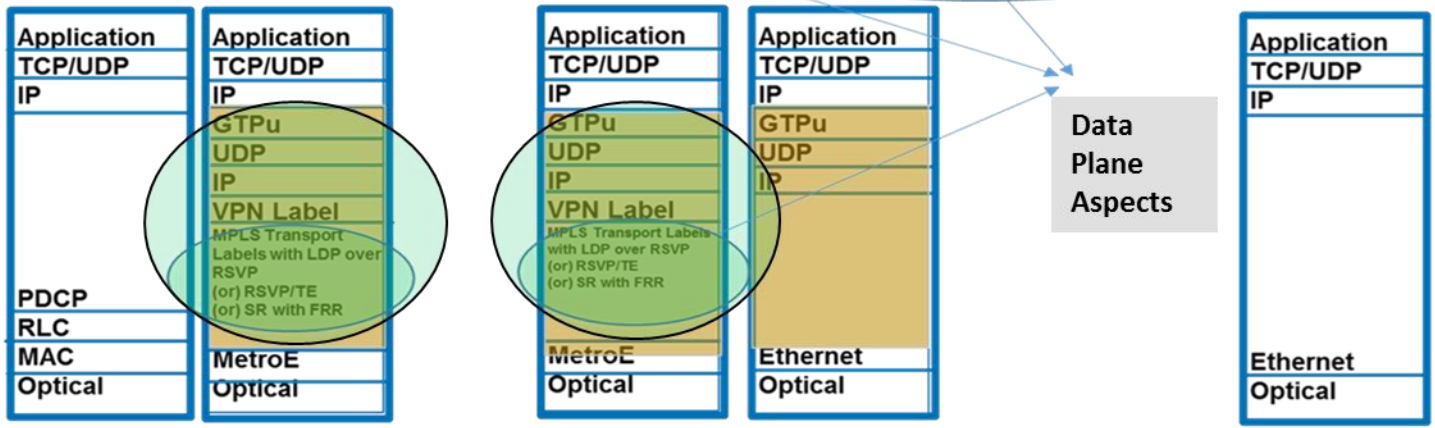
5G Network Architecture (Recap)



- CUPS architecture
- Key features eMBB, MIMO and URLLC
- For backward compatibility GTP is still defined for N3
- N9 interface is a significant change among others for transport network designs (tunneling/encapsulation yet to be defined)
- New Mobility Scenarios

REL15 – 5G NR and 4G Core network (Simplified)





Data Plane Aspects

**Note: SSC modes and new inter-UPF interface N9 interface is not shown here

What problem is being solved

- **GTP Overhead in tunnel creation in the control plane and data plane over head**
  - This part of the proposal only covers data plane aspects with BPP Protocols
  - An approach for Control plane changes needed to remove tunnel creation overhead is covered below

> **"Improving Performance and Scalability of Next Generation Cellular Networks"**
> Ali Mohammadkhan ; K. K. Ramakrishnan ; Uma Chunduri ; Kiran Makhijani
> **IEEE Internet Computing, Year: 2019 , Volume: 23 , Issue: 1**
> Pages: 54 - 63, IEEE Journals & Magazines

- **Lack of light weight path steering mechanism in the backhaul (for new 5G services)**
- **Lack of QoS awareness for low latency services**
- **Lack of high-precision service in the transport to meet various slices requirements w.r.t latency and jitter**

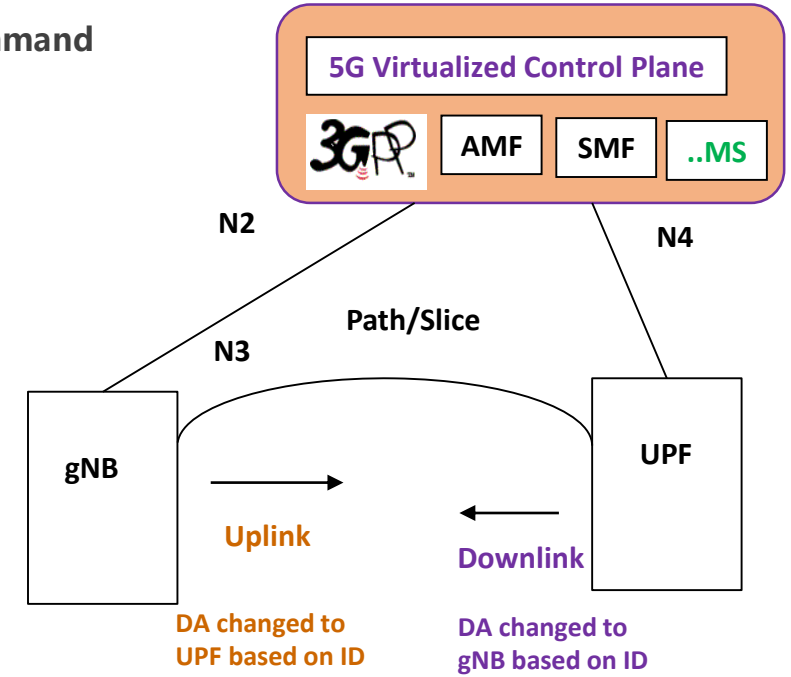
Summary:
Meta Network Operations (aka commands)for Mobility

- For regular uplink/downlink data packets:
  - Change-DA command, Session-LB command and Remove BPP block command (this can be implicit) would present
- During Mobility event:
  - In the downlink direction: Buffer-start, Buffer-Continue and Release-Buffer Instructions
- Metadata in the BPP:
  - Similar to UDP port numbers for increasing entropy at the receiving side (session load balancing)
  - VPN (label/Identifier) Information
  - Original DST address information

BPP Command1 (Novel Tunnel Free solution)

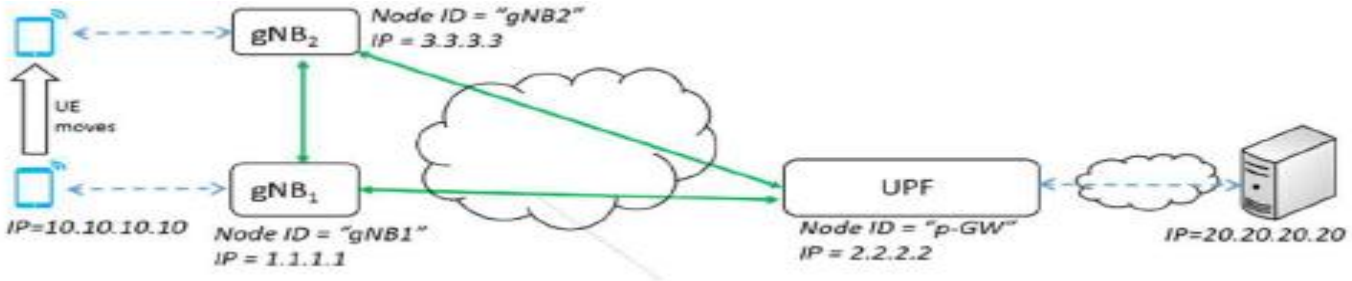
Big Packet Protocol Block – Change Destination Address Command

- For an incoming uplink packet (from UE) at gNB, destination IP address is set one of the addresses in the internet and Source address is the UE IP address.
- At gNB this address is replaced with UPF IP address (as indicated by MS with ID/IP in the pkt. after resolution). If the next-hdr is TCP/UDP checksum is updated.
- Actual destination address is set as parameter for conditional Change destination address instruction". Condition is if packet-destination-IP = one-of the-local-ips-of-the-node.
- In the downlink direction similar process happens; but in this destination IP is UE IP which is replaced with gNB IP as resolved with ID.



BPP Command1 - Encoding

Change Destination Address Command - Encoding



IP Packet: DA Changed from 20.20.20.20 ➔ 2.2.2.2 with BPP Change-DA Conditional Instruction

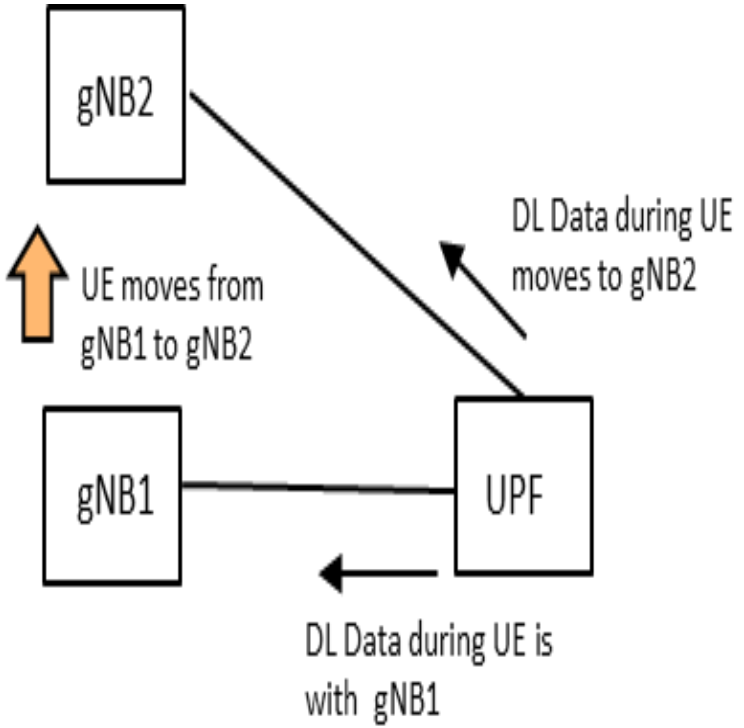
{[eth type=BPP][IPv4/IPv6][BPP HDR, Command Block, Metadata Block], Payload}

```
Cmd1:
  cond: eq (par.meta "node-id" (offset 0, len4), par.data "node-id");
  action: swap
     (par.meta "original-da" (offset 4, len 4), par.data "hdr-dest-address");
  action: session-lb
    (par.meta "load-balance tag" (offset 8 len 2));
  action: serialize-next strip-bpp;
  action: checksum-regenerate;
```

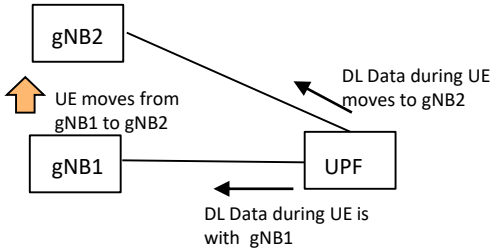
BPP Command2

Big Packet Block – Offload/Buffering related Command

- For some of the QCI/bearers it is extremely critical to buffer the data during mobility events.
- The reason for buffering needed is not to loose packets during UE air interface reconfiguration time (only for this duration).
- This is needed for today VoLTE traffic and more critical for other QCIs with extreme reliability of network traffic is required.
- Here BPP instructions are “Buffering Start”, “Keep Buffering” “Buffer Release” with Sequence number as indicated in the metadata of the packet.
- Either gNB or UPF can set this instructions as these nodes would no when the mobility event occurred/completed (5GC)
- A tool for control plane signaling reduction (further study on going)



BPP Command2 - Encoding



Offload/Buffering related command – Encoding

```
Cmd1: --buffer-start
 cond: eq (par.meta ("pkt-dst-ip"), par.data "node-id");
 action: start-buffer-packet ;
 metadata: 4-byte-sequence-number
```

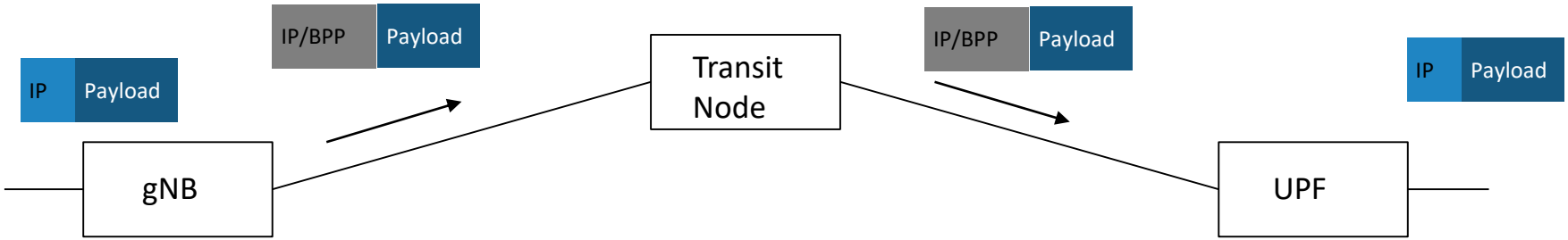
```
Cmd1: --continue-buffering;
 cond: eq (par.meta ("pkt-dst-ip"), par.data "node-id");
 action: continue-buffering;
 metadata: 4-byte-sequence-number
```

```
Cmd1: --release-buffer;
 cond: eq (par.meta ("pkt-dst-ip"), par.data "node-id");
 action: release-buffer;
 metadata: 4-byte-sequence-number
```

Note: Different packets will contain different commands (depending on the dynamic context); they will not all be in the same packet

Out of scope: giving “early warning” to allow preventive action when buffer is about to be exhausted (function of buffer size, delay, and packet size)

BPP Block with all instructions– Uplink traffic



BPP

C1: Conditional buffering Instruction with SEQ no as parameter (Optional)
C2: Conditional VPN Command with multiple parameters
C3: Change DA instruction with address to be restored as parameter
M1: VPN Label meta data
M2: For QoS/Entropy meta
M3: Other optional OAM data
M4: Original DA metadata

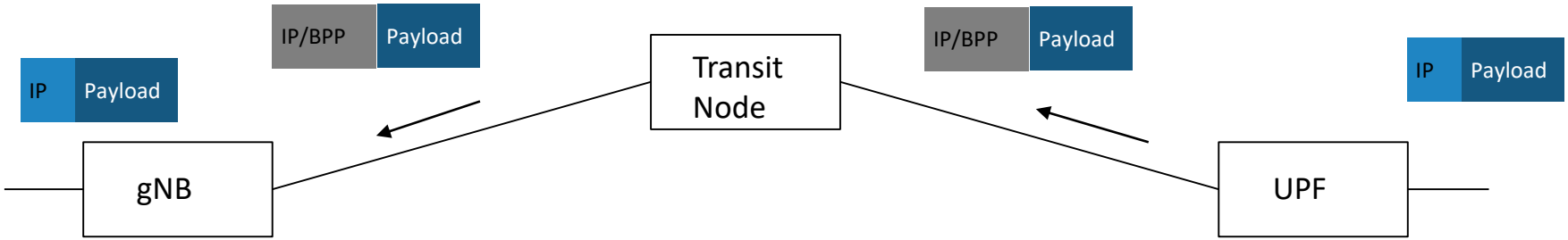
BPP

C1: No-OP
C2: No-OP
C3: No-OP

BPP

C1: No-OP
C2: Execute
C3: Execute
Use Metadata

BPP Block with all instructions– Downlink traffic



BPP

C1: Execute and set the correct next-hop
C2: No-OP
C3: Execute
C4: Execute
Use Metadata

BPP

C1: No-OP
C2: No-OP
C3: No-OP
Use Some of Metadata

BPP

C1: Conditional buffering Instruction with SEQ no as parameter (Optional)
C2: Conditional VPN Instruction with multiple parameters
C3: Change DA instruction with address to be restored as parameter
M1: VPN Label Metadata
M2: For QoS/Entropy meta
M3: Other optional OAM data
M4: Original DA

Summary

- This lays out basic BPP primitives and how this can be used in 5G basic and HO scenario to reduce various overheads (control and data plane)
- Conditional buffering command for various existing HO scenario
- BPP data plane allows a novel approach to enables high-precision service in the transport to meet various 5G slicing requirements w.r.t stringent latency and jitter
- Additional QoS parameters can be encoded as the meta data to be used along the transport path

Use Cases and Example Applications

- Time-Centric Forwarding for High-Precision SLOs
- A Greenfield Service for Data Mobility
- Operational Flow Profiling

Problem context

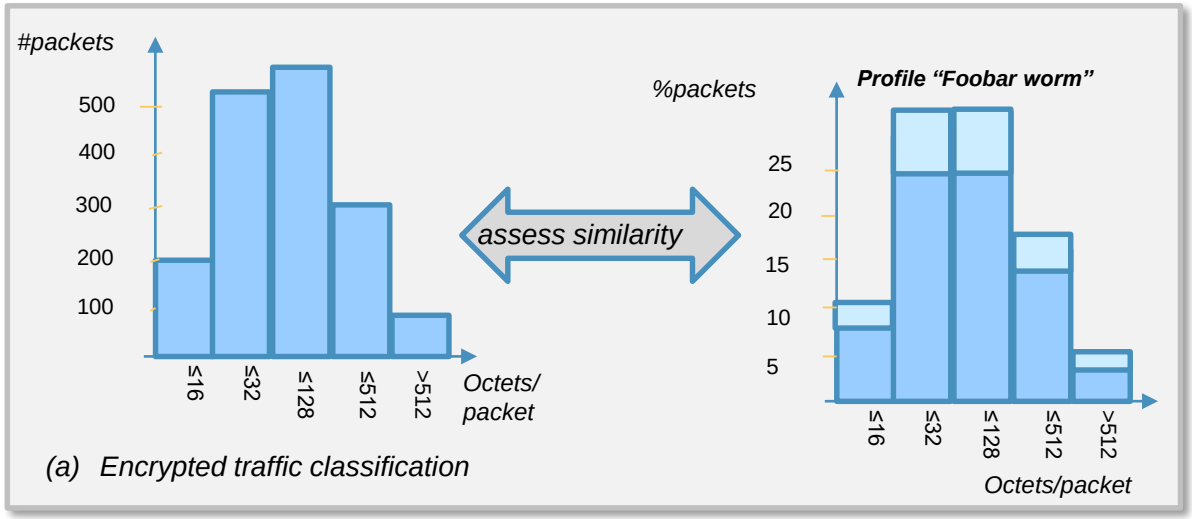
- Gaining insights into network and flow behavior is key to many operational problems
  - Understanding service level degradation & fluctuation
  - Detection of security threats
  - Optimization of network performance
- Need simple ways to capture and analyze the essence of what occurs during a flow
  - Consider and correlate flow packets and operational context
  - Dynamic definition and on-the-fly extensibility
  - Low overhead for generation, collection, and analysis
  - Real-time

OFP Concept

- OFP: A profile of operational characteristics, of flow and packet properties, and other context encountered by packets of a flow
  - A “profile” refers to some kind of aggregation intended to flesh out certain insights about the flow
  - Example aggregations: distributions / histograms, top(n), means, ...
  - Profiled data not limited to packet properties
- Applications for monitoring, troubleshooting, assurance, optimization, ...
- Observation: histograms are good indicators of distributions and outliers
  - Vary number of buckets
  - Vary boundaries between buckets
  - Vary underlying data item to profile
  - Vary criteria/conditions which to profile (e.g. every packet, or not)

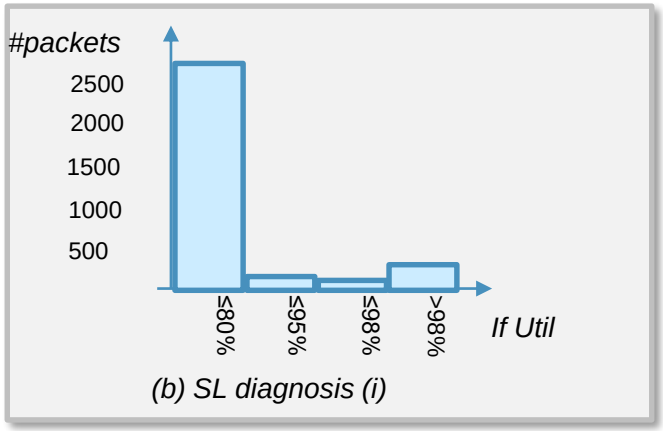
Examples

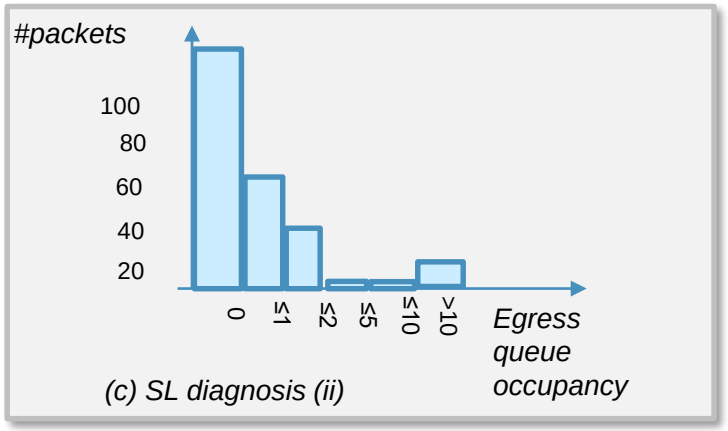
- Classical example: flow profiling
- Histogram with buckets to visualize distribution of packet
- Use to categorize encrypted traffic, e.g. for security purposes



(a) Encrypted traffic classification

Examples (contd)





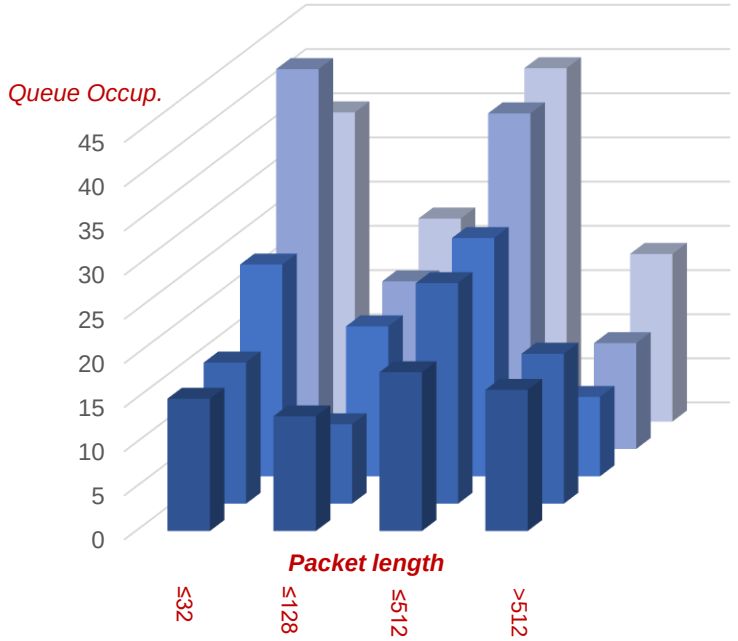
Number of packets encountering high ifUtil may help explain packet drops

Varying egress queue occupancy may help explain jitter, give measure of resource reservation effectiveness

Examples (contd.)

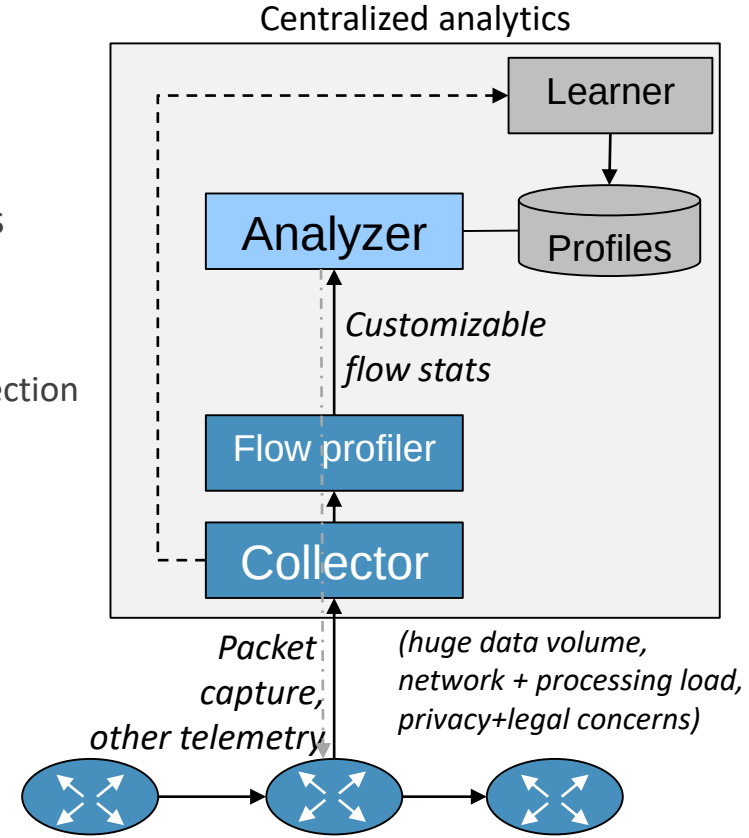
Multidimensional profiles

- E.g. separate queue occupancy stats by packet length
- E.g. queue occupancy stats across a path (assuming collection of stats across the path)



Existing solutions

- Network analytics are all the rage
  - Crunch large data volumes for operational insights
    - Insights useful but limited (e.g. trends, anomalies of time series)
    - ML adds classification but data generation and collection still bottleneck
  - Expensive (data generation/collection + analytics)
  - Data sources include
    - Packet capture
    - iOAM records
- Netflow/IPFIX as embedded alternative to reduce data volume
  - Output combineable with further analytics
  - Relatively static, limited in what can be aggregated



Network-Programmable OFPs using BPP

Network-Programmable Operational Flow Profiling provides

- Powerful network analytics
- delivering answers to network analytics questions that no other solution can provide today
- dynamically programmable by users
- with superior scaling and real-time characteristics, maintaining a current profile while the flow is occurring
- without need for generation and collection of large data volumes
- without dependency on centralized analytics framework
- performed embedded inside the network using BPP

Basic algorithm

Action "histo-***x***-uint***y***-incr", parameters:

- x: number of buckets (one of 3,4,5,7,10)
- y: uint 8/16/32 to imply bucket size 1, 2, or 4
- Target data item
- Bucket boundaries (x-1 parameters)
- Histogram data location to update

- If no OFP/statelet, initialize
  - Number and size of buckets, initialized to 0
  - Flow expiration / inactivity timer
- Update OFP
  - Fetch target data item
  - Apply aggregation function [f(ofp, item)]: select matching bucket and increment
  - Can introduce conditions: e.g. only consider when packet length > x, or ifUtil > x)
- Export on demand, on certain intervals or milestones, on flow expiration, on conditions
  - Export methods tbd, e.g. piggyback as metadata & extract from edge or separate export

BPP Mapping

- Example: Histogram of egress-queue-depth, 5 buckets

```
Cmd1: serialize-next;
      cond not statelet;
      action statelet-initialize
        (par.value 20, par.value 15);
Cmd2:
  cond: -;
  action: histo-5-uint32-incr
    (par.data "e-q-depth",
     par.meta (offset 0, size 1),
     par.meta (offset 1, size 1),
     par.meta (offset 2, size 1),
     par.meta (offset 3, size 1),
     par.statelet (offset 0, size 20));
```

Optimizations conceivable
e.g. include Cmd1 in 1st packet only

Size 20 octets fits 5 buckets with 4 octets each

Expiration (inactivity) 15 seconds

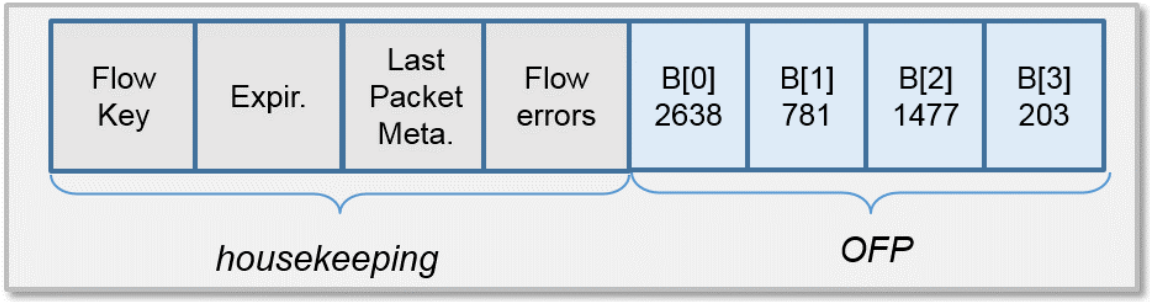
Bucket boundaries in metadata,
variations conceivable:
(a) Include value directly (par.value);
(b) Cache boundaries in statelet

BPP Mapping

- Example: Histogram of egress-queue-depth encountered by packets of length > 512 and egress-if-util > 90%, 5 buckets

```
Cmd1: serialize-next;
      cond not statelet;
      action statelet-initialize
        (par.value 20, par.value 15);
Cmd2:
  cond: [and-next] gt (par.data packetlength, par.value 512);
  cond: gt (par.data egress-if-util, par.value 90);
  action: histo-5-uint32-incr
    (par.data "e-q-depth",
     par.meta (offset 0, size 1),
     par.meta (offset 1, size 1),
     par.meta (offset 2, size 1),
     par.meta (offset 3, size 1),
     par.statelet (offset 0, size 20));
```

Statelet structure



- Notes:
  - Could store OFP metadata in variation (i.e. histogram description)
    - Cut down on BPP Block Size
    - Stateful implementation; caching, initialization, path congruency considerations apply
  - Update errors can constitute separate “bucket” item

Discussion

- Performance-optimizations possible to allow for packet forwarding before OFP update is completed
  - Non-blocking commands for statelet updates
  - Requires extensions to packet processing architecture → research opportunity

We believe that Operational Flow Profiles can leapfrog the current state-of-the-art in network analytics

- *Compare impact of Netflow, IOAM mindshare in datacenter, etc*
- *Strengthens Service Assurance and ability to provide Service Level Guarantees – supports High-Precision Theme*
- *Multiple applications in monitoring, security, diagnosis, automation*

Outline

- Motivation: Why are we here?
- Existing approaches and state-of-the-art
- New Networking and Network Programming Framework: BPP
- Use cases and example applications
- Open research questions
- Conclusions

Preamble

- BPP is still largely at concept stage
  - PoC under way but experiences need to be gained at every level, eg.
    - Implementation & mapping BPP into hardware
    - BPP SDK to develop and test BPP apps
    - Deployment and operations of BPP framework
  - Various gaps to filled, features to be added
    - from export of statelet data to non-blocking extensions
- Intended as one response to challenges for future networks
  - “Networkless” – from vendor-, to SP-, to customer/administrator-defined
  - Ability to rapidly extend and introduce new functionality
  - Holistic vs piecemeal technologies
- Many open research questions (some BPP-specific, some general)

Research topic: Data insertion into packets

- Problem: added content increases packet length, leading to potential MTU issues
  - Static – on creation of the BPP packet
  - Dynamic – in network transit
  - Problem is not unique to BPP: e.g. SRv6, NSH, iOAM, ...
- What's needed: solutions that avoid impacting usable packet payload
- Possible solution approaches
  - Fragmentation and reassembly, considerations:
    - Corner cases for selected BPP program semantics such as counter updates (avoid double-execution of commands for segments), data items based on packet properties (e.g. packet length, semantics of packet arrival times), consistent condition evaluation across fragments
    - Error handling (in case of error in one packet not another)
    - Inter-fragment path changes (reroute protection)
  - Complementation with controllers and command caching to mitigate issues
  - Restrictions to certain use cases, deployment scenarios – many are still covered

Research topic: Line-Rate Processing of Programmable Flow Logic

- Problem: Facilitating processing of programmable commands at line rate
- What's needed
  - Support of BPP (and similar) programming by processor architectures
    - Pipelining today assumes fixed-length processing
    - Need ability to deal with variable fields, variable numbers of commands
  - Facilitation of concurrent processing and support for non-blocking extensions
  - Data item instrumentation for fast access
- Possible solution approaches (important aspects such as mapping to HW tbd)
  - P4 with extensions (to overcome limitations for variable length fields,multiple commands)
  - Pipeline evolution: multiple command stages, parallelization, orthogonal command categories....
  - Serialization directives; non-blocking extensions (e.g. finish statelet update after forwarding)
  - Limiting BPP-style processing to control and signaling plane, not data plane

Research topic: End host and interdomain support and security

- Problem: Trust Boundaries between multiple domains
  - Signing, authenticating, authorizing commands and metadata in ways that are not prohibitive in terms of processing and packet overhead
  - Not unique to BPP – SFC, NSH, iOAM, ...
- What's needed
  - Secure ways to program packet and flow behavior from the edge that cannot be "abused" and tampered with
  - While allowing "legitimate" modifications to packets, metadata due to intended processing behavior
- Solution approaches
  - Limit deployments to single domains *(mitigation but not a true solution)*

Note: single domain networks are on the rise...
many use cases and deployment scenarios are still covered

Research topic: New accounting solutions

- Problem: how to provide incentives to “honor” processing of commands (specifically a problem with inter-domain deployments), and how to validate that it has taken place?
  - Should BPP commands be expected to “come free”?
  - Enable new business models, incentivize introduction of new services
  - Again, not unique to BPP - other domains face analogous problems
- What’s needed: accounting solutions for programmable packet processing
  - Validation of proper processing – packet-based, flow-based
  - Resource reservations (that can be charged)
  - Packet and flow compute as a service (that can be charged)
  - Pre-pay and post-pay variants
  - Avoid theft of “digital currency” and of services
- Solution approaches
  - Trusted/protected accounting instrumentation and statelet infrastructure

Research topic: Intelligent flow and traffic processing in the age of IPSEC

- Problem: how to provide smart treatment and custom programming of flows when traffic is encrypted and IPSEC is used?
  - Does not work with tunnel mode – need to process BPP Blocks
  - Transport mode is an option – ingress/egress gateways
- What's needed:
  - Solutions that permit for complete privacy while allowing for legitimate processing of traffic
- Solution approaches
  - "BPP over IPSec" – put BPP envelope over Ipsec traffic
  - Metadata tagging – clients telling network about the traffic (introducing follow-on problems, incl. who and how to tag, validating accuracy of tags, authentication, etc)

Research topic: Flow Programming Models and SDKs for "networkless networking"

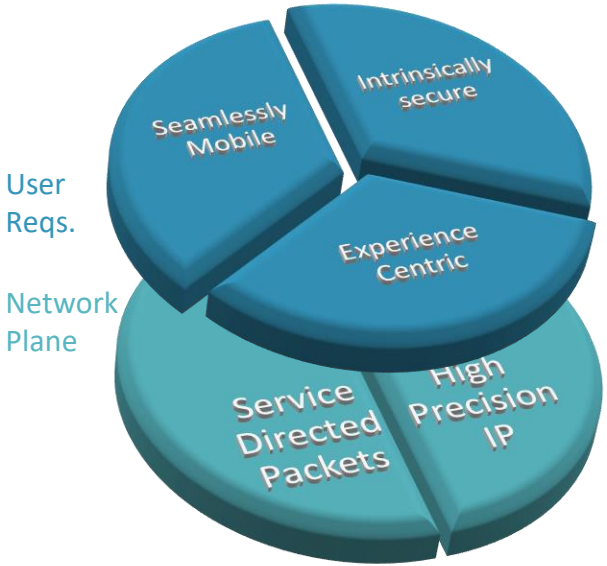
- Problem: Users need to be able to express "network intent", translate it into programming of flow behaviors, and validate intent in the network
- What's needed:
  - Proper sets of primitives that can be easily mapped to the network
  - SDKs and other tools that facilitate higher-layer abstractions (and automate their translation into the network), including dynamic behavior
  - A "lambda" equivalent for flow processing
  - Monitoring tools and test sandboxes
    (to validate actual flow behavior matches intent)
- Solution approaches:
  - P4, AN – Sprocket programming abstractions
  - Really, sorely lacking

Outline

- Motivation: Why are we here?
- Existing approaches and state-of-the-art
- New Networking and Network Programming Framework: BPP
- Use cases and example applications
- Open research questions
- Conclusions

In conclusion

- New networking requirements push current Internet technology to its limit
  - New services: VR/AR, Tactile, Industrial Internet
  - Ambitious & high-precision service level guarantees
  - Ease of operations, rapidly program, deploy, customize
  - Reduced acceptance of best effort approaches
- Emerging point solutions lead to complex + fragmented technology landscape
- Need for holistic solutions: rethink, unify, simplify
  - BPP provides one solution approach
  - Customize and program network, flow, packet behavior
  - Facilitate operational simplicity and visibility
  - Accessible to customers/administrators, not just vendors, MSDCs, and tier 1 SPs



BPP Contributions

- BPP is a new protocol and framework that enables high-precision networking for future networking applications
  - Programmable packets guide dynamic, context aware network processing for their flow
  - Empower users (enterprises, administrators, app developers, not just MSDCs / Tier 1 SPs) to customize and program flow and packet behavior
  - Applications range from operational simplicity and visibility to better networking services, overcoming current network limitations and ossification issues
  - “Networkless”: Networking infra is opaque to users – “manage flows not infra”
  - Isolate programming scope to given packets of a given flow –
    no device reprogramming, avoidance of problems with earlier approaches (e.g. AN)
  - Avoid fragmented point solutions,
    reduce dependency on standards and product cycles for new features

Where to go from here

- Our goals: articulate challenges, propose a particular solution, stimulate how we think about how we approach networking
- Next steps: PoC, practical validation, initial deployment experiences for selected use cases
- Many challenges and plenty of opportunities for future research remain:
  - End host and inter-domain security
  - Accounting solutions to incentivize custom flow processing
  - Programming Model and API/SDK for users and DevOps; network intent
  - Hardware extensions and support for flexible (e.g. variable length) packet and command processing
  - High-performance programmable flow cache infrastructure
  - and many more

Contact information

Alexander Clemm alexander.clemm@huawei.com

Uma Chunduri uma.chunduri@huawei.com

Padma Pillay-Esnault padma@huawei.com

Huawei R&D USA

2330 Central Expressway

Santa Clara, California 95050, USA

Thank you!

Outline

- Motivation: Why are we here?
- Existing approaches and state-of-the-art
- New Networking and Network Programming Framework: BPP
- Use cases and example applications
- Open research questions
- Conclusions

Alex – 20 minutes
Uma: IETF, P4 – 10 minutes
Padma: SDN, NFV, SFC – 10 minutes
Alex: AN – 5 minutes
Alex: Overview, Protocol, Examples – 45 minutes
COFFEE BREAK (30 minutes)
Padma: Engine, Deploy – 30 minutes
Alex: OFP – 15 minutes
Uma: mobile, routing – 15 minutes
Alex: HPC – 15 minutes
Alex – 15minutes