Towards an Architecture for Policy-Based Management of Software Defined Coalitions

C. Williams
Defence Science and Technology Laboratory, Salisbury, UK
cwilliams@dstl.gov.uk

E. Bertino
Dept. of Computer Science Perdue University, West Lafayette, IN, USA
bertino@perdue.edu

D. Verma, S. Calo
IBM, Yorktown Heights, NY, USA
{dverma, scalo}@us.ibm.com

K. Leung
EEE & Computer Science Depts., Imperial College, London
kin.leung@imperial.ac.uk

C. Dearlove
BAE Systems, Chelmsford, UK
chris.dearlove@baesystems.com

Abstract— **Future infrastructure systems will require increased flexibility and agility in order to respond to changing mission goals, external threats and complex environments. Two key enablers for such agility are Software Defined Networking (SDN) and Policy Based Management (PBM). This paper discusses the needs of SDN and PBM and suggests a common architecture that allows these two techniques to coexist and work synergistically. In particular the case of Software Defined Coalitions (SDC) is considered, where infrastructure resources (networking, storage and processing) are contributed from multiple partners to achieve a common goal.**

Keywords—SDN; PBM; SDN controller

I. Introduction

In the context of future multi-national coalitions, communications and computation infrastructure will need to operate efficiently under highly dynamic contexts, which could include changes to mission objectives, the external environment, threats and internal disruption. Dynamic coalitions represent a critical application domain as they require the ability to dynamically configure computing, networking, information, and service resources (collectively referred to as Software Defined Slices) based on requirements of specific coalition missions and operations. They require systems that are agile and adapt to changing context. Timely adaptation will requires autonomous behaviours when human response times are too slow. This requires a flexible infrastructure which can change its behaviours and configuration. On this basis Software Defined Networking (SDN) is an enabler to achieve flexibility of the infrastructure, operating on the basis of policies provided by a Policy Based Management (PBM) system to achieve the required agility. In principle, SDN and PBM provide complementary functions for the management of dynamic systems. However, in practice there is no clear distinction; the sub-components of both systems are not neatly delineated. Without careful design there could be unwanted interactions between their control/management processes. As a result it is not possible to treat them as independent systems. Instead their functions and processes should be carefully coupled in the system architecture. In this paper we discuss the research challenges in effective integration of SDN and PBM technologies for dynamic coalitions.

The paper is organized as follows. Section II describes the background to SDN, and the extension into Software Defined Slices/Coalitions, and Section III introduces PBM. The benefits of the integration of SDN and PBM are discussed in Section IV, and the proposed architecture is described in Section V.

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government notwithstanding any copyright notation hereon.

II. Software Defined Coalitions

A. *Background to Software Defined Networking*

In a conventional networked system each network device runs both data plane and control plane functionality. The control plane in every network device operates common protocols, such as for routing, through peer-to-peer communication without a central controller. This architecture has distributed control where all network devices run the same protocol and they exchange information peer-to-peer to collate knowledge and gain awareness about the global network state (network situation awareness). However such knowledge may be incomplete given the distributed nature of the architecture. Furthermore, network situation awareness is not necessarily shared due to its distributed nature (each device sees a subset of all measurements). Thus decisions within each device are taken by a local optimisation process based on locally derived network awareness. While the distributed nature is robust against network disruption/failure the optimisation algorithms achieve a sub-optimal result due to limited knowledge of the whole network's state.

In SDN the control plane and data plane are separated, such that devices (usually) only contain a programmable forwarding function within the data plane. SDN employs a logically centralised controller that has a global view of network state within a given network domain. However, in the tactical communications environment, where disruption to connectivity is likely, it is undesirable to have single points of failure, and hence the SDN controller may be physically distributed to provide resilience. There may also be issues caused by limited bandwidth combined with high volumes of network control traffic. While much of the SDN literature focuses on data plane devices as forwarding (routing) devices, as a first generalisation of current SDN approaches, the software in the data plane can be programmed with greater functionality. This enables devices to have a greater richness in how they processes packets/flows and allows devices to take on the roles of firewall or gateway node on demand (know at Network Function Virtualisation, NFV).

B. SDN architectures

Fig. 1 below shows a high level SDN system architecture. Central to the architecture is the control layer in which SDN controllers (SDNC) exist. An SDNC connects to a set of networking devices under common management control (common ownership), creating a controller domain. Within the control layer are network services such as routing, mobility and traffic engineering. Where multiple SDNCs coexist then coordination between them is though the East-West Interface (EWI). The control layer takes service requests from the application layer, via the North-Bound Interface (NBI), and carries out the necessary orchestration and resource provisioning, to match the available resources in the infrastructure layer to meet the service requirements. Application services provide the guidance and objectives against which the SDNC optimises its behaviours and the decisions it makes relating to the device level programming and policy formulation. The South-Bound Interface (SBI) provides control information to routing devices (e.g. routing table entries or programming). The SBI also supports reverse information flows such as event based messages, flow statistics and messaging when received packets do not match routing table entries (seeking direction on what to do, else the packet will be dropped).

A strength of the SDN approach is the use of abstraction, so the control layer does not need to know the implementation details of each device. Within each device is an API to map the provided control information to device specific instructions. Similarly, the application services do not need to know the network implementation details. Communication between the layers is through open standards, thus allowing heterogeneity of implementations to coexist.

The EWI provides communication between controller domains (inter-domain) or between control entities in the same domain when the control plane is distributed across multiple controllers (intra-domain). The relationship between controllers can be peer-to-peer or hierarchical.

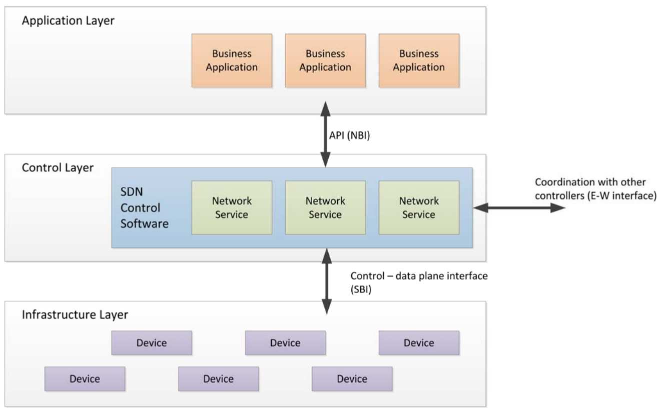

Fig. 1 SDN Architecture

The SDNC is the intelligent entity that controls resources to deliver network services. Its core function is the real-time multi-dimensional convergence of a changing resource environment and a changing service demand environment toward an optimum, where the optimisation criteria may also change over time [1]. The controller abstracts resource demands on physical resources to enable resource sharing, load balancing and to support redundancy of resource provision for resilience. Key functions of the SDNC are to:

- Capture and negotiate application service requests
- Determine device availability and capability
- Orchestrate resources to satisfy service demands
- Arbitrate over resource conflicts
- Build network situation awareness
- Manage network topology
- Manage security
- Provide configuration, programming and policy to infrastructure devices
- Coordinate with other controllers for traffic traversing multiple controller domains, for distributed controllers or for shared resource.

Coordination between controllers requires a distinct subset of network services. A detailed proposal given in [2] is expanded in the later discussion.

C. Distributed SDN control

In the tactical environment completely removing control functionality from the forwarding devices is not desirable, since this would incur significant overheads if the data plane is continuously 'chatting' to the control plane to get directions for each packet [3]. Instead the SDNC can delegate some control functionality to the managed devices (referred to here as local control), alongside the data plane function. This would be updated as directed by the SDNC. Such a control approach allows the device nodes some autonomy, stopping continuous reference to the controller and allowing them to operate independently even when disconnected.

D. Software Defined Coalitions

An extension to SDN is to generalise the concept to include not just network components, but a range of other software defined capabilities and resources (SDx[1]), e.g. storage, computation, management. Thus a controller may not supervise devices of just a single class (e.g. just routers or just storage), but will control a number of device/resource classes to enable better optimisation of resources and balance of network loading. A related concept is that of Software Defined Environments (SDE), though to date the emphasis has been on SDEs within a data centre/enterprise environment [4].

Due to timescales (latency) of the control required, the extent of delegated control in the devices could change both as a function of resource type and context.

[1] Note: In this paper SDx will be used to denote the generic class of software defined resources (network, storage, computation), but excludes Software Defined Slices/Coalitions. A CoI is the superset of SDx/SDS/SDC.

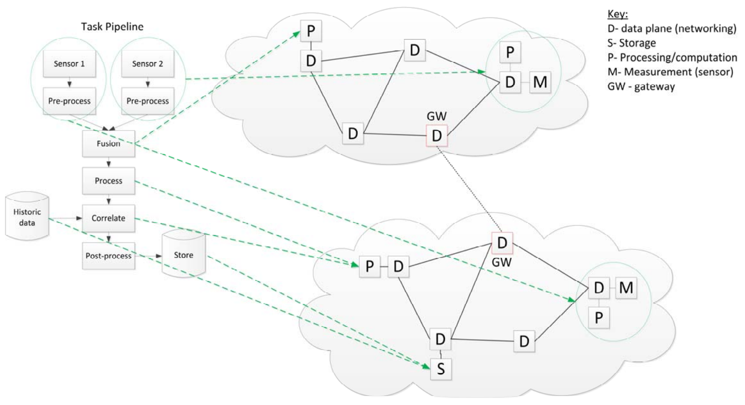

Fig. 2 Software Defined Slice

A key extension of the SDE concept is to allow subsets of software defined network/network-attached capabilities to be brought together into a Community of Interest (CoI) to work collaboratively for a common goal. As a minimum one CoI would be for networking with a logical SDNC, thus each network domain forms a CoI in its own right. Further, where there are multiple instances of networked storage, a storage CoI could support resilient distributed storage for other devices (such as managing caches, etc.). A similar approach can be applied to computation/processing.

A CoI has a common purpose that can be decomposed into a workflow. From this, resource requirements can be identified. The controller for that CoI is then responsible for orchestrating all the necessary resource to deliver the workflow, as shown in Fig. 2. While SDN models a routing pipeline (to transfer information), a generalised CoI system would create a more complex processing pipeline (task based). The resources in SDx sub-systems may be given to a CoI from management domains without shared ownership (e.g. different coalition partners). The capabilities brought together are collectively referred to as Software Defined Slice (SDS) consisting of the resources in the CoI and a control mechanism. When resources exist across network/nation boundaries, they are referred to as Software Defined Coalition (SDC). For simplicity the CoI controller mechanism is not shown in Fig. 2.

Unlike resource controllers (e.g. SDNC) which are permanent entities, a slice is typically a transitory structure, with an SDS control mechanism created on demand. A slice can be generated in response to a high level business demand (e.g. sensors/storage/networking/processing to support an ISR collect tasking) or in support of a lower level requirement focused on infrastructure operations (e.g. multi-partner cyber defence monitoring capability).

An initial set of requirements for a system to support the SDS concept are captured here:

- Resource discovery; for a slice to determine what resources are available to it, what their capabilities are, and what the constraints on their use are.
- Resource negotiation; the ability to negotiate with the resource owner on access, capacity, quality and prioritisation of resource allocation.
- Resource tasking/agreement; to subsequently task allocated resources according to the needs of the slice.
- Resource orchestration & management; the ongoing management of slice resources. This would include management of redundancy/resilience mechanisms.
- Resource conflict resolution; using task prioritisation.
- Performance monitoring & fault management; monitoring resource performance and its contribution to slice objectives, and responding to faults/failures/degradation in resource availability.
- Management of returning resources when not needed, or when the slice is closing down.

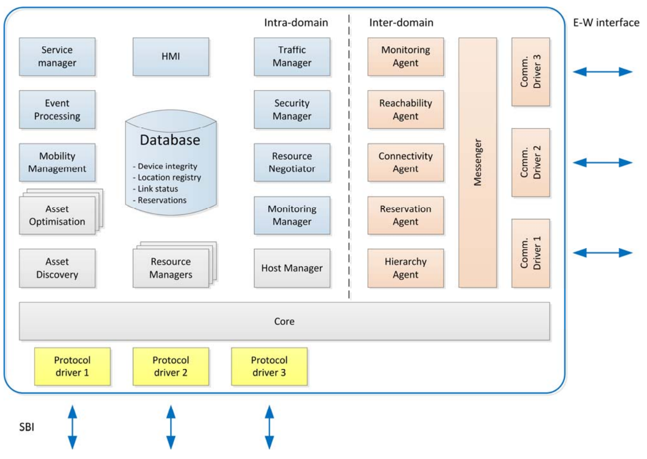

Fig. 3 SDS Controller (SDSC) Architecture

Based on these requirements, an SDSC architecture is outlined in Fig. 3, which is an extension of the architecture proposed in [2]. It generalises resource management, rather than just referring to networking functionality as in a SDNC. The communication across the hierarchy is through the EWI and requires a hierarchy management agent to ensure that the hierarchical precedence is properly enforced. The Resource Negotiator identifies the resource needs of the slice (from a process flow) and uses the output of the Asset Discovery service to find suitable assets/resources with which it enters into a negotiation. An agreement then allocates the resources (e.g. using policy interchange). The utilisation of allocated resources is then managed as though the SDSC owned the resource. Effectiveness of the resource management will be enhanced if it has a good appreciation of the dependencies and trades between resources, allowing it to optimise resource utilisation.

III. Policy Based Management

A policy is a conditional rule that can change the behaviour or state of the system by imposing constraints, mandating actions to be executed, or forbidding actions. In practice, policies are technology-independent rules aiming to enhance the hard-coded functionality of managed devices by introducing interpreted logic that can be dynamically changed without modifying the underlying implementation. This allows for a certain degree of programmability without the need to interrupt the operation of either the managed system or of the management system. PBM is a key enabler for agile systems that can respond dynamically to changes in the mission, threat or context/environment.

The use of policies allows management of systems to be abstracted away from the individual components, providing greater scalability and agility, particularly for large complex systems. Scalability is supported through defining policy domains (e.g. by QoS type, resilience, security, reliability, by device class) that are applied across sets of devices, without having to routinely write individual policy sets per device. This approach is analogous to 'mission command' in the human domain.

A policy statement is typically written in the form <Event><Condition><Action> (ECA), which is interpreted as "When <Event> occurs and <Condition> is true, then do <Action>". This can be stateless or stateful, where a policy state will imply which policies apply and will limit which policies' states the system can move to next. Key components of a PBM system are:

1. Policy management tool: Used by the administrator to input and manage policies into the system.
2. Policy Enforcement Points (PEP): Interfaces used by devices to require policy decisions with respect to specific actions, requests, situations.
3. Policy Repository: Stores policies generated by the management tool and makes them available to PDPs.
4. Policy Decision Points (PDP): Responsible for interpreting policies stored in the repository, taking policy decisions and communicating these decisions to PEPs.

Like SDN, there are issues to address within the military environment that include: availability of connectivity to support policy dissemination and monitoring of policy compliance, resilience of disconnection/intermittency to policy system components and the network overhead associated with policy flows. In the context of information transfer across multiple (e.g. coalition networks, policy enforcement needs to consider end to end policy issues).

IV. Bringing PBM and SDN together

The previous sections have presented the compelling arguments for PBM and SDN in isolation. Are there benefits in supporting both concepts together, and if so, how should this be done? The focus of the discussion will be on SDN, but the more general case of SDx is a natural extension. The next section then extends the discussion to SDS/SDC systems.

A common feature in both PBM and SDN is the use of abstraction, allowing the management and control of devices to be device independent. Both require a common information schema to be defined by a high level policy language, thus both will have APIs within devices that convert such schema into device dependent rules that need not be exposed externally. NFV employed alongside SDN supports the device-independent approach of PBM. Both PBM and SDx can have reasoning capabilities, and the relation of these reasoning capabilities is returned to shortly.

A. *How PBM helps SDN*

The software defined nature of devices provides the flexibility to push new functionality to devices, thus enhancing the ability to define new parameters as input to <event> and <condition> and for new <action>s to be defined, which were not in the original programming. So for example, new sensors can be deployed and alternative reasoning algorithms can be provided to the PDP/policy refinement functions.

B. *How SDN helps PBM*

The traffic management capabilities of an SDN can clearly enhance the resilience of policy dissemination. Further, a software defined storage approach can abstract the physical storage mechanism for the logical policy repository, thus a distributed storage solution could be employed that is transparent to the policy manager.

It is useful to summarise the set of policies that would be relevant to SDN based on the above implementation. These include policies for:

- Controlling information flows between networks and managing internetwork gateways
- Prioritising information flows and resource usage
- Deciding how different content types should be handled (e.g. QoS, paths allowed/disallowed)
- Selecting controllers and their placement
- Taking decisions about whether to create a distributed control plane, and with how many controllers
- Managing topology and routing, including setting up redundant pathways
- Addressing consistency requirements for network situation awareness
- Fault recovery approach, including what devices can do with controller disconnection
- Placement of functions on devices (e.g. relating to trust classes, or selection of gateway nodes).

The ability of a PBM system to generate (internally or from external input) new policies from the classes above, in response to changing circumstances, enables the controller to behave dynamically according to context [5]. Node and network mobility also requires policies to change (such as location dependence) and will support handover of nodes between networks. A policy system should also enable the consistent handling of information flows even between SDx management domains. Policy sets can be created for a group of devices or flow, and applied consistently across all partner infrastructures. Where a SDNC delegates some control functionality to devices, the degree of delegation can be defined in policy. After control has been delegated, further policies can dictate the ability of devices to refine the extent of freedom in refining the delegation.

Within any policy system, policy conflicts must be resolved. A further layer of policy instructions can be disseminated across the infrastructure providing guidance on how to resolve conflict, or how to report it.

Policy systems can create policy sets/domains with a common scope, and thus support overlapping policy domains. For example each flow could be considered a policy domain, as part of a larger structure (e.g. flow→network→slice→mission). Relevant flow policy is only needed by a device if the device is on the flow (the SDNC function decides relevance). In which case, efficiency savings of using multicast rather than broadcast may help to reduce network overhead of policy dissemination. For bootstrapping devices, there could be a generic policy set that acts as a default (which would include broadcast flows) that is then refined/replaced by specific flow policy sets. A device with multiple functionalities (not just routing/forwarding, but firewall, storage/cache, compute, etc.) would subscribe to the relevant policy sets (e.g. policy on where/when/which data to cache alongside flow forwarding table for network attached storage that has a routing capability).

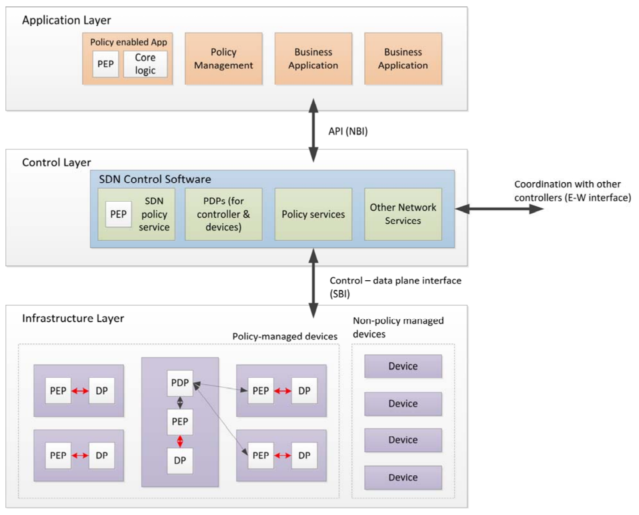

Fig. 4 Proposed SDS-PDM Architecture

In a dynamic tactical environment it may not be possible to generate the relevant policies if they are context dependent, and so flow based policies will be generated on-the-fly per session. Applications create guiding principles across flows, and the SDxC refines the per flow policy.

C. Proposed PBM-SDS architecture

While SDS and PBM appear complementary, as a consequence of the multiple management/control entities within these systems, we need an architecture that provides coherence between entities and that scales to large systems (many controllers). Fig. 4 outlines the proposed architecture.

Conceptually, this is a service based approach. The Policy Manager is a business application and is part of the Application Layer. This layer also includes policy-managed applications, each of which contains a PEP. Indeed, PEPs exist at all layers. Other policy components not within the end devices are located within the Control Layer (e.g. PDPs, local policy repositories), and are Network Services alongside SDS control services. Examples of a policy service would be policy refinement services with enhanced reasoning capabilities or a service that monitors policy conformance and reports back to the Policy Manager. Supporting such monitoring capability will require instrumentation in the devices and controller in order to make a determination of conformance. Note that one of the policy end points will be the SDS controller itself, and so a PEP must exist in the controller as shown in the SDS policy service. This ensures other services in the control layer have the relevant policies and are reporting back to the Policy Manager.

The Policy Manager provides a set of obligations and constraints to the control layer as a subset of the NBI information schema. The SBI carries policy information to the devices. As the system supports increasing diversity of information flows over the SBI in order to support policy exchange and control of network/storage/computation devices, there will be a base communication mechanism between controller and devices. The base mechanism will transport different information schema such as the ones expressed according to policy languages such as OpenFlow. This is consistent with the multiple protocols supported by the SBI shown in Fig. 3. A single standard to capture all necessary information schema across the SBI is unlikely to be sufficiently scalable and extensible.

As illustrated in Fig. 4 policy exchanges over the SBI could be information to either a PDP or a PEP. This information, while of different natures (relating to the degrees of freedom it conveys to the device), will be expressed according to a common policy language, and the distinction does not matter to the SBI. In a policy-controlled device the PEP translates the policy language into device-specific rules, but like an SDS system, the device would have an API that translates other SBI information into a device-specific set of instructions. Internal to the device the SDS API and PEP could be merged. Maintaining a PEP in the devices improves interoperability since device communication is via open standards, and controllers can maintain device implementation abstraction. Moving the PDP from the device to the controller minimises the computation needed in the device, but maintaining a large set of PDPs in the controller may cause significant processing burden, depending on the type of policy algorithms employed. With PDPs in the controller, policy conflict requiring communication between PDPs to resolve the conflict (where a higher level policy does not exist) would not place any additional load on the network. Further, with PDPs embedded in the controller, the controller can more easily participate in conflict resolution (e.g. providing situational awareness/context information, or supporting communication with other networks (which may be on the data path) via its EWI. A compromise is for some devices to host a PDP that serves policy to a number of local devices that only have PEP functionality (see Fig. 4).

A PDP internal to the controller would reason over the relevant policy sets (it would be responsible for getting them from a Policy Server Service). It would take sensor input from other controller services to evaluate <Event> and <Condition> and then would provide the resulting <Action> to the relevant Network Services in order to instantiate the policy in controller behaviours or sent the action to the relevant device/set of devices. The PDP merges policy sets, so that the PEP does not have to explicitly discriminate between services/slices, but merely carries out the forwarding /storage/compute functions as directed (based on packet headers). Thus if the output from the PDP is conflict free the PEP does not need conflict identification/resolution mechanisms. Assuring that the PEP does not have to deal with conflict requires that that the PEP must only receive input from one PDP. However, one PDP can serve multiple devices, each with its own PEP. When one PDP serves multiple devices, a policy is required to determine in which device the PDP could/should be located.

Feedback on system state and performance is particularly important for policy refinement. How do you know that the system is implementing the intended policy goals? What if the refinement process leads to actionable policies that are infeasible, i.e. cannot be executed by the system? Thus devices need to advertise their capabilities and the policy refinement process needs to take these constraints into account. This requires capability for discovery/advertisement processes. This is particularly of concern in dynamic coalition environments, where assets may be contributed from other partners.

D. SDS Controller (SDSC) aspects

Following the system architecture description, more detailed consideration is now given to the SDSC. The SDSC takes policies from its internal PEP, which then influence the decisions it makes in relation to:

- Instructions over the SBI to devices and how they behave. Policies at this level take advantage of the network/context awareness within the SDSC.
- Interactions with other SDSCs, particularly to understand prioritisation and handling of flows that traverse SDSC domain boundaries.
- Selection of devices that can support the SDSC (e.g. only on trusted hardware or one with TPM capability, requirement for physically secure location).
- Relocation of the SDSC or to fragment the control plane.
- Placement of SDSCs in a distributed controller environment.
- Allowable interactions with other controllers (especially where there are differing degrees of trust).
- Degree of delegated control, and how it is monitored.

In a multiple controller environment (e.g. through distributed SDNC or with SDS/SDC) the EWI needs to convey policy information. As with the SBI, while controllers need to agree on the policy language to use, the definition of the EWI may just be a transport mechanism that can support a number of policy languages, to allow for extensibility. Policy can provide direction on behaviours when the network state is unknown or uncertain, or where it is stale, in order to address issues of consistency of network state in a multi-controller environment. Where controllers become disconnected, policies can be applied to change behaviours, or to change connectivity mechanisms, such as switching between in-band and out-of-band controller communication.

As already discussed, delegated control functionality from the controller to devices provides resilience to controller disconnection and the ability to reason in changing contexts even without controller connectivity. Thus control delegation may reduce the need for controller replication/fragmentation. Policy instructions can be used to control the delegation and can define/constrain the reasoning behaviours. Thus control delegation can be managed dynamically and even decided locally if the node believes connectivity to the controller is at risk (or has already gone). Variations are likely to arise when balancing control/autonomy across devices and device classes, thus controllers need to be aware of device capabilities before delegating control.

Since the PDP has reasoning abilities as well as any delegated control from the SDSC, merging these reasoning functions into a single function within a device will ensure consistency. To enable this, the policy language and other SBI information exchanges need a common information schema.

For delegation of control and policy refinement, some form of device-based computation is required. A benefit of SDS and PBM combined, is that new reasoning/refinement algorithms can be transferred to devices and their behaviour managed through appropriate policy sets. Thus reasoning/refinement algorithms have their own policy domain.

V. Conclusions

In this paper, we have analysed the two paradigms of SDN and PBM, and proposed a merged architecture that allows a system to use the best features of both technologies and improve agility in operation. We have provided the detail of this architecture in the context of dynamic coalitions, and discussed how the components of SDN and PBM can be combined to create a consolidated architecture. In future work, we plan to continue the refinement of the architecture to address specific scenarios that arise in coalition contexts.

References

[1] Kim, H., Feamster, N., "Improving network management with software defined networking", IEEE Communications Magazine 51, no. 2 (2013).

[2] Phemius, K., Bouet, M, Leguay, J., "DISCO: Distributed Multi-domain SDN Controllers", NOMS 2014.

[3] Curtis, A., et al, “DevoFlow: Scaling Flow Management for High-performance Networks”, Sigcomm 2011

[4] Li, C-S., et al. "Software defined environments: An introduction." *IBM Journal of Research and Development* 58, no. 2/3 (2014): 1-1.

[5] Verma, D., et. al., "Dynamic and adaptive policy models for coalition operations." In SPIE Defense+ Security (2017): 1019006-1019006