Sharing parts of network switches and flowspaces is possible by using network virtualization tools like FlowVisor [14] within a single policy domain. Also, several standardization bodies, such as ONF (Open Network Foundation), IETF (Internet Engineering Task Force) and OIF (Optical Internet Forum), are developing the initial standard for coordinating SDN controllers by exposing abstracted network information via software API (called north-bound interface or NBI). However, both multi-domain control delegation of part of a network and SDN network peering require partial information sharing. Since this sharing relates to domain policy enforcement, this remains a challenging issue.

In [3], the authors point out that the peering model defined by the IETF for GMPLS does not account for organizational boundaries where full information sharing, as the IETF proposed, is prohibited. The IETF overlay model, which calls for zero information sharing, is more compatible with the current approach with different autonomous systems (ASes) and the widespread adoption of IP/BGP. However, inefficiencies in multi-domain networking exist as a result of limited information sharing and conflicting hidden policies (e.g. hot-potato routing). The efforts in [4, 5, 11] illuminate more granularity between all-or-nothing sharing where information for path computation is shared. The works of [10, 7] propose inter-domain route computation to be completed by a trusted third-party with additional knowledge of multiple domains. This is in contrast to our implementation which does not rely on full information sharing or third-party consultation.

3. CP-CP SHARING, SYNCHRONIZATION

In traditional computer networks, each device in the network may be capable of data-plane (DP) actions and control-plane (CP) actions. In this section, we discuss our approach to information sharing and message passing between different control-plane actors (CPAs). CPAs may carry out functions such as calculating spanning trees, OSPF, RIP, and BGP as well as resource reservation. Given the diverse goals of CPAs, the CP-CP information exchange format should support the dissemination, synchronization, and negotiation of any general purpose CP data. The following subsections outline the basic mechanisms that are required for useful CP information exchange and explore the particular challenges of synchronized dissemination of consistent changes to shared graphs.

3.1 Graphs for CP-CP

People naturally treat networks as graphs, whether they are writing software to implement network-based services, provisioning VPNs, or trying to figure out how to implement BGP policies. Our approach stores information in a graph's vertices, edges, and attributes to represent an entire network topology, link states, policies, metrics, and associated meta data. Specific regions of the graph can correspond to the topology of a specific layer (e.g. optical switches or OpenFlow switches), some abstraction (e.g., a "connection" that represents a path with a bandwidth commitment or perhaps a virtual router), or a set of dependencies.

A subgraph is a restriction of a graph based on some criterion. By restricting parts of the graph that directly connect to certain DP resources, the subgraph can contain a subset of a network's nodes, links, ports, traffic classes, policies, etc. By restricting attributes of the vertices and edges, a subgraph can even represent a subset of a network's logical resources (e.g. MPLS label space or bandwidth).

3.2 Subgraph Versions and Shadowing

Figure 1 illustrates how network resources may be shared by a CPA. In this example, CPA A has a graph and wishes to carry out a CP function that requires sharing a subgraph with another CPA. It creates a "subgraph sharing" node, which points to the vertices and edges that it wishes to share (illustrated with striped vertex pattern). A message is passed to the target CPA containing all of the information required to reconstruct the subgraph (i.e., shared vertices, edges, and attributes). Note that the same vertices and edges can be shared with multiple other CPAs. The shared subgraph is instantiated by the receiving CPA and can be integrated into its existing local graph.

Each subgraph element has a value associated with it of the form OWNER:VERSION, where OWNER indicates the only CPA that is allowed to edit that element and VERSION is the version number. Each edit by the OWNER increments the VERSION number. A CPA can request that the current owner transfer ownership. If the request is accepted, the requesting CPA is given ownership of that element.

In Figure 1a, the shadow set is identified as A:1, A:2, and A:3 by CPA A. Automatic sharing rules are set up to add head and tail vertices of shared edges if they are not already part of the shadow set. As a result, edge A:1 causes vertices A:4 and A:5 to be added as well. Edge A:2 additionally includes vertex A:6.

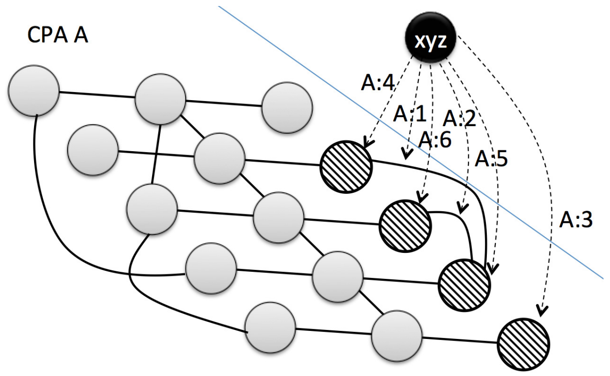
In Figure 1b, the shadow set, A:1 to A:6 is replicated in CPA C. CPA C is aware of the provenance of A:1 to A:6 as originating in CPA A. Likewise, when CPA C adds edge C:1, a separate auto-sharing rule is consulted to see if edges added between shadowed vertices should be added automatically to the shadowing set. In this case, it should be added, causing edge C:1 to be shadowed back to CPA A. This keeps the two shadow copies of the subgraph consistent with each other.

Whenever a CPA with a shadow copy of a subgraph modifies it, updates are forwarded to the other CPAs informing them of the changes. If the CP network (which connects the CPAs) is connected, then all CPAs can evaluate the update and determine if it affects them. If the CP network is partitioned, then the update only is able to reach the CPAs in the same partition as the originating CPA. When the CP network heals, neighboring CPAs exchange digests that allow them to identify and request changes they have missed.

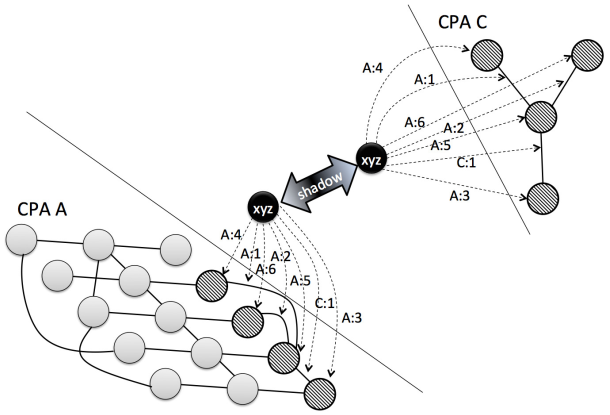
3.3 Event Graphs

It is sometimes important to be able to impose order and pre-conditions on certain subgraph actions. For example, it may be desirable to make a new flow or forwarding path before breaking an old one or to fully set up all the links in a path before allowing packets to flow onto the first hop. Additionally, it may be desirable to roll back to a previous state if an error occurs. To this end we have implemented a batch transaction structure (BT) (illustrated in Figure 2). It specifies a set of BTs that must occur before it will be executed, a set of actions (callback functions) to be evaluated and a set of new BTs that will be scheduled depending on whether the actions succeed or fail. BTs allow a single atomic event to achieve the following goals.

1. Demarcate the beginning and ending of a database



(a) Shadow Set Node Points to Subgraph to Share



(b) New Version Triggering a Shadow update

Figure 1: Components Involved for Shadowing Across Multiple Control Plane Agents (CPAs)

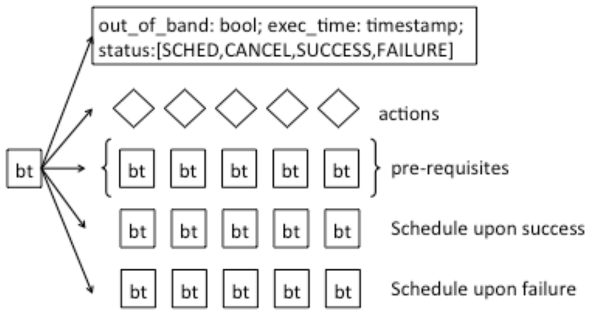

out_of_band: bool; exec_time: timestamp;
status:[SCHED,CANCEL,SUCCESS,FAILURE]
actions
bt
pre-requisites
Schedule upon success
Schedule upon failure

Figure 2: Batch Transaction Structure

transaction for triggering and shadowing

2. Execute and accumulate status on several function call actions

3. Represent task-graph joins by gating execution until all pre-condition events have terminated

4. Add subsequent batch transactions onto the timeline based on the aggregate success or failure of the action list.

3.4 Triggers

Triggers provide a mechanism for specifying functions that should be automatically invoked when certain actions are performed on a graph. As such, triggers provide a behavioral dimension to the graph data, and in some cases, eliminate the need for explicit inter-module APIs. For example, adding a virtual link (edge) to a virtual router abstraction (vertex) could trigger the invocation of a function that will execute a Dijkstra algorithm to compute the detailed path required to support the additional virtual link. In turn, attaching the vertices and edges of the detailed path could trigger network configuration actions. This example cascades an event through two or more software modules without explicit API calls and potentially without full inter-module awareness of the low-level actions being taken.

Triggering, shadowing, and graph versioning provide mechanisms to control the granularity of interactions between different CPAs. These mechanisms provide a cooperative and efficient data structure for CP-CP messaging, control, and synchronization.

3.5 Network Avatars and CP-CP Interaction

Inter-domain subgraph shadowing allows each domain to provide other networks with an avatar that allows it to control public visibility and interaction. To be more precise, a shadowed subgraph can be viewed as a virtual representation of a network that is made visible to remote CPAs and is continually updated as changes occur. Those remote CPAs can take action on the subgraphs and use them as tools to both control and observe the status, activity, and policies of the other domains.

This is an improvement to today's mostly opaque multi-domain visibility and guesswork interactions. In the next section, we discuss how the interactions of avatars can also trigger changes to the real network.

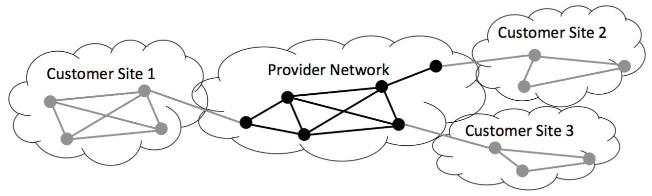
4. INTRA-CP AND CP-DP FUNCTIONS

When subgraphs are exchanged between different control domains (CP-CP), it may be preferable to exchange abstracted representations, leaving out detail of the real network that exists in either domain.

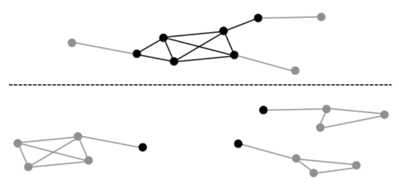
In this section, we discuss maintaining different levels of abstraction in a single CPA for this purpose. We provide examples at common network layers and explain how our implementation uses *rendering* both between abstraction levels and between the control plane and the data plane (CP-DP) to configure the physical network.

4.1 Abstractions

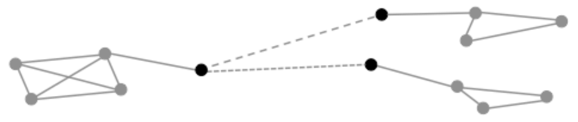
At the highest level of abstraction, graphs can provide a virtualized network representation that allows one to abstract away the details of a network topology. These high-level abstractions provide sufficient information to implement a service, but no more. For example, the forwarding and traffic management services of a given network could be represented as a virtual switch at layer 2 or layer 3. This would allow users to configure end-to-end forwarding, pri-



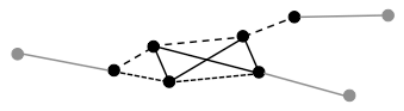
(a) Physical network



(b) Initial provider/customer views



(c) Customer view with Site 1 as hub with two MPLS LSPs (dashed)



(d) Provider view showing path of two LSPs

Figure 3: Multi-domain VPN provider and customer interactions through graph shadowing

ority and bandwidth without having to know the details of the network topology.

The next level down has a more detailed topological abstraction that captures the internal connectivity of the network, available bandwidth on links, bandwidth allocated to flows, and hop by hop forwarding rules. This detail is sufficient to run a Dijkstra algorithm that sets up end-to-end paths and manages network usage. Abstractions at this level insulate algorithms from technology details such as the physical medium used and types of switches used.

Below that are yet a more detailed abstractions, which represent the lower-layer technologies deployed which incorporate node-level operational constraints. For example, in optical transport networks, these might include information about which lambdas are in use on a given lightpath. Abstractions at this level enable algorithms that determine when to light up a new lambda, instantiating new lower-layer paths to off-load higher-layer switching. Such abstractions also insulate users from the vendor specific details.

4.2 Rendering

Rendering is the function that maps between different levels of abstraction. Changes to any level of abstraction, in the form of CPA graph manipulations, may trigger rendering of the updated abstraction to the other abstraction levels. For example, adding port-to-port forwarding to a virtual layer 2 switch at the top level requires determining the actual path through the network topology one level down. The rendering function performs these actions (e.g., runs Dijkstra's algorithm to find a path with the required bandwidth) and modifies the lower-level abstractions accordingly. Rendering may also percolate up in cases of changes to the topology below. This occurs in instances of link failures.

Rendering is not only a formulaic translation between abstractions. Rendering can also invoke, track and enforce business logic for handling outages, rebalancing loads, lighting up new lambdas, instantiating new firewalls, and more depending on the network.

The lowest level of rendering is called network rendering. At this level, changes to the detailed abstractions are rendered to invoke changes in the actual network. In our implementation the network rendering engines use the OpenFlow protocol to configure switches so that they match the lowest layer abstractions. We also render configurations for linux gateway switches to reflect route manipulations.

4.3 Avatars and the Physical Network

Bringing the discussion back to the analogy of the network as an avatar, recall that in Section 3, CP-CP avatar interactions were supported by graph shadowing. In this section, CP-DP rendering was discussed where actions upon abstractions trigger actions or changes in configurations in the physical network.

The actions on abstractions may be caused by the local domain's CPA or by an action shadowed from other CPAs. This supports the notion of rendering avatar actions down to the physical network regardless of which actor caused the action (provided that there is permission for the action). Rendering can be thought of as a way to maintain multi-layer consistent state within a network's avatar.

Abstraction, rendering between abstraction levels, and shadowing allow multiple CPAs to collaborate, allow OpenFlow and other CP-DP mechanisms to easily hook into the network graph representation and enable a full suite of CP-CP, CP-DP, and DP-CP behaviors.

5. EXAMPLE IMPLEMENTATION

We have implemented a prototype Distributed Resource Controller (DRC) based on the concepts presented in Sections 3 and 4. Our implementation includes a graph database for storing and manipulating network abstractions while listening for changes that may require triggering, rendering, and shadowing. We selected the Dex graph database [15], on top of which we developed a C++ API implementing a subset of the Gremlin graph traversal language [12].

To investigate the issues associated with CP-CP coordination and rendering, we used DRC to implement a multi-domain virtual private network (VPN) service over multi-protocol label switching (MPLS). The multi-domain VPN service consists of two domains: a multi-site corporate network (the customer) and a provider network that will be used to connect the multiple sites of the customer as shown in Figure 3a.

Initially, the customer and provider each have a DRC, containing private graphs that represent their own networks. They then exchange a subgraph that represents their points-of-presence (nodes that are connected to each other's networks) as illustrated in Figure 3b. They configure this shared subgraph so that edges added between these points-of-presence will automatically be shadowed as part of the subgraph.

In this example, the customer decides that it wants direct links between sites 1 and 2 and between sites 1 and 3. When the customer instantiates this desired VPN topology (Figure 3c), the new edges are shadowed back to the provider, including link attributes to specify the desired site-to-site bandwidth. In the case of a layer-3 VPN, the customer can specify inter-site IP routes to enable appropriate IP forwarding across virtual links.

The provider has set triggers so that these new edges invoke a Dijkstra algorithm to compute paths to support the requested virtual links (Figure 3d). Then, additional triggers provision MPLS tunnels connecting the customer sites by invoking network rendering. If the customer changes its VPN topology so that the hub is located at site 2, the provider's VPN provisioning code ensures that a batch transaction is created which instantiates the new MPLS tunnels before tearing down the old tunnels.

The approach described in this section is an example of how to use DRC to build and manage a multi-site VPN. Alternative uses supported by the DRC shadowing infrastructure include cases where:

1. The provider shares a subgraph containing all possible links between customer points-of-presence. A customer can set a boolean attribute on each link to choose which ones to instantiate.

2. The customer sets a numberic attribute on each link to specify a desired bandwidth. In response, the provider sets an attribute that specifies the associated cost; finally the customer sets a boolean attribute to select that link.

3. The customer uses his graph model of the multi-site enterprise network, including the inter-site link SLAs, for network planning and performance analysis of the aggregate multi-site network.

6. APPLICATION SCENARIOS

The shadowed subgraph approach can also be used to address a wide range of other problems:

- **Multi-Layer Transport Networks.** Jointly managing the multiple layers present in optical networks provides opportunities for energy savings and capital expenditure (CAPEX) reduction. This requires using graphs to represent not only the packet network but also the Level 0 fiber graph that indicates physical spans of fiber bundles between switching points and the capabilities of each switching point.

  Optical switches capable of Dense Wave Division Multiplexing configure can be configured to support multi-hop optical light-paths, creating a more abstracted graph of longer Level 1 spans that have no electronic. Within each light-path or "lambda", ODU time slots are allocated along multiple Level 1 spans in series to provide end-to-end packet transport at a selectable bandwidth, and in parallel in order to establish protection paths.

  Each of these control-plane representations of the optical network is easily encapsulated in a property graph. Building on core techniques employed for PHAROS [1] and current developments in the Optical Internetworking Forum, each layer of abstraction in the optical hierarchy has its own vertex and edge types in an optical property graph. Triggers can propagate network events up the layers as well as a means of driving configuration events down the hierarchy towards the CP-DP messaging protocols.

- **Scalability and Robustness.** Large SDN networks require multiple controllers for scalability and robustness. Subgraphs can be used to tailor the scalability approach to the services being deployed. For example, controllers could use a subgraph that represents a geographic group of switches for services that typically use regional resources, or a subgraph could represents a slice of resources across the whole network, allowing a single controller to respond to service requests unilaterally. The resources associated with subgraphs can be coordinated using peer-to-peer or hierarchic algorithms.

  Our focus on handling scalable service instantiation uses a decomposition approach. Each DRC instance is given responsibility for a scope that does not overlap with that of any other DRC instance; as in our PHAROS program [1], a scope consists of a service context and a set of assigned resources. The service context defines the service requests for which this DRC instance will perform setup. Allocating non-overlapping resources to each DRC instance allows a high degree of optimization and highly consistent setup times, as a DRC instance can execute a global optimization algorithm that takes into account all resources and all service demands within its scope. There is no backtracking or thrashing and no risk of nonlinear increases in signaling traffic in times of high utilization. There is some risk of suboptimal resource decisions, but the architecture allows the reassignment of resources or service contexts.

- **Infrastructure as a Service.** There is significant interest in providing customers with a slice of a provider's infrastructure (network, data center, software, etc.). Subgraphs allow the provider to partition resource at any level of abstraction and to give fine-grained control of those resources to the customer who can then further subdivide them if desired.

  Delegating resources is not only useful for scalability but is also key to rapid end-user driven service creation and data-driven middleware deployment. Consider a scenario in which an end-user wishes to send data to multiple locations using a multicast spanning tree. The DRC can, in response to a user's request, delegate the virtual layer 2 switch to the end user. The end user can then dynamically configure the virtual switch to send the traffic to whichever end points have enough storage.

  Shadowed subgraphs can be subdivided and delegated further. Also, delegation can be performed at any layer. For example, a provider could delegate a subset of a virtual topology. In this case, the changes the delagatee makes are propagated to the provider's DRC, which then renders them down the abstraction stack until the network rendering engines effect changes in the network. Alternatively, the provider could delegate a subset of the topology abstraction. This could be useful when the delagatee needs to know the underlying topology. Again, changes made in the shadow abstraction are pushed to the provider's DRC, which renders down the abstraction stack to the physical network.

7. CONCLUDING REMARKS

SDN has delivered on its early promise to decouple the data-plane (DP) from the control-plane (CP) in computer networks. The proposal of OpenFlow as an example method of communication between CP devices and DP devices has served SDN well in achieving its vision thus far. In this paper, we proposed a method to open CP-to-CP communication just as OpenFlow has opened CP-to-DP communication. Specifically, we proposed a format for representing arbitrary network topologies, link states, policies, metrics, as well as associated meta data as property graphs. We also proposed a messaging scheme for sharing selected subgraph and property data among CP devices without loss of privacy, and ensuring eventual consistency through shadowing. Finally, we introduced the notion of a network avatar, which, when manipulated by local or remote CP actors (CPAs), renders the network configuration changes. We have implemented these concepts in a multi-domain MPLS VPN and discussed the viablility of shadowing in the realm of SDN. This implementation was the subject of a live demonstration on the GENI network testbed at GEC 17 [2].

In the future, we will extend our approach to address the problems associated with failover and robustness in a distributed graph database environment. Also, we will implement multi-layer optimization algorithms for selecting disjoint paths, which can be used to maximize flow rate or serve as backup paths.

The code for DRC is available under a BSD license by contacting Chris Kappler.

8. ACKNOWLEDGEMENTS

Thanks to Kristin Rauschenbach, Steven Dabideen, Regina Rosales-Hain and Shinya Ishida for discussions and implementation.

This work was partially supported by the Department of Energy (DoE) via Grant No. DE-SC0007340. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DoE or the U.S. Government.

9. REFERENCES

[1] I. Baldine, A. Jackson, J. Jacob, W. Leland, J. Lowry, W. Miliken, P. Pal, R. Ramanathan, K. Rauschenbach, C. Santivanez, and D. Wood. *Next-Generation Internet Architectures and Protocols*, chapter PHAROS: An Architecture for Next-Generation Core Optical Networks, pages 154–179. Cambridge University Press, 2011.

[2] GEC17 demo: Distributed resource controller. `https://www.youtube.com/watch?v=R0Xz4_3CEzA`. Accessed 10/2013.

[3] S. Das, G. Parulkar, and N. McKeown. Why openflow/sdn can succeed where gmpls failed. In *European Conference and Exhibition on Optical Communication*. Optical Society of America, 2012.

[4] A. Farrel, J.-P. Vasseur, and J. Ash. A path computation element (pce)-based architecture. Technical report, RFC 4655, August, 2006.

[5] P. Godfrey, I. Ganichev, S. Shenker, and I. Stoica. Pathlet routing. *ACM SIGCOMM Computer Communication Review*, 39(4):111–122, 2009.

[6] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. Nox: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

[7] D. Gupta, A. Segal, A. Panda, G. Segev, M. Schapira, J. Feigenbaum, J. Rexford, and S. Shenker. A new approach to interdomain routing based on secure multi-party computation. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pages 37–42. ACM, 2012.

[8] H. Kim and N. Feamster. Improving network management with software defined networking. *Communications Magazine, IEEE*, 51(2), 2013.

[9] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: a distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[10] V. Kotronis, X. Dimitropoulos, and B. Ager. Outsourcing the routing control logic: better internet routing based on sdn principles. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pages 55–60. ACM, 2012.

[11] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker. Software-defined internet architecture: decoupling architecture from infrastructure. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pages 43–48. ACM, 2012.

[12] M. Rodriguez. Gremlin graph traversal language. `https://code.google.com/p/orient/wiki/Gremlin`. Accessed 6/2013.

[13] S. Shenker, M. Casado, T. Koponen, and N. McKeown. The future of networking, and the past of protocols. *Open Networking Summit*, 2011.

[14] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Flowvisor: A network virtualization layer. *OpenFlow Switch Consortium, Tech. Rep*, 2009.

[15] Sparsity technologies dex. `http://www.sparsity-technologies.com/dex.php`. Accessed 6/2013.