The eXtensible Session Protocol: A Protocol for Future Internet Architectures

Ezra Kissel
Department of Computer & Information Sciences
University of Delaware, Newark, DE 19716
kissel@cis.udel.edu

Martin Swany
School of Informatics and Computing
Indiana University, Bloomington, IN 47405
swany@iu.edu

Abstract

Managing modern heterogeneous network technologies in a simple, uniform manner has become an increasingly difficult challenge. To help address this issue, we propose a session layer protocol called the eXtensible Session Protocol (XSP) designed to integrate existing network systems, providing the ability to easily introduce additional protocol functionality as needed, including application-driven network allocation and integration with network "middleboxes." The XSP implementation is currently targeted at network middleware architectures where it allows for the transparent integration and configuration of new and existing network components and services

1 Introduction

This paper presents the eXtensible Session Protocol, XSP. The goal of XSP is to provide a general and extensible protocol to manage the interaction between applications and network-based services, and among the devices that provide those services. A session, in our model, is generically "a period of a particular activity." A survey of existing technologies for managing data movement over current and emerging network infrastructures shows that a unified or generic solution for interacting with network services has not emerged. The challenge increases when we consider the heterogeneity of network architectures, transport layer services and application interfaces in the network ecosystem.

While there is no unified approach to session layer functionality in the current Internet, there are many instances of functionality that is consistent with a session layer. XSP is conceptually related to the ITU-T Recommendation X.225 connection-oriented session protocol specification [22], which defined *" a single protocol for the transfer of data and control information from one session entity to a peer session entity...between systems which support the session layer of the OSI reference model."*

The essence of this approach is simply that the session layer lives above the transport layer, and provides protocol encapsulation for both control and data protocol units. This model then subsumes cases like SIP, where the protocol control information is exchanged out of band, with the data encapsulation being essentially null at the session layer and using e.g. RDP at the transport layer. It also expresses the DTN "bundling" approach by including both control and data parts in the PDU.

The focus of this paper is on two additional use models. The first is on the use of performance oriented "middleboxes", which in the session paradigm are simply gateways. These middleboxes and can act as "accelerators" and mitigate performance problems over long distance, high speed networks. The second use of XSP manages dynamic allocation of network resources including, e.g. session driven creation of LSPs through MPLS networks, VLANs transported over Ethernet, SONET, etc., or paths constructed with explicit rules being inserted in OpenFlow [9] enabled switches.

Lastly, we assert that XSP enables many of the capabilities that are emerging as desirable for future Internet architectures. Initiating a connection to a session entity, and abstracting a direct transport layer connection away from applications, makes it straightforward to connect two session endpoints based on an "identifier", which is distinct from a network "location." This has clear benefits, not the least of which is mobility. The potential to exchange protocol control information with an intermediate point as part of a session connection to an endpoint, changes the security model as well. We have developed a framework that allows a host to authenticate with an X.509 certificate and effectively install firewall rules that allow particular transport layer connections to flow through for the life of the session.

The outline of this paper is as follows. Sections 2 and 3 discusss background, motivation, and related work for our XSP approach. We present details about the XSP session layer and protocol implementation in Section 4. In Section 5 we present current XSP use models and we discuss the current and future state of XSP in Section 6. We con-

clude the paper in Section 7.

2 Background

Historically, the role, architecture, and necessity of the session layer have been questioned. The upper layers of the OSI reference model [18] were conceived in the late 1970s in an attempt to quickly standardize a reasonable working model for emerging protocol designs [17]. This process was mired with disagreement and pitted the established telecom service providers against the fledgling computer industry, leading to a murky collection of functionality defining the *Session*, *Presentation*, and *Application* layers. What arose from this debate were session layer capabilities that dealt with various dialog control and synchronization primitives, which were strongly opposed by those in the ARPANET camp. These primitives served to establish a specific set of protocol design patterns in various telecommunication arenas but this strict layering of functionality was charged with complicating, and at worst, stymying the development of future applications within emerging computer networks. In the end, a consensus emerged that acknowledged that even if these functions were not in the *best* place, they were close enough for practical, engineering purposes and could be sorted out as the upper layers became better understood. This debate, of course, was never fully resolved. In many ways, this lack of resolution has led to some of the challenges facing the Internet today, while upper layer protocol considerations as originally defined in the OSI model have received little attention in most network application designs.

It goes without saying that the TCP/IP, or "Internet" model [11], is the dominant description framework for computer network protocols. Given that protocols are not rigidly designated into strict upper layers within this model, what was originally defined as session layer functionality is often explicitly implemented within applications themselves. In fact, it has been observed that *"the session functions are actually common modules for building mechanisms used by applications that should be considered, in essence, libraries for the application layer"* [17]. To complicate things, the OSI session layer defines functionality that corresponds to features implemented within the transport layer in the TCP/IP model, e.g. port numbering and the associated management of connections between services on well-known ports. This conflating of functionality has left designers of emerging networks and protocols in a difficult position, one in which exploring any significant funtional change within the TCP/IP model has serious deployment consequences.

We are also not alone in realizing that the current Internet model has a number of challenges to overcome. As devices become more mobile, intermittent, and increasingly heterogeneous, the well-established protocols within the transport layer in particular have created a number of roadblocks in supporting emerging network architectures. Projects such as MobilityFirst [5] are developing collections of services for supporting the explosion of mobile devices on the Internet, which taken together hope to address new modes of routing, naming, and in-network storage in future real-world and experimental architectures. Others have proposed breaking up the transport layer to better manage addressing, congestion, and flow control through the introduction of additional *Flow Regulation* and *Endpoint* layers [23]. Although these motivations are all sound, we believe that significant architectural changes can be best realized through the establishment of a dedicated session layer that encompasses control and data plane management functionality without seriously disrupting the operation of existing transport layer features. Our work with XSP addresses both the control and addressing of transport layer connections while providing a framework for incremental deployment with existing implementations.

The session layer itself has been experiencing renewed interest in recent years. Particularly in the realm of mobile and delay-tolerant networks, the session layer can encapsulate a number of capabilities for managing transport re-binding, device naming, and connection recovery and checkpointing [19, 31]. As initiatives such as FIND [7] explore the future direction of the Internet, many consider these features to be desirable. Indeed, as networks become increasingly dynamic and network topologies, connectivity models, and usage patterns become more varied, they may be essential. Traditional end-to-end arguments are also being re-evaluated as these architectural changes drive new modes of operation [13]. We believe that XSP is in a position to address not only the needs of the current Internet, but those of evolving future internet architectures.

3 Related Work

While there has been relatively little published work in protocol development at layer 5, there have been numerous middlebox, overlay, and control plane protocols proposed over the years that share common aspects with XSP. We now briefly survey related work that has influenced our own design criteria and choices.

Perhaps the most widely known protocol providing session layer functionality is the Session Initiation Protocol (SIP) [41]. Its primary use is in enabling state and signaling for multimedia communication sessions, which may utilize a number of underlying data stream protocols such as RTP and RTSP. Even so, some approaches such as

NUTTS [29] have adapted SIP to provide "name-routing" in an end-middle-end middlebox architecture. A drawback of SIP from the perspective of our work is that it only performs signaling out-of-band and does not provide an integral data movement service, limiting its applicability as a general purpose framework for end-to-end services. On the other end of the spectrum, TESLA [43] provides an end-to-end session architecture for the management of data flows, routing, and migration without considering independent configuration and management of network devices and path services.

In addition to SIP, a number of other protocols relate directly to XSP. We have investigated integrating relevant features of SCTP [44] such as multipathing and the bundling of SCTP associations as potential elements for a session-based signaling and data movement service. The Delay Tolerant Networking Research Group (DTNRG) [16] and its associated Bundle Protocol [49] attempts to solve the problem of message delivery and routing in challenging network environments, including very large delay transmission and frequently disconnected network paths. Recently, a session layer for DTN [19] has been proposed that will allow receiver-driven applications to manage relationships between individual "bundles". Mapping multiple transport streams as in SCTP, and the ordering and timing of messages within a stream as in DTN, are directly addressed by XSP in a single framework.

There have been a number of proposed middlebox communication frameworks whose functionality may be generalized with XSP. We share a common motivation with the inter-middlebox architecure MIDCOM [46], designed to embed application intelligence into a network of MIDCOM agents. Using a standard inter-agent protocol, MIDCOM focused on firewall and NAT services as well as the management of bundled session applications like VoIP. Unfortunately, MIDCOM never saw deployment in the real world and has since developed in other directions. More recently, ForCES [20] proposes to provide a framework for logically separating control and data planes within network processor devices. It primarily focuses on integrated systems such routers and other network appliances while defining the protocol for communication between distinct control and forwarding elements. The scope of ForCES is tied to the interaction of closely coupled network elements whereas XSP aims to provide a more general approach to signaling across applications and devices from an end-to-end perspective. We envision that XSP may interface with a ForCES control element for managing that particular class of network device.

The Next Steps in Signaling (NSIS) [8] working group proposes a two-layer IP signaling standard focusing on QoS signaling and a reuse of existing reservation protocols such as RSVP. Similarly to FoRCES, NSIS is targeted at a specific set of use cases involving network device configuration that we believe fall under our XSP signaling model.

In addition, our approach utilizes some guidelines from BEEP [40], a framework for developing network application protocols.

4 The XSP Session Layer

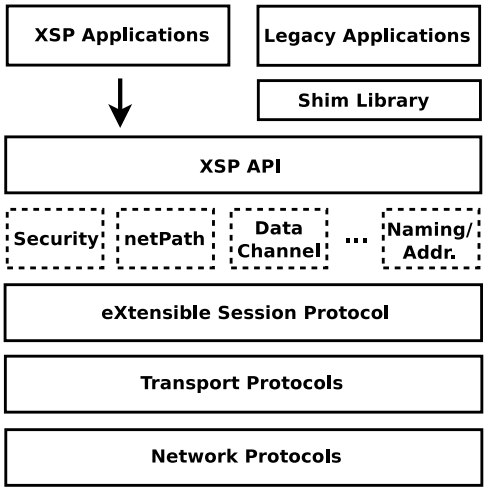

Figure 1: XSP stack architecture

A key insight in our work is that an articulated, session-based network path can provide a number of benefits for managing connections, in terms of control and data messaging, as well as improving the performance of the configured data channel when appropriate. In general, the management a number of separate transport-specific features allows XSP to take advantage of an end-to-end path that requires the configuration and traversal of path segments that have unique characteristics. Thus, a primary goal of XSP is to provide a core set of functionality and associated signaling required for network services and applications to exchange and negotiate capabilities in a common and easily extensible framework.

The XSP session layer is designed as a collection of modular session layer service handlers, or XSP-SH, that are accessed through a common API. Providing a standard interface to the network, each XSP-SH uses the underlying XSP protocol to communicate between session-enabled endpoints. Any and all session functionality defined by XSP resides above the transport layer, keeping the existing lower layers of the Internet model intact. Our XSP network architecture is illustrated in Figure 1, with the XSP-SH layer indicated by dotted boxes.

Each XSP-SH provides a framework for implementing a specific instance of that service for the application. For example, the *Security* XSP-SH makes avail-

able a number of different authentication methods that can be requested, including *anonymous*, *username/password*, *X.509/SSL*, *SSH*, and so on. The XSP-SH layer is implemented as a set of modular libraries that can be dynamically loaded when requested by the application. Additional modules can be added by simply implementing another handler within the desired XSP-SH framework. We show a subset of the available framework components in the XSP-SH layer. *netPath* provides a number of modules for configuring network devices, services that can dynamically provision network paths, and devices that speak a common protocol such as OpenFlow [9] or NETCONF [21]. The use of various control and data channels is supported with the *Protocol Channel* XSP-SH, and *Naming/Addressing* allows for the integration of external endpoint location and identification systems.

One of the challenges for the XSP-SH abstraction is defining the core set of features that the session layer should provide. While the current XSP-SH layer enables applications to use what we consider a common set of built-in features (i.e. the current state of the art), the true advantage is realized with the extensibility of the framework for supporting additional capabilities as future network architectures evolve and take shape, all within a consistent interface. We take a closer look at how this applies to the *Protocol Channel* and *netPath* XSP-SH frameworks in our two XSP use models described in Section 5.

4.1 eXtensible Session Protocol

To support the architecture described above, the XSP protocol implementation must provide for the negotiation, establishment, message exchange, and termination of a session between application processes and any intermediate devices. This negotiation involves a set of requests and responses that are transmitted over the network as session-layer PDUs (SPDUs) between network end-points that speak a session layer protocol implementation. Our binary XSP protocol specification is designed to provide a general mechanism for this exchange of protocol control information (PCI) and application data.

The basic XSP PDU structure consists of a fixed-length message header defining the following fields: version, flags, message type, option count, 128 bit source and destination endpoint identifiers (EIDs), a 128 bit session identifier, and 16 bits of reserved space. XSP EIDs are a complex type that are able to represent a number of address bitfields as well as human readable names (HRNs). Table 1 briefly describes each field within the XSP PDU.

In addition to the common XSP header, there is a variable size option block used for the transmission of XSP control messages and application data. The option block header contains fields for the option type, service port, and length, which define the properties of the attached option data. The length of the option block is indicated by the first 16 bits of the option length field, which is size-prefixed in order to not only efficiently send small ($< 2^{16}$ byte) messages, but also much larger application data PDUs. If set to 0xFFFF, the next protocol word is a 64-bit unsigned integer length in network byte order that represents the length of the immediately following option data. The XSP PDU structure is shown in Figure 2. Similar to IPv6, the protocol word width is 64-bits to allow optimizations on 64-bit CPUs. Since every 64-bit boundary is also a 32-bit boundary, 32-bit CPUs are not negatively affected by this optimization.

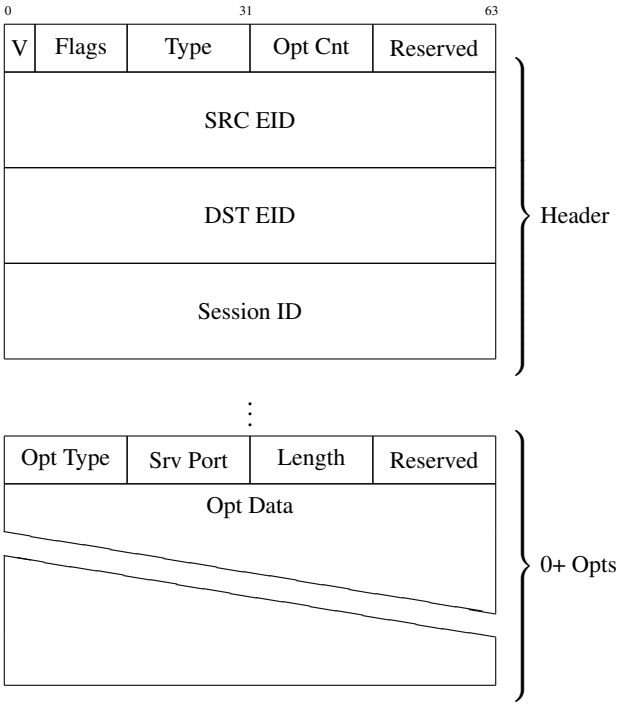

Figure 2: *The XSP PDU*

Header Field	bits	Description
V	4	Protocol version (e.g. 1)
Flags	12	Message flags for the message. (e.g. priority)
Type	16	Specifies the type of XSP message. May indicate expected option block types.
Option Count	16	Specifies the number of option blocks associated with the XSP message.
SRC EID	128	The source identifier for this message. (e.g. IPv4, IPv6, HRN)
DST EID	128	The destination identifier for this message.
Session ID	128	The session identifier for this message, i.e. a random hex string.
Option Type	16	Specifies the option type for the given option block.
Service Port	16	Specifies an optional "session port" associated with the option data.
Length	16	Length of option data. 0xFFFF for data $> 2^{16} - 1$ bytes.

Table 1: *Description of XSP PDU fields*

During the lifetime of an XSP session, messages consisting of the basic XSP header and zero or more option

blocks are exchanged over the XSP session context. These option blocks may define requested transport layer capabilities, articulate an end-to-end path, exchange application data, or configure any other XSP-SH features currently supported along each network segment. This exchange takes place as a negotiation between XSP-enabled network elements (NEs) spanning the endpoints defined by the established session. A number of XSP primitives provide this protocol functionality, which correspond to XSP message types indicated by the *Type* field within the XSP PDU. A subset of the core XSP primitives is listed in Table 2.

The core protocol implements these primitives and makes them available as part of the XSP-SH layer. For example, the *XSP_AUTH_TYPE* primitive is used by all handlers conforming to the *Security* XSP-SH. In contrast to the built-in types, an application or service developed using XSP registers its option block handlers with the XSP session layer, enabling the processing of option block types associated with the service via the *SESS_APP_DATA* primitive. An XSP-enabled NE receiving such a message iterates over the option blocks and parses only those options that an active application has registered to the XSP-SH layer. These option blocks are opaque to the XSP session and are simply made available to the application via the XSP API. Depending on local configuration and policy, the NE may forward all or none of the received option blocks to other Network Elements (NEs) associated with the session.

As the name implies, one of the key benefits of XSP is the extensible nature of the protocol. Functionality is easily extended by defining additional option block types to be communicated across XSP-enabled NEs and defining handlers for those option types. The option block service port field is used to associate a particular service or application with the XSP session implementation, allowing overlap between option types communicated along the same session instance. This is in many ways analagous to the IANA service port number assignments in the Internet model, but is extended in XSP to apply across potentially many endpoints within the active session.

Once XSP messages are received off the wire, an NE may accept and process a given option type or otherwise ignore and forward the option block to the next hop, if any. This method of selective option processing provides a distinction between "hop-scoped" and "session-scoped" option types. A hop-scoped option may be processed by only interested intermediate NEs while a session-scoped options may be forwarded to the session endpoint without additional overhead. This distinction is useful when an end-to-end session may include a number of NEs and session control PDUs are signaled along the data forwarding path, but the message is only parsed at those NEs that are expecting a particular option type.

Finally, it is worth noting that the XSP protocol abstraction allows for XSP sessions to be established in a transport-agnostic fashion, meaning an active session may be maintained over any number of available transport protocols available via the *Protocol Channel* XSP-SH. In practice, and perhaps not surprisingly, XSP control sessions are most often established over TCP connections given their reliability and ubiquity in today's networks.

4.2 Application Support

The core XSP library is known as *libxsp*. Building an XSP-enabled application involves linking against *libxsp* to access the main protocol functionality, XSP-SH framework, and configurable connection support for establishing new sessions. An *Application* XSP-SH provides hooks for supporting application messaging via external modules, as described above. These application-defined handlers implement the desired message format and serialization/deserializaition methods needed to process the message before being encoded/decoded by the XSP protocol implementation. This approach allows XSP to communicate application-specific information in a general manner while simultaneously maintaining session state between a number of associated NEs.

The current XSP implementation includes a Sockets API-compatible client library, *libxsp_client*, that is being retooled to expand feature support. This provides familiar semantics such as *open*, *close*, *connect*, *send*, *recv* and extends these with session-specific calls that interact with the XSP-SH layer. Thus, an application may establish a session for a reliable, stream-oriented connection, as would be provided by TCP, while being able to configure a desired authorization scheme, dynamic network path, separate data channels, etc., all from a common interface.

Our XSP implementation also provides a transparent wrapper, known as the *Shim Library* indicated in Figure 1. Using library interposition (e.g. via the Linux LD_PRELOAD mechanism), the wrapper allows existing applications to take advantage of XSP without requiring any source code modifications.

5 Use Models

In conjunction with the development of XSP, we have applied our session layer architecture in addressing specific issues within today's Internet. We now focus on two use models that integrate XSP in order to improve wide-area network performance and provide a standard interface for dynamic network configuration.

Primitive	Description
SESS_OPEN	Establish a new XSP session. Session NE hops may be specified in option blocks.
SESS_CLOSE	Close an existing XSP session.
SESS_ACK	Acknowledge a received XSP message. May contain additional information within an option block.
SESS_NACK	Negatively acknowledge a received XSP message. May contain an error indication within an option block.
SESS_DATA_OPEN	Open a new data connection bound to the current session.
SESS_DATA_CLOSE	Close a data connection bound to the current session.
SESS_DATA_CHK	Checkpoint or synchronize data connection.
SESS_APP_DATA	Send application-specific data within registered option blocks.
SESS_AUTH_TYPE	Send and negotiate authentication type.
SESS_NET_PATH	Configure a network path defined as a common set of rules.

Table 2: A subset of XSP primitives

5.1 Rethinking Bulk Data Movement

Achieving reliable, high-speed data transfer performance remains a "holy grail" for many in the research and education (R&E) and e-Science communities, and it is increasingly important for the commercial sector as well. While available link and backbone capacity have rapidly increased, the achievable throughput for typical end-to-end applications has failed to increase commensurately. In many cases, application throughput may be significantly less than what is theoretically achievable unless a considerable amount of effort is spent on host, application, and network "tuning" by users and network administrators alike. The growing WAN acceleration industry underscores this need.

Our earlier work with Phoebus [33, 34] is a direct response to this performance gap, particularly when bulk data movement is concerned. Phoebus is a middleware system that applies our XSP session layer, along with associated forwarding infrastructure, for improving throughput in today's networks. Phoebus is a descendant of the Logistical Session Layer (LSL) [47,48], which used a similar protocol and approach. Using XSP, Phoebus is able to explicitly mitigate the heterogeneity in network environments by breaking the end-to-end connection into a series of connections, each spanning a different network segment. In this model, Phoebus Gateways (PGs) located at strategic locations in the network take responsibility for forwarding users' data to the next PG in the path, or to the destination host. The Phoebus network "inlay" of intelligent gateways allows data transfers to be adapted at application run time, based on available network resources and conditions.

In order to adapt between protocols along different network segments, Phoebus uses the *Protocol Channel* XSP-SH to implement a number of transfer backends. These backends can then be used interchangeably via the shared XSP API while the underlying XSP framework handles any differences in protocol semantics. The existing Phoebus implementation has developed and experimented with a number of protocol backends, including TCP, UDP, and MX [25], as well as userspace protocol implementations such as UDT [28].

Building upon our experiences with Phoebus, we developed a modular extension to the Phoebus architecture called Session Layer Burst Switching, or SLaBS [32]. SLaBS uses the XSP session layer to enable an intelligent store-and-forward data movement service that takes advantage of large buffers at PG adaptation points to form data bursts and optimizes their transmission over dedicated network resources. Appropriately enough, we call these bursts "slabs" and the process of forming slabs "slabbing". In essence, slabs are SPDUs that are formed by coalescing smaller SPDUs from the edge (e.g. user application flows) that have been terminated at PGs. Incoming SPDUs are multiplexed, or reframed, into larger SPDUs (slabs) which are more suitable for high-performance transmission over wide-area networks using the protocol adaptations available with Phoebus. The XSP session layer provides the mechanism for exchanging slab information over a SLaBS control session and the multiplexing/demultiplexing of SPDUs between SLaBS gateways. Additional details are available in our previous work [32, 34].

Up to now, all of the data movement within the Phoebus and SLaBS approaches has used a synchronous, *"send when available"* approach. We now describe an extension to SLaBS that takes advantage of an asynchronous, *"come and get it when ready"* model for data transfers supported by XSP signaling.

5.1.1 Bulk Asynchronous GET

As networks continue to get faster, a key performance consideration is not only the efficiency of the transport protocol but also the level of operating system (OS) involvement in supporting data movement between end-hosts. Traditionally, a transfer application will read data from the local resource (e.g. disk) and invoke the OS to transfer that data with TCP (or UDP with some user-level protocol implementation.) As the transfer proceeds, the OS is tasked with meeting the application requests for copying data into its user-space buffer while simultaneously, and synchronously, sending data over the network and acknowledging receipt of PDUs from the transport layer. With 10Gb/s network interfaces becoming more common in high-performance computing environments, we are currently in a situation where an application strug-

gles to achieve commensurate throughput without significantly taxing the CPU, or at least some siginificant fraction of available cores. Although there are a number of optimizations available, from kernel socket splicing to TCP offload engines, this situation is not sustainable for applications interested in moving bulk data sets over networks at 10Gb/s and beyond.

One alternative is to decouple the data movement over the network from the involvement of the operating system itself. In this model, sending a resource, whether it be a set of files or data already within the page cache, involves letting the OS simply stage the data in memory on behalf of the requesting application while allowing the remote host to asynchronously "get" the prepared memory regions via some transport mechanism. We call this particular type of transfer scenario Bulk Asynchronous GET, or BAG.

The BAG approach entails having a producer make some resources available for a remote consumer to access, during some particular window of activity, which is exactly what our earlier definition of a session allows via XSP. The BAG operation is asynchronous in that the producer is only notified of a transfer completion if requested or required by the implementation, and typically via an out-of-band message. An analogy can be drawn between BAG and that of shipping packages via UPS or FedEx. The sender is not, generally, continuously involved with the delivery of their items from one location to another. Instead, UPS is notified that some number of packages are available at a particular address for pickup, the sender makes them available at their front door, and UPS "gets" the packages and delivers them to the desired destination. A sender may even request delivery confirmation which can be received or checked "out-of-band" via a web site.

What BAG requires is the ability to decouple the sender (really the host operating system) from the task of pushing the actual data through the network, freeing up the system to perform other tasks, just as UPS frees a shipper of packages to go about their day. Fortunately, the rapidly evolving area of Remote Direct Memory Access (RDMA) technologies has provided exactly this capability.

The BAG approach draws inspiration from data center environments where switched-fabric interconnects like InfiniBand and similar RDMA protocols have played a significant role in enabling massive parallelization with improved throughput and reduced latency and overhead. Supporting *zero-copy* networking, RDMA operates on the principle of transferring data directly from the memory of one system to another, across a network, while bypassing the operating system and eliminating the need to copy data between user and kernel memory space. These direct memory operations are supported by enabling network adapters to register, or "pin", memory and directly access these explicitly allocated regions without involvement or context switching from the host operating system.

Recently, enhancements to Ethernet for "data center bridging" have led to RDMA implementations that run directly over existing layer-2 network infrastucture, using RDMA-enabled network adapters called rNICs. By allowing the network adapter to encapsulate memory-resident application data within layer-2 frames directly, the overhead of higher level protocols can be virtually eliminated. A number of RDMA over Ethernet (RoE) implementations are under development and a standard for high-performance Ethernet rNICs has been proposed and implemented within currently available hardware [3, 37].

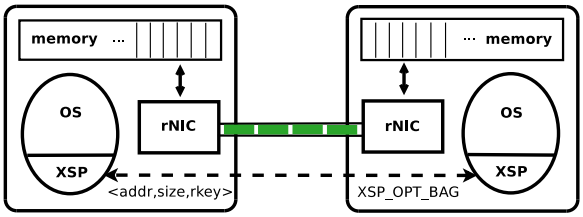

Figure 3: System-level view of RDMA transfers with XSP-driven Bulk Asynchronous GET (XSP-BAG)

Using XSP, we have developed a conceptual model of a BAG service that uses an RDMA transport and have begun implementing the necessary components within the XSP session layer. This has involved two main tasks: (i) creating an RoE *Protocol Channel* XSP-SH, and (ii) adding XSP option types that allows the application to exchange the necessary metadata to perform the remote GET operations and signal transfer completion events. The RDMA protocol handler in XSP-SH uses the OpenFabrics [12] *rdmacm* and Infiniband *ibverbs* libraries to establish the RDMA transport context and initiate the supported RDMA operations. The employed Infiniband "RDMA READ" operation is equivalent to the GET described in Bulk Asynchronous GET and requires the exchange of memory region pointers to move data from one RDMA-connected host to another. The XSP option blocks defined for BAG transfers (option type *XSP_OPT_BAG*) encode and exchange the necessary local and remote addresses, keys, and size of each memory region to transfer.

Figure 3 shows a system-level view of XSP providing the necessary signaling to maintain a BAG transfer. After the XSP session establishes the RDMA context, registers local and remote buffers, and exchanges pointers, the rNICs proceed to transfer data within the designated memory regions without further involvement from the OS. What is missing from this picture is the service that "stages" requested data in memory to be transferred. We now investigate our SLaBS data movement system as one such in-the-network service, and we envision a future dedicated host capability that can support the memory staging of local resources through the intelligent use

of the OS page table implementation.

5.1.2 SLaBS with XSP-BAG

Our first implementation of the BAG approach extends SLaBS with the ability to use RDMA over Ethernet to more efficiently transfer slabs across dedicated network paths. Here, the memory regions to GET are the slab SPDUs being buffered at the SLaBS gateway and the main challenge involves extending the threaded buffer model within SLaBS to support efficient BAG transfers over high-latency WAN paths.

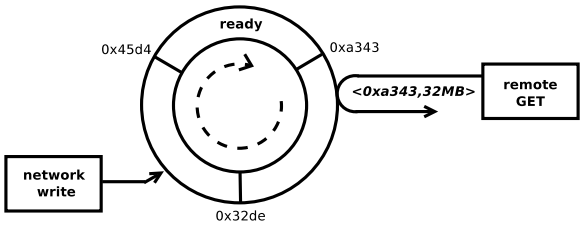

Figure 4: SLaBS "triple buffering" for XSP-BAG

As network latency increases, it is well understood that pipelining the transmission of network buffers is required in order to continually keep data "in-flight" within the network. TCP solves this with a sliding window protocol clocked to the round-trip time (RTT) of the path, the effects of which become exaggerated over so-called "long fat networks" leading to considerable performance issues. In contrast, SLaBS maintains an open loop model over the dedicated core network paths which allows us to determine ahead of time what resources are necessary and to pace slab transfers based on buffer and throughput capabilities at various points in the network. This amounts to ensuring that the buffering implementation has at least one bandwidth-delay product[1] (BDP) sized buffer in transit at any given time. However, in the BAG model with RDMA, there is an inherent trade-off between the number of registered memory regions and total buffer size required to saturate the given network path.

Our current SLaBS buffer implementation, illustrated in Figure 4, uses an adjustable size ring buffer with configurable memory region partitions within the overall buffer. To simplify the amount of memory region metadata exchange required with XSP, we employ a simple "triple buffering" scheme where three memory regions are exchanged between SLaBS gateways in a round-robin fashion to keep the network saturated. While incoming SPDUs are written to one region, the second region is a "ready-to-send"slab and its associated metadata has already been sent to the remote side within an XSP slab option block. The remote side posts the GET operations as they are received over the XSP control session while a third region is continuously being retrieved.

5.1.3 Performance Evaluation

This section presents our initial performance results as we evaluated the XSP-BAG additions to the SLaBS gateway system. All results were collected from a testbed environment consisting of 7 nodes connected with 10Gb/s Myricom Ethernet NICs, each node having at least two 10Gb/s interfaces. The testbed forms a linear network topology with client and server nodes at the edges, two SLaBS gateways in the middle, and 3 delay nodes segmenting the network into representative LAN and WAN segments. The edge host and netem nodes were Sun X2200 servers with quad-core AMD Opteron CPUs and 4GB of RAM, while the gateway systems contained AMD Phenom II X4 processors and included 8GB of high-speed DDR2 RAM. The gateways systems were additionally outfitted with two Mellanox rNICs in order to evaluate SLaBS performance with native, or "hard", RDMA over Ethernet. All of our transfer tests are memory-to-memory copies to avoid disk I/O bottlenecks., and unless otherwise noted, network traffic is generated using the *iperf* [2] bechmark.

Figure 5 shows that at 10Gb/s, the SLaBS gateway systems are not able to successfully form slab SPDUs and simultaneously burst buffered slabs over either TCP or UDP data channels. Increasing the number of additional incoming streams does not significantly affect the buffering or backend performance. With the XSP-BAG extensions and the RDMA data channel, the slab bursting performance nearly matches that of only writing SPDUs into the slab buffer from the edge connections. Indeed, the RDMA data channel transmits at the maximum bandwidth achievable by the rNIC, approximately 9.71Gb/s.

We also tested the popular file transfer tool, GridFTP [27], using both a direct TCP connection and when enabling SLaBS via the XSP wrapper library to transfer 128GB between GridFTP servers. As shown in Figure 6, the maximum achievable GridFTP transfer rate on our testbed systems is approximately 8.3Gb/s, fully utilizing a CPU core. [3] As the WAN latency is increased, direct TCP performance suffers whereas with SLaBS, and the RDMA data channel enabled, observable transfer performance remains consistent beyond 100ms, improving upon the direct TCP transfer by up to 18% in the highest latency case.

Figure 7 illustrates the importance of chosing the right sized SLaBS buffer to adequately maintain full utilization of the WAN path as latency increases. The buffer sizes indicated in the chart represents the total allocated buffer

[1] The bandwidth delay product of a network path is typically calcuated as $BDP = RTT(seconds) * rate(Bps)$

[2] Popular network measurement tool – see http://iperf.sourceforge.net

[3] Using the threaded GridFTP server and parallel streams at these rates only served to further decreases observable transfer performance.

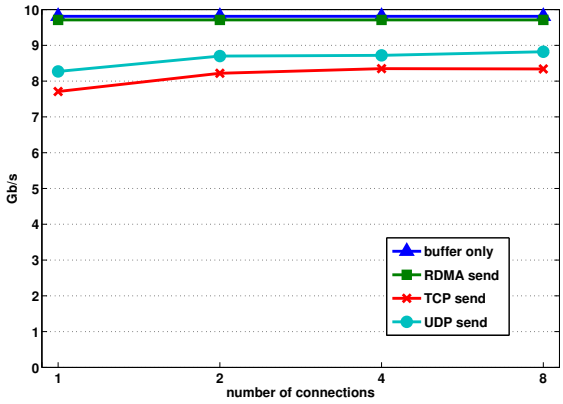

Figure 5: SLaBS data channel performance

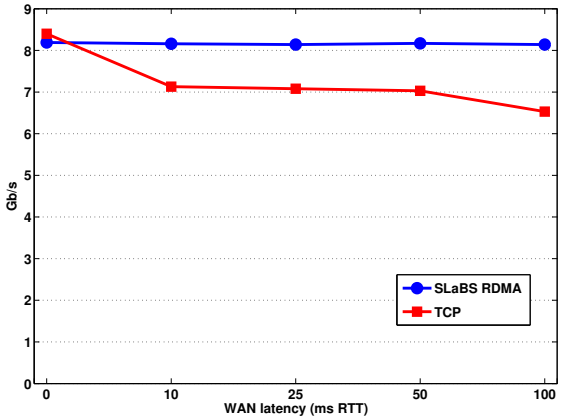

Figure 6: GridFTP transfers with increasing latency

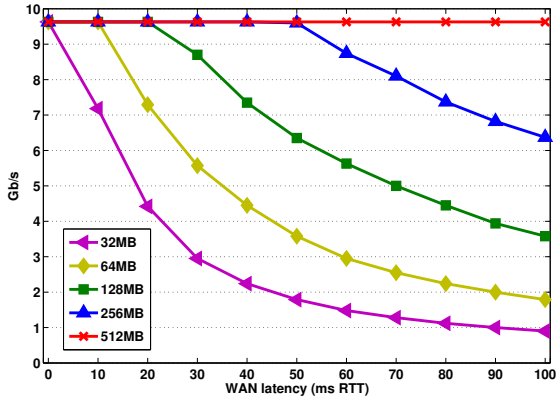

Figure 7: The effect of latency on buffer size

within the SLaBS gateway, 1/3 of which is in-flight via a remote GET at any given time. As that GET size approaches the BDP of the WAN path, performance of the RDMA data channel begins to decrease as the remote side has to wait for the next XSP slab record to pipeline, effectively "wasting" the available network capacity and increasing the signaling to GET ratio. With a 512MB buffer,

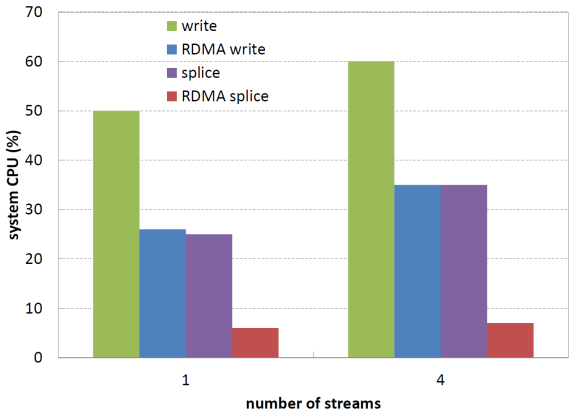

Figure 8: SLaBS CPU overhead

SlaBS is able to achieve full WAN network utilization well beyond 100ms, covering a large number of typical WAN path latencies while requiring a memory footprint achievable in most network service platforms.

Finally, we look at the benefits to system CPU utilization [4] in the gateway when using the XSP-BAG approach with SlaBS. Figure 8 compares TCP socket *write* calls, with and without RDMA, versus kernel socket splice optimizations, with and without RDMA, in the egress (remote GET side) SLaBS gateway. The *splice* and *vmsplice* syscalls use a kernel memory pipe to "splice" or connect file descriptors (may refer to sockets) and memory pages while avoiding costly userspace copying. Using only the splice calls instead of TCP write, we can achieve a nearly 40-50% reduction in total CPU usage; however, the transfer is still still bound by OS overhead. With the RDMA data channel and splice, the picture changes dramatically and we see only 6-7% total CPU utilization at nearly 10Gb/s transfer rates.

These results clearly show the benefit of the XSP-BAG model in improving performance of data movement service like SLaBS. Not only can we more effectively use dedicated network resources, but the total load on the system can be significantly reduced, allowing for other potential optimizations such as encryption or compression of the data stream.

5.2 Dynamic networks

The effective configuration and management of network resources is an issue of growing importance as networks become increasingly diverse and dynamically configured paths play a central role in a particular class of new applications and services. Indeed, the SLaBS and XSP-BAG

[4] We show total system utilization across cores. In our 4-core systems, this equates to 25% total CPU load respresenting one completely non-idle core, 50%: two non-idle cores, etc.

models just described depend on the ability of the network to provide dedicated and stable network resources to support high-performance data movement. Along with this diversity, devices in the network are also becoming smarter. Programmable router platforms [1,4] and OpenFlow [9] enabled switches allow new experimental protocols, services, and management techniques to be evaluated over existing network infrastructure.

The deployment of passive optical networks [35] (PONs) in bringing fiber to the home has spurred research into adopting such optical networks as a way to scale Internet infrastructure to 10-100G speeds and beyond [42,45]. Eliminating the need to convert optical into electrical signals allow PONs to reduce complexity and costs, and all-optical technology is finding its way into routers where it has the potential to dramatically increase Internet backbone capacity. As the name implies, PONs passively switch multiplexed waves over optical fiber and splitters, as opposed to lambda-switching or optical burst switching techniques [39] where active electronics are necessary to configure the lightpath either statically or just ahead of the data burst. With an appropriate control plane capability, data movement services like SLaBS are in a perfect position to utilize such high-performance paths where the deterministic path characteristics allow us to optimize the protocol and link usage.

In support of scientific computing and R&E communities, one approach to managing a specific class of backbone network resources has resulted in so-called Dynamic Network Environments (DNEs). These DNEs allow network resources to be provisioned "on the fly" by users, services and advanced applications. Demand-driven allocation of these ephemeral, dedicated links and paths enables unprecedented optimization of network utilization and is an ideal tool for demanding network applications. These "circuit networks" currently support high performance and Grid computing applications that must reliably and quickly move large quantities of data, and there are currently a number of existing DNE control plane technologies in general use. These include reservation and provisioning systems such as LambdaStation [14] and Terapaths [26]. ESnet and Internet2 have developed network-specific reservation systems, OSCARS [38] and DRAGON [36], to deploy on their respective circuit networks.

A considerable amount of effort is now being focused on the problem of local, or end-site network configuration. In many cases this involves extending a circuit provisioned via existing DNE systems into the local-area network of a particular institution where it can provide a dedicated, end-to-end virtual path for the requesting application or user. Efforts such as PWE3 [15] are exploring similar edge-to-edge capabilities over MPLS paths. Given the diversity of available devices within the typical end-site, standardization efforts such as NETCONF [21] are becoming increasingly important in providing a consistent interface to install, manipulate, and delete configurations within these environments.

Taking a step back, what users and applications fundamentaly require is a general mechanism to create, modify, and remove dynamic paths based on a shared global view of available network resources. An immediate use model for XSP is the ability to provide such a common and consistent interface for existing control plane architectures and systems. By allowing network configuration to be bound and managed by a session of activity, XSP allows applications to transparently configure and utilize a wide range of provisionable networks with minimal effort.

5.2.1 XSP netPath Model

Since our early work with Phoebus and its ability to act as an application "on-ramp" for DNEs, we have continued to extend the XSP framework to allow for more general network configuration. The *netPath* XSP-SH now enables the extensible and interchangeable use of an increasingly growing subset of the above technologes. To date, we have implemented handlers for OSCARS, Terapaths, and OpenFlow and have begun implementations for NETCONF and end-host network configuration for Linux-based systems.

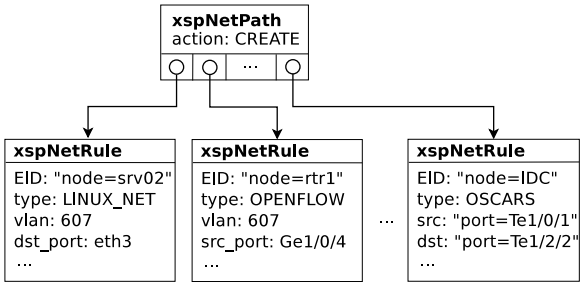

Figure 9: An xspNetPath structure contains a list of "rules" to be configured by the indicated *netPath* XSP-SH modules

In order to provide a common interface for each handler, the *netPath* framework abstracts a set of configuration parameters, or "network rules", that are needed to configure the underlying network device. Each *netPath* XSP-SH implementation uses this common abstraction to install a given network rule as determined by the specified XSP-SH handler type. By realizing that a large majority of network configuration tasks involves only a few core set of operations, we have been able to create rules that can be expressed with a relatively short list of parameters. In this we share a common theme with OpenFlow and their n-tuple for defining all configurable flows in an OpenFLow capable switch. Configuring dynamic

network paths in particular is typically a process of bringing up the appropriate interfaces, installing a number of forwarding rules, and applying a set of properties such as QoS bits and VLAN tags, or capacity and duration constraints in the case of DNE reservations.

The *netPath* interface exported via the XSP API presents an **xspNetPath** structure to the application which can specify a set of dynamic path actions such as CREATE, MODIFY, DELETE, QUERY, etc. This structure can be manipulated to contain a number of **xspNetRule** entries that, taken together, defines an end-to-end path, or it may simply contain a single rule to effect some particular configuration on a specific device. Figure 9 illustrates this structure.

The xspNetPath structures are encoded within XSP option blocks and sent along the active session as *SESS_NET_PATH* messages. An XSP NE configured with the *netPath* XSP-SH indicated within the xspNetRule entries load the appropriate handler and applies the specified configuration. An XSP EID within the xspNetRule entries allow the *netPath* handlers to identify and locate the particular network device or service to configure. The underlying handler interacts with the active session to provide acknowledgements if the operation completed successfully, or descriptive NACKs if it was unable to apply some or all of the device configuration.

5.2.2 Describing Dynamic Networks

A key requirement in the effective use of dynamic networks is the ability to accurately and completely describe the network topology and the services contained within. This information should include not only a list of services but should also detail the attributes of attached devices, the characteristics of network links, and the ability to find measurements that provide a picture of the current state of the network. Services should also be able to register information about what features and capabilities they provide in order to be discovered and utilized. Collecting and publishing measurement, service, and topology data on a large scale has been the focus of the perfSONAR [30] effort and is one that has taken an important role in the operation of DNEs as described above.

One of the major challenges born out of perfSONAR-related development has been the unification of service and network topology information in a common framework and schema representation. Representing complete topology information in a generic way is made difficult since detailed, cross-layer connectivity is often desirable or even necessary in describing dynamically changing device configurations and when attempting pathfinding across such networks. The task is further complicated when dealing with network features such as link aggregation and protocol encapsulation, and translation between network segments with potentially different VLAN tags and QinQ encoding. Driven by these requirements, we have worked on a topology representation that extends existing standards and draws upon a number of current best-practices. Our goal is to describe existing networks as completely as possible while remaining flexible enough to handle future developments and additions. These efforts have recently culminated in what we call the Unified Network Information Service (UNIS) [10], providing a general schema and a deployable service for publishing named services and topology information in a globally distributed fashion.

```
<node id=''urn:unis:...:node=srv1''>
 <port id=''urn:unis:...:node=srv1:port=eth3''>
  <name>eth3</name>
  <description>10G connection to rtr1</description>
  <capacity>10000000000</capacity>
  <link id=''urn:unis:...:node=srv1:port=eth3:link=srv1−rtr1''>
    <relation type=''source''>
      <portIdRef>urn:unis:...:node=srv1:port=eth3</portIdRef>
    </relation>
    <relation type=''sink''>
      <portIdRef>urn:unis:...:node=rtr1:port=Te1/0/1</portIdRef>
    </relation>
  </link>
 </port>
 <rule id=''urn:unis:...:node=srv01:rule=eth3_vlan_607''>
  <type>LINUX_NET</type>
  <vlan>
   <id>607</id>
   <portIdRef>urn:unis:...:node=srv01:port=eth3</portIdRef>
  </vlan>
 </rule>
</node>

<path id=''urn:unis:...:path=srv1−circuit''
 <symmetric>true</symmetric>
...
  <id>2</id>
   <ruleIdRef>urn:unis:...:service=OSCARS:rule=circuit</ruleIdRef>
  </hop>
...
  <hop>
   <id>5</id>
   <ruleIdRef>urn:unis:...:node=rtr1:rule=Ge1/0/4_607</ruleIdRef>
   <ruleIdRef>urn:unis:...:node=rtr1:rule=vlan_607_trunk</ruleIdRef>
  </hop>
  <hop>
   <id>6</id>
   <ruleIdRef>urn:unis:...:node=srv1:rule=eth3_vlan_607</ruleIdRef>
  </hop>
</path>
```

Figure 10: A UNIS XML Path example with annotated rules

The UNIS schema builds upon the same base elements as defined in the NM-WG [6] topology schema used by perfSONAR and allows for interoperability between systems. These base elements allow networks to be described in terms of *domains*, *nodes*, *ports*, *links*, *networks*, *paths*, and *services* in a flexible and extensible manner. We have recently extended the UNIS schema to allow for annotations to these base elements so as to include network configuration, or "rule", information within the network topology description itself. An example of such a topology fragment is show in Figure 10. In this model, a network path in UNIS can be represented as a list of rule ref-

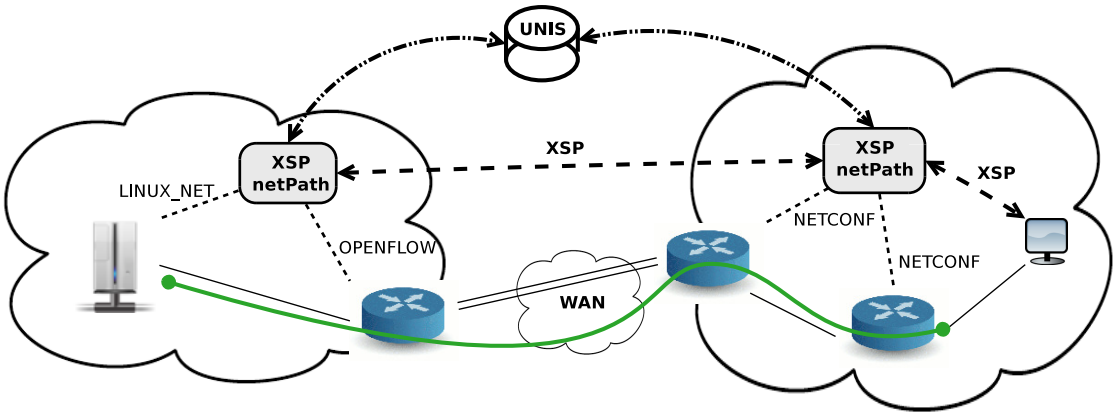

Figure 11: Conceptual view of the *netPath* XSP-SH being used to provide a common interface for dynamic network configuration, in this example creating a "virtual circuit" between two remote domains.

erences, which are annotations to *node*, *port*, and *service* elements within the topology description.

The similarity to the XSP netPath model is not by accident. In fact, resolving a UNIS *path* of rule references is intended to represent the same network configuration information in the equivalent xspNetPath structure. This has a number of benefits, not the least of which is the ability to maintain a consistent and exportable view of active paths and associated network device configuration within a given network domain. Clients interfacing with the UNIS service can query path information to learn the state of the network and actively modify device configurations using XSP *netPath*. In a similar fashion, XSP NEs may update UNIS with the current status of the device configuration by updating rule annotations in the topology description.

Using UNIS and XSP *netPath*, we are in the process of creating and evaluating the effectiveness of network environments similar to the one shown in Figure 11. Within each network, XSP NEs with *netPath* handlers manage the configuration of a number of network devices within a particular domain. These running XSP instances may be part of a deployed gateway such as Phoebus or are otherwise made available as standalone XSP services or host agents depending on the end-site deployment. By providing a standard interface and session negotiation between *netPath* handlers, XSP can automatically configure dynamic network paths on behalf of the requesting application based on topology information obtained from UNIS, bringing the "virtual circuit" all the way to the end-host if desired.

Finally, one of the additional features UNIS provides is that of service lookup and name resolution. If not resolveable via the local XSP NE configuration, the EIDs specified within each xspNetRule are located with queries to UNIS, which maintains registered service information along with network topology. One area we wish to explore is the implementation of a *Naming/Addressing* XSP-SH that would allow the XSP session layer to use UNIS in a more general way.

6 Status and Future Work

Our XSP use models have described solutions within two areas that play an important role in future Internet designs, namely performance and the management and configuration of network devices. Within our XSP framework, we anticipate extending functionality to the session layer in areas such as security, routing, forwarding, naming, and addressing that will bring new architectural features to users and applications through a common interface. We believe an added benefit of our session-based approach will help address the larger issues of availability, deployment, and economic viability in current and future networks. We also readily admit that this work is ambitious and is confronted with a number of unknowns as networks continue to evolve. However, it should be noted that our approach does not have to reinvent the Internet in one fell swoop, but rather, it may be one of gradual adoption–implementing a core set of services that can work alongside existing Internet infrastructure while enhancing and extending functionality where possible.

With the current XSP protocol and library implementations, we have solved one of the key barriers to developing new functionality in a consistent and easily accesible way. Through the XSP-SH layer, a particular implementation may come or go but the application interface to a standard set of services remains consistent. This allows for a great amount of flexibility in evaluating new architectural approaches over a variety of techniques. To support XSP deployments, we have developed a service known as the *XSP Daemon*, or XSPd, which implements the core XSP protocol functionality and a configurable XSP-SH layer. As

an XSP-enabled NE, XSPd can support new and developing session-layer features and interact with other XSP services like Phoebus and SLaBS. One common scenario is to let XSPd authenticate and authorize high-performance transfer applications while configuring dynamic network paths that enable Phoebus and SLaBS to accelerate the data channel, all within a single session spanning multiple NEs.

One of the biggest challenges in exploring new and emerging network technologies has involved the absence of a platform to experimentally validate new ideas in real-world environments without seriously disrupting production systems. To address this, and to supplement our local testbeds, we are actively experimenting with the Global Environment for Network Innovations (GENI) [24]. GENI provides a virtual laboratory for experimenters to develop and explore innovative and at-scale future Internet architectures, with a significant number of available resources designed to be fully virtualized and instrumented while at the same time enabling deep programmability. We are taking advantage of the heterogeneous network devices available at both core and edge networks with the ability to integrate existing or custom components as new, shareable resources within GENI itself. Our XSP experiments have shown promising results in configuring dynamic paths and employing Phoebus and SLaBS over the GENI backbone.

As future work, we plan to extend the XSP-BAG model as an active service on end-hosts in order to achieve better integration with the native operating system, supporting high-performance, bulk data transfers from disk-to-disk. With the availability of 40G and 100G network testbed on the horizon, we hope to evaluate both the end-host and SLaBS BAG transfer models at much higher network speeds.

An implementation of the XSP protocol as a native kernel module, using a sockets-compatible interface via an AF_INET_XSP mechanism, is also being considered. This would provide much tighter integration and improved performance for a number of data movement applications, while at the same time simplifying our session layer stack architecture.

Finally, we are investigating the role of UNIS to provide a core set of naming and addressing services. A growing area of research involves this notion of location and identifier separation and we are targeting UNIS to play an active role in this regard. By allowing service registration across both public and private address space, UNIS provides a mechanism for storing and distributing location-independent names (EIDs) for resource discovery and access. XSP-enabled NEs can use this discovery mechanism to locate the associated "border gateways" in the network that perform late-binding of existing transport layer addresses (e.g. TCP 5-tuple) and establish forwarding rules to provide access to the running service. We will also look into integrating related mechanisms such as LISP [2] and HIP [50] as additional XSP-SH modules.

7 Conclusion

This paper presents our XSP session layer and protocol implementation as an extensible framework for managing the interaction between applications and network-based services. Given the scope of our architecture, we have focused on two use models in particular and describe how XSP is being applied to improve performance in modern networks and dynamically configure dedicated network paths on which our performance-enhancing middleware services depend. We illustrate our Bulk Asynchronous GET approach as a model for how XSP can merge technologies and services from opposite ends of the networking spectrum, from data centers to wide-area networks, into a single, common interface for applications and users. As the Internet continues to evolve, we believe XSP is positioned as a general and extensible platform for exploring future internet architectures.

References

[1] Cisco Application Extension Platform. `http://www.cisco.com/en/US/products/ps9701/`.

[2] Cisco locator/id separation protocol. `http://lisp4.cisco.com/`.

[3] Connectx-2 EN with RDMA over Ethernet (RoCE). `http://www.mellanox.com/related-docs/prod_software/ConnectX-2_RDMA_RoCE.pdf`.

[4] Juniper Junos SDK. `http://www.juniper.net/us/en/products-services/nos/junos/junos-sdk/`.

[5] Mobilityfirst future internet architecture project. `http://mobilityfirst.winlab.rutgers.edu/`.

[6] Network Measurement Working Group. `http://nmwg.internet2.edu/`.

[7] NSF NeTS: Future Internet Design Initiative. `http://www.nets-find.net/`.

[8] NSIS: Next steps in signaling. `http://datatracker.ietf.org/wg/nsis/charter/`.

[9] OpenFlow. `http://www.openflowswitch.org/documents/openflow-wp-latest.pdf`.

[10] Unified Network Information Service. `https://spaces.internet2.edu/download/attachments/6881319/UNIS_white_paper.pdf`.

[11] Requirements for internet hosts - communication layers, 1989.

[12] T. Benjegerdes. Infiniband and openfabrics at sc06. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM.

[13] M. S. Blumenthal and D. D. Clark. Rethinking the design of the internet: the end-to-end arguments vs. the brave new world. *ACM Trans. Internet Technol.*, 1(1):70–109, 2001.

[14] A. Bobyshev, M. Crawford, P. DeMar, V. Grigaliunas, M. Grigoriev, A. Moibenko, D. Petravick, and R. Rechenmacher. Lambda station: On-demand flow based routing for data intensive grid applications over multitopology networks. In *Proceedings of the 3rd Intenational Conference on Broadband Communications (IEEE)*, 2006.

[15] S. Bryant, G. Swallow, L. Martini, and D. McPherson. Pseudowire Emulation Edge-to-Edge (PWE3) Control Word for Use over an MPLS PSN. RFC 4385 (Proposed Standard), February 2006. Updated by RFC 5586.

[16] V. G. Cerf, S. C. Burleigh, A. J. Hooke, L. Torgerson, R. C. Durst, K. L. Scott, K. Fall, and H. S. Weiss. Rfc 4838: Delay-tolerant network architecture. `http://www.ietf.org/rfc/rfc4838.txt`, April 2007.

[17] J. Day. *Patterns in Network Architecture: A Return to Fundamentals*.

[18] J. D. Day and H. Zimmermann. The osi reference model. *Proceedings of the IEEE*, 71(12):1334–1340, 1983.

[19] M. Demmer and K. Fall. The design and implementation of a session layer for delay-tolerant networks. *Comput. Commun.*, 32(16):1724–1730, 2009.

[20] A. Doria, R. Haas, J. Hadi Salim, H. Khosravi, and W. M. Wang. ForCES Protocol Specification. Internet Draft draft-ietf-forces-protocol-22.txt, March 2009.

[21] R. Enns. NETCONF Configuration Protocol. RFC 4741 (Proposed Standard), December 2006.

[22] A. N. S. for Information Processing Systems. Open systems interconnection – basic connection oriented session protocol specification. ANSI/ISO 8327-1987, 1992.

[23] B. Ford and J. Iyengar. Breaking up the transport logjam. In *Proceedings of ACM HotNets*, 2008.

[24] Geni: Global Environement for Network Innovations. `http://www.geni.net/`.

[25] P. Geoffray. Myrinet express (mx): Is your interconnect smart? In *Proceedings of the High Performance Computing and Grid in Asia Pacific Region, Seventh International Conference*, pages 452–452, Washington, DC, USA, 2004. IEEE Computer Society.

[26] B. Gibbard, D. Katramatos, and D. Yu. Terapaths: A qos-enabled collaborative data sharing infrastructure for peta-scale computing research. In *Proceedings of the 3rd Intenational Conference on Broadband Communications (IEEE)*, 2006.

[27] GridFTP. `http://www.globus.org/datagrid/gridftp.html`.

[28] Y. Gu and R. Grossman. Udt: Udp-based data transfer for high-speed wide area networks. *Computer Networks (Elsevier) Volume 51, Issue 7.*, 2007.

[29] S. Guha. P.: An end-middle-end approach to connection establishment. In *In: Proceedings of SIGCOMM.07, Kyoto*, 2007.

[30] A. Hanemann, J. Boote, E. Boyd, J. Durand, L. Kudarimoti, R. Lapacz, M. Swany, S. Trocha, and J. Zurawski. PerfSONAR: A service oriented architecture for multi-domain network monitoring. In *Proceedings of the Third International Conference on Service Oriented Computing (ICSOC 2005)*, ACM Sigsoft and Sigweb, pages 241–254, December 2005.

[31] Y. Ismailov, K. Palmskog, M. Widell, P. Arvidsson, and Y. Wang. Session layer resurgence: Towards mobile, disconnection- and delay-tolerant communication. In *ECUMN '07: Proceedings of the Fourth European Conference on Universal Multiservice Networks*, pages 337–345, Washington, DC, USA, 2007. IEEE Computer Society.

[32] E. Kissel and M. Swany. Session layer burst switching for high performance data movement. In *Proceedings of PFLDNet*, 2010.

[33] E. Kissel, M. Swany, and A. Brown. Improving gridftp performance using the phoebus session layer. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–10, New York, NY, USA, 2009. ACM.

[34] E. Kissel, M. Swany, and A. Brown. Phoebus: A system for high throughput data movement. *J. Parallel Distrib. Comput.*, 71:266–279, February 2011.

[35] G. Kramer and G. Pesavento. Ethernet passive optical network (epon): building a next-generation optical access network. *Communications magazine, IEEE*, 40(2):66–73, 2002.

[36] T. Lehman, X. Yang, C. P. Guok, N. S. V. Rao, A. Lake, J. Vollbrecht, and N. Ghani. Control plane architecture and design considerations for multi-service, multi-layer, multi-domain hybrid networks. In *High Speed Networking Workshop, INFOCOM 2007*, May 2007.

[37] M. Oberg, H. M. Tufo, T. Voran, and M. Woitaszek. Evaluation of rdma over ethernet technology for building cost effective linux clusters. `http://www.linuxclustersinstitute.org/conferences/archive/2006/PDF/34-Oberg_M_final.pdf`.

[38] ESnet On-demand Secure Circuits and Advance Reservation System (OSCARS). `http://www.es.net/oscars/`.

[39] C. Y. M. Qiao. Optical Burst switching (obs) - a new paradigm for an optical internet. *Journal of High Speed Networks*, 8:69–84, 1999.

[40] M. Rose. The Blocks Extensible Exchange Protocol Core, March 2001.

[41] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, and M. H. andE. Schooler. Sip: Session initiation protocol. RFC 3261, June 2002.

[42] B. Ruffin, J. Downie, and J. Hurley. Purely passive long reach 10 ge-pon architecture based on duobinary signals and ultra-low loss optical fiber. *Optics Society of America*, 2008.

[43] J. Salz, A. C. Snoeren, and H. Balakrishnan. Tesla: A transparent, extensible session-layer architecture for end-to-end network services. In *In Proc. of the Fourth USENIX Symposium on Internet Technologies and Systems (USITS*, pages 211–224, 2003.

[44] SCTP. `http://www.sctp.org/`.

[45] H. Song, B.-W. Kim, and B. Mukherjee. Long-reach optical access networks: A survey of research challenges, demonstrations, and bandwidth assignment mechanisms. *Communications Surveys and Tutorials, IEEE*, 12(1), 2010.

[46] P. Srisuresh, J. Kuthan, J. Rosenberg, A. Molitor, and A. Rayhan. Middlebox communication architecture and framework. RFC 3303, August 2002.

[47] M. Swany. Improving Throughput for Grid Applications with Network Logistics. In *Supercomputing 2004*, November 2004.

[48] M. Swany and R. Wolski. Data logistics in network computing: The Logistical Session Layer. In *IEEE Network Computing and Applications*, October 2001.

[49] R. Wang, X. Wu, T. Wang, and T. Taleb. Experimental evaluation of delay tolerant networking (dtn) protocols for long-delay cislunar communications. In *GLOBECOM'09: Proceedings of the 28th IEEE conference on Global telecommunications*, pages 3497–3501, Piscataway, NJ, USA, 2009. IEEE Press.

[50] J. Ylitalo, J. Melén, P. Salmela, and H. Petander. An experimental evaluation of a hip based network mobility scheme. In *WWIC'08: Proceedings of the 6th international conference on Wired/wireless internet communications*, pages 139–151, Berlin, Heidelberg, 2008. Springer-Verlag.