1 XSP Draft

1.1 eXtensible Session Protocol

One of the primary goals of XSP is to provide a general and easily extensible protocol specification that defines the interaction between network devices, services, middleware frameworks, and applications. Building upon existing session protocol recommendations, XSP presents an interface for control and data exchange between enabled peers or network end-points. A survey of existing technologies for managing data movement over current network infrastructure and systems has made it clear that a unified or generic solution for network service management and control has not emerged. The challenge is made greater when acknowledging the heterogeneity of network architectures, transport layer services, and application interfaces that exist in modern networks.

We believe that XSP can help fill this gap by providing a coherent and extensible session layer protocol and framework for applications and middlebox services that must support these increasingly diverse networks and network devices. The basis for XSP is modeled on the ITU-T Recommendation X.225 connection-oriented session protocol specification [4]. Most relevant for this work, the recommendation outlines the procedures for “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.” This excerpt provides the conceptual basis for XSP.

More specifically, as a session layer protocol, XSP must provide for the establishment, termination, and negotiation of a session between end-user application processes. This negotiation consists of a set of requests and responses (session-layer PDUs) between network end-points that run a session-layer protocol implementation. A “session” might be responsible for a number of different tasks including authentication, enforcement of policy, checkpointing, synchronization, and recovery of existing session state. Importantly, the Session layer allows multiple connections from lower layers to be bound and managed within a single session construct. This enables a number of functions:

- Mapping of session addresses onto transport addresses;
- Selection of transport quality of service and tuning parameters;
- Negotiation of session parameters;
- Transfer of transparent user data;
- Ability to distinguish between session connections;
- Recovery and synchronization of transport connections.

While the functions described above are all defined within the existing X.225 specification, the broader role of the Session layer has been a topic of significant contention and debate since its inception via the OSI standards body. However, regardless of the debate, we have identified a real need for the functions defined within the Session layer and have begun to demonstrate their potential for evolving networks through the development of XSP.

A key insight to XSP 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 increasing the performance of the configured data channel when necessary. As a central use case, selecting transport layer protocols and optimizations via a session protocol are crucial to services like Phoebus. In
general, managing a number of separate transport-specific features allows us to take advantage of an end-to-end path that may require the use of path segments that have particular characteristics.

The heterogeneity of network devices on the Internet provides a generic example of the role XSP can play. A typical end-to-end path through the Internet may traverse a number of different devices and unique network segments. Data transfer clients may request resources from a dynamic circuit service, require some transformation service deployed in the network, or may need to traverse a firewall/NAT device to reach a remote end-point. By establishing a session between each intermediate device, XSP has the ability to negotiate and setup the data path on behalf of the client and apply optimizations along each segment as appropriate. Thus, XSP may signal the circuit service to instantiate a dedicated optical link, request compression for the data stream, and signal an XSP-enabled controller to activate firewalls rules in order to access a protected service. Each of these use-cases may be accomplished in a manner transparent to the client or explicitly controlled via a well-defined API. The goal of XSP is to provide the signaling required for network services to exchange and negotiate capabilities in a common and extensible framework.

We propose the following core research areas for XSP:

- **Protocol specification**: The development of a flexible specification for XSP that includes extensible option types. XSP will be designed as a binary protocol with minimal overhead and must include comprehensive library support.

- **Session state management**: Support for stateful sessions between XSP instances, where state shall be maintained over the lifetime of the session independent of the connection status of each instance. This will allow the protocol to “rendezvous” with an existing session state.

- **Transport layer negotiation and checkpointing**: Develop control and optimization of the underlying transport layer as a key feature of XSP. A number of layer-4 connections may be bound to a single session, requiring appropriate synchronization and checkpointing of the connection states.

- **Endpoint ID addressing and resource locators**: Investigate service lookup and addressing of XSP endpoints via a common URN schema while remaining compatible with existing 5-tuple addressing.

- **Security and admission control**: Provide XSP endpoint authentication and/or encryption via TLS/SSL with X.509 certificates. The security mechanism may be negotiated at session establishment and should also support negotiating application-specific authentication methods.

Given these desired properties and the lessons learned from our previous endeavors, the following sections outline some initial ideas for the XSP message format and supported functionality. The XSP specification is loosely based on the original Phoebus session protocol with a new focus on providing the core XSP features described above. We anticipate that the protocol will undergo a number of revisions as we continue to implement and develop these ideas.

### 1.1.1 XSP Messages

The basic XSP PDU structure consists of a protocol header defining the following fields: version, flags, message type, option count, 128 bit source and destination IDs, a 128 bit session ID, and
48 bits of reserved space. The message is shown in Figure 1. In addition to this common header there is a variable size option block consisting of an option type field, a service port field, a length field, and a variable size field for option data. These fields are described in Table 1. Similarly 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.

Following the basic XSP header, zero or more option blocks may be transmitted. These XSP option blocks may define requested transport layer capabilities, articulate an end-to-end path, or configure any other features currently supported along each network segment. This exchange takes place as a negotiation between XSP-enabled network elements (NEs) spanning the end points defined by the established session. The XSP header is to remain relatively simple. It specifies the source and destination associated with an E2E session and leaves any intermediate NEs and control information along the path to be specified within option blocks.

As the name implies, one of the key benefits in utilizing XSP is the extensible nature of the protocol. Protocol functionality may be easily extended by defining additional option block types to be communicated across XSP-enabled NEs. Option types may be explicitly defined by an application utilizing the XSP protocol library, requiring the application to specify the option data encoding and serialization/deserialization methods. This technique allows XSP to communicate application-specific information in a general manner while simultaneously maintaining session state between a number of NEs.

<table>
<thead>
<tr>
<th>V</th>
<th>Flags</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option Count</td>
<td>Reserved</td>
<td>SRC EID</td>
<td>DST EID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Option Type</th>
<th>Service Port</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option Data</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: *The XSP PDU.*
Based on local configuration and implementation of features, a particular NE may accept and process a given option type or otherwise ignore and forward the option block to the next hop. 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 option may be forwarded to the session endpoint without additional overhead. This distinction is useful when an E2E path may include a number of NEs and session data is signaled along the forwarding path, but the message is only parsed at those NEs that are expecting the message.

A number of XSP primitives must also be defined to provide basic protocol functionality, which correspond to XSP message types defined by the Type field. We have listed a number of such primitives in Table 2. This list represents a minimal set of functionality to be initially introduced as part of XSP; additional primitives will be developed as the need arises and the specification matures.

<table>
<thead>
<tr>
<th>Header Field</th>
<th>Size (bits)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>4</td>
<td>Protocol version (e.g. 1)</td>
</tr>
<tr>
<td>Flags</td>
<td>12</td>
<td>Message flags for the message. (e.g. priority)</td>
</tr>
<tr>
<td>Type</td>
<td>16</td>
<td>Specifies the type of XSP message. May indicate expected option block types.</td>
</tr>
<tr>
<td>Length</td>
<td>32</td>
<td>Total length of the message in bytes.</td>
</tr>
<tr>
<td>Option Count</td>
<td>16</td>
<td>Specifies the number of option blocks associated with the XSP message.</td>
</tr>
<tr>
<td>SRC EID</td>
<td>128</td>
<td>The source identifier for this message. (e.g. IPv6 address)</td>
</tr>
<tr>
<td>DST EID</td>
<td>128</td>
<td>The destination identifier for this message.</td>
</tr>
<tr>
<td>Session ID</td>
<td>128</td>
<td>The session identifier for this message, i.e. a random hex string.</td>
</tr>
<tr>
<td>Option Type</td>
<td>16</td>
<td>Specifies the option type for the given option block.</td>
</tr>
<tr>
<td>Service Port</td>
<td>16</td>
<td>Specifies an optional layer-4 port associated with the option data. May indicate if the service addressed by the XSP hop is interested in the option block.</td>
</tr>
<tr>
<td>Option Data</td>
<td>&lt; 2^32</td>
<td>Variable size option data.</td>
</tr>
</tbody>
</table>

Table 1: Definition of XSP PDU fields.

Based on local configuration and implementation of features, a particular NE may accept and process a given option type or otherwise ignore and forward the option block to the next hop. 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 E2E path may include a number of NEs and session data is signaled along the forwarding path, but the message is only parsed at those NEs that are expecting the message.

A number of XSP primitives must also be defined to provide basic protocol functionality, which correspond to XSP message types defined by the Type field. We have listed a number of such primitives in Table 2. This list represents a minimal set of functionality to be initially introduced as part of XSP; additional primitives will be developed as the need arises and the specification matures.

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSP_MSG_SESS_OPEN</td>
<td>Establish a new XSP session. Session hops may be specified in the XSP_OPT_HOP option block type.</td>
</tr>
<tr>
<td>XSP_MSG_SESS_CLOSE</td>
<td>Close an existing XSP session.</td>
</tr>
<tr>
<td>XSP_MSG_ACK</td>
<td>Acknowledge a received XSP message. May contain additional information within an option block.</td>
</tr>
<tr>
<td>XSP_MSG_NACK</td>
<td>Negatively acknowledge a received XSP message. May contain an error indication within an option block.</td>
</tr>
<tr>
<td>XSP_MSG_DATA_OPEN</td>
<td>Open a new data connection bound to the current session.</td>
</tr>
<tr>
<td>XSP_MSG_DATA_CLOSE</td>
<td>Close a data connection bound to the current session.</td>
</tr>
<tr>
<td>XSP_MSG_DATACHK</td>
<td>Checkpoint or synchronize data connection. Data connection status indicated within XSP_OPT_DATA_CHK option block type.</td>
</tr>
<tr>
<td>XSP_MSG_APP_DATA</td>
<td>Indicate that the message contains application-specific data within option blocks. The option count field must be greater than zero.</td>
</tr>
<tr>
<td>XSP_MSG_AUTH_TYPE</td>
<td>Indicate authentication type.</td>
</tr>
</tbody>
</table>

Table 2: Preliminary XSP primitives.

1.1.2 Protocol Signaling and Library Support

XSP message exchanges will be supported in two standard ways: in-band and out-of-band signaling. In the in-band case, XSP messages will share the same connection used by the application for data transfer. This is useful when a single transport layer connection is forwarded along a series of performance enhancing NEs, which is supported in the Phoebus model. One of the apparent advantages is that no additional control connection is required, simplifying the communication model. The downside is that additional overhead is introduced within the data stream. Performing
signaling out-of-band with a separate control connection eliminates this data channel overhead while providing better flexibility. Figure 2 illustrates both possibilities.

(a) XSP in-band signaling.

(b) XSP out-of-band signaling.

Figure 2: XSP signaling between supported network elements.

One of the challenges we will investigate is how to best manage a number of client-initiated connections bound to a single session. For example, an XSP session may manage a number of parallel TCP connections used by an application between two endpoints. Within the network, persistent services might also bind a number of transport connections that share a common destination in order to apply optimizations and signal the next NE hop. Such a scenario is a potential use-case in SLaBS where flows are aggregated within a data burst and signaling is performed out-of-band. In both cases, connections are to be managed via the XSP \texttt{DATA\_OPEN}, \texttt{DATA\_CLOSE}, and \texttt{DATA\_CHK} primitives and connection state must be signaled to all NEs involved in the active session. The XSP client library must provide a high-level API for applications and services to effectively utilize this protocol functionality.

Prototyping and developing the XSP library, or \texttt{libXSP}, will be a significant undertaking in support of our research efforts. In addition to the XSP API providing a high-level session interface, \texttt{libXSP} will also export a sockets-like client library to remain compatible with data movement applications that can take advantage of services like Phoebus via in-band signaling. This feature will also provide a client wrapper for legacy applications to transparently use XSP in Unix-like environments by intercepting common socket calls like \texttt{connect}, \texttt{socket}, \texttt{send}, and \texttt{recv}, etc.

The usage model for applications is realized through the XSP client library or transparent wrapper. In order to develop XSP-enabled services, \texttt{libXSP} will implement the main protocol functionality, including configurable connection listeners and hooks for supporting application-specific XSP option blocks via external modules. These modules implement the application message format and serialization/deserialization methods needed in order to decode and process the application-defined option blocks. A service developed using XSP will register its option block handlers with \texttt{libXSP} thus enabling the processing of option block types associated with the service via the \texttt{APP\_DATA} primitive. An XSP-enabled service receiving an \texttt{APP\_DATA} message will iterate over the option
blocks and parse only those options that the deployed service has implemented and registered with libXSP. Depending on configuration and policy, the NE may forward all or none of the remaining option blocks to other NEs associated with the session.

This model of extensible option blocks will enable a number of XSP modules to be easily developed and support a number of network services. In addition to Phoebus-SLaBS, we will investigate developing XSP modules for DNE interfaces and available programmable router platforms as new ways to provide improved network performance.

1.1.3 Persistent Sessions

An interesting feature for XSP is the idea of maintaining state associated with an active session and having the ability to restore that state when requested. This state would persist beyond the duration of a session connection and would require maintaining a ledger of active sessions, to be implemented within libXSP. Such functionality would allow a number of negotiated configurations or settings associated with the session to be immediately applied when the session state is restored.

Implementing this feature could be accomplished via a “session-rendezvous” capability. Within the SESS_OPEN message, specifying an active session ID would indicate that the newly requested session should resume operation at the previously established session state. This method would need to be applied to all NEs that are associated with the particular session; however, it is not assumed that every application will require that an active session connection be maintained across all NEs. An XSP session connection might last only long enough to update the configuration of an NE, leaving the session state active on the NE when the connection closes. A number of otherwise disconnected NEs could be signaled in this manner. If an attempt is made to rendezvous with a non-existent session, a new session will become active on the NE using the specified session ID. It should be noted that the chance of a collision on the session ID is virtually nonexistent given the 128 bit length of the field.

Session state should remain persistent until explicitly signaled via SESS_CLOSE or when an optional session lifetime has expired. This lifetime could be specified within an option block when the session is first established. An open question is whether or not state should be maintained only during the lifetime of the XSP-enabled service or should some mechanism exist to save state to non-volatile storage. There are circumstances where both policies would be desirable, but we will consider the former as the most predominant and practical use-case.

One direct application of persistent session state is the management of transport connections over time. Some TCP stacks (Linux Kernel since 2.4) employ a “retentive TCP” feature. This mechanism caches TCP control information (cwnd, rtt, rttvar, ssthresh, and reorder) in the routing table for a particular destination. This information is cached for up to 10 minutes so that any subsequent connections to that destination will use the cached values during that period. Unfortunately, this caching effectively imposes an artificial cap on cwnd and ssthresh. This means that if a previous transfer experienced loss and reduced its window, all new connections to the same destination will use the reduced cwnd value and will not even try to increase its window. The cached ssthresh will also cut off the slow-start for the next transfer.

Although TCP metric caching is frequently disabled for high-performance applications, this idea of saving and explicitly setting TCP control information makes sense in the right context. It may be desirable to “pre-tune” the TCP connection with parameters that are known to have achieved good performance for earlier connections. By binding TCP connections to an E2E session construct,
XSP could provide much better insight into the state of the network than is currently achieved by the reactive process employed by TCP. With this knowledge, XSP could tell the endpoint what TCP parameters to start out with based on the current session state, which is a function of the associated NEs and link characteristics over which the session is maintaining active connections. These starting parameters can be saved as part of the persistent session state for the lifetime of the session. Over the duration of a transport connection, XSP could also explicitly adjust cwnd to avoid aggressive backoff due to transient congestion events perceived by TCP at the edge.

Some other applications that could take advantage of persistent sessions include services that require infrequent updates to their configuration. Potential use-cases include Internet routing as in BGP and perhaps IGP deployments, DNEs where circuit resources are reserved for extended periods of time, and NEs with programmable hardware. We plan to further develop ideas for these and other applications of XSP as the protocol design matures.

1.1.4 Supporting Location Independent Endpoint IDs

The distinction between how a device is attached to the network (location) and “who” or what that device is (identity) can be facilitated by considering a session of communication, signaled and managed using XSP. This distinction is becoming ever more important as systems become increasingly virtualized and as services may exist across multiple address spaces throughout their lifetime. A particularly compelling scenario is that of a service residing in an RFC1918 [6] network address space yet remaining publicly addressable via a globally registered service name. Firewall and NAT traversal protocols such as STUN, TURN, and SOCKS, used to support access to devices in private address space are well-understood and will not be rehashed here. We note that XSP is in a position to take advantage of these existing techniques while providing name-routing to services within the protocol framework.

A number of methods for separating location and identity have been proposed (LISP, i3, NUTTS, etc.). With XSP, we intend to build upon these developments while investigating how to remain compatible with existing Internet addressing and supporting a new endpoint identifier (EID) addressing and routing capability. The immediate research task involves integrating the UNIS URN schema within XSP and implementing service registration to XSP-enabled gateways that utilize the URNs as routable EIDs. By doing so, XSP will be able to support existing services that take advantage of this schema while providing the necessary signaling to apply access control and policy.

The identifiers specified within UNIS have been designed to be globally unique, human-readable, and extensible, following in the style of URNs. A proposed namespace for the UNIS URN is “unis”, with an EID subnamespace of “eid”. All identifiers would begin with “urn:unis:eid”. The service identifiers would employ 4 major fields: domain, node, service, and port. The identifiers also provide a natural hierarchy. For example,

```
urn:unis:eid:domain=cis.udel.edu:node=blackseal:service=http
```

specifies that the HTTP service is available on the host blackseal within the cis.udel.edu domain. If a service is provided for an entire domain, then the service field should immediately precede domain. To specify a non-standard service port, the identifier becomes

```
urn:unis:eid:domain=cis.udel.edu:node=blackseal:port=9091:service=https
```

to indicate that an HTTPS service is available on port 9091. The “*” wildcard may also be used when a particular field is unknown.
urn:unis:eid:domain=cis.udel.edu:node=blackseal:port=:service=http

This URN schema is only a small part of the complete UNIS specification, which is being developed as part of the schema being standardized by the OGF NML Working Group. By adopting and extending this schema, XSP will remain compatible with a wide range of network and service descriptions that utilize this standard representation.

Supporting location/identifier separation requires the use of gateways in the network that accept EID registrations from active service endpoints. These gateways, acting as registrars, then provide late-binding of the EID to the layer-4 address of the service (IP/port).\(^1\) The gateways themselves could be deployed in a number of ways, either as stand-alone registrars or as part of an existing network service deployment. One possibility we will investigate is enabling Phoebus Gateways to provide EID addressing capability while operating in unison with transport layer optimizations of the forwarded service data path.

Another use-case is that of a dedicated XSP-enabled gateway providing EID service registration and firewall/NAT configuration at the edge of a private network. Such a scenario is depicted in Figure 3. A client is interested in some data provided by “myservice” that has been discovered via a UNIS information store. UNIS identifies this service via an EID, which must now be located. The client contacts its nearest XSP-enabled NE with a service request for “myservice” and locates the GW acting as a service registrar for the domain. The GW is aware of the private address of the server that has registered “myservice” and configures the firewall to install NAT rules for the registered service IP and port number. Once the client is notified of the public address specified in the firewall configuration, it may use XSP to instantiate a data path to the server.

Admittedly, this description is very high-level, and the intent here is only to demonstrate the role XSP could play in supporting location/identity separation. A number of questions need to be answered to fully realize this capability, including how to best manage and distribute routing tables between XSP gateways, what signaling is necessary to enable efficient EID registration and lookup requests, and what device support is required to provide the appropriate firewall/NAT

\(^1\)It is conceivable that the port field is actually not necessary in the EID schema as the service registration would indicate the listening socket port number for the registered service name.
configuration. While a full implementation of such a system is perhaps outside of this proposal’s scope, we intend to research and develop a sufficient prototype implementation within *libXSP* that can support EID addressing for Phoebus-SLaBS services.

1.1.5 Security and Trust

Authorization, authentication, and accounting (AAA) considerations are often after-thoughts in many system designs. In order to avoid this common pitfall, we propose to include basic security and user identity as an initially supported feature of XSP. XSP will depend on the OpenSSL library [2] to provide user and service authentication via X.509 certificates and encryption over the session channel. Both authentication and encryption may be negotiated during XSP session establishment, allowing both endpoints to control available AAA capabilities.

A common scenario is authenticating a valid user when an application seeks to access an NE via XSP. The XSP service running on the NE would maintain one or more Certificate Authorities (CAs) to which user or host certificates may be validated against. Assuming a client/server model, when a client opens a new session to the service, it would indicate that it requires a valid X.509 certificate in the option block associated with the *AUTH_TYPE* message. If the authentication method is present, the client would start an SSL context and verify its certificate over the open XSP session. In this manner, the server may also be verified to the client by presenting a valid signature. If the current authentication method fails or is not available, the client may send a *NACK* and other authentication methods may be negotiated via XSP if supported.

Establishing a chain of trust among XSP-enabled NEs is another important consideration. Depending on the use-case, a user or host certificate may require verification at each hop associated with an active session. In some cases, a transitive trust arrangement may be sufficient: if a client A is granted access to service B, then A is also granted access to service C, provided that B is authorized to access C. More complex access and usage policy may be expressed using Security Assertion Markup Language (SAML) assertions negotiated with XSP *AUTH_TYPE* option blocks.

Beyond providing TLS/SSL, the XSP security model should remain flexible enough to provide application-specific AAA methods when required. Independent of the built-in XSP mechanisms, an application or XSP module may implement some specific security policy that would be communicated between two endpoints for the duration of the session. Such a mechanism would be completely defined and associated with a particular application and would be communicated via *APP_DATA* messages along with any other application-specific data. To provide more flexible client support, future work for the XSP client library could include an implementation of the Generic Security Services API (GSS-API) [5] so that client applications may use their existing, compatible security services (e.g. Kerberos [1]) with XSP.

Finally, federated single sign-on and attribute exchange frameworks like Shibboleth [3] are additional avenues for XSP security-related work. This model of organizational identity is a potentially important one when XSP-enabled services are to be deployed in a federated environment.

References


