

A transport layer architectural framework for optical burst switched (OBS) networks

Arnold Bragg[†], Ilia Baldine and Dan Stevenson

Advanced Networking Research Division

MCNC Research and Development Institute

3021 Cornwallis Road, Research Triangle Park NC 27709 USA

ABSTRACT

We discuss technical issues and general requirements for a transport layer architecture (*viz.*, services and protocols) for optical burst switched (OBS) networks. We focus on: **(1)** the OBS transport layer, and **(2)** on ancillary services above and below the OBS transport layer that may be required to implement or augment traditional transport layer services. The OBS transport layer and its ancillary services collectively comprise: **(1)** a reference model, **(2)** a suite of transport services, and **(3)** a conceptual transport protocol.

Keywords: Optical burst switching, OBS, transport layer, transport protocol, transport services.

1. INTRODUCTION

There are at least four reasons why a transport layer architecture for OBS networks will likely differ from transport layer architectures for other networks, especially networks that use IP or a comparable layer 3 datagram service:

(1) The network architectures are fundamentally different¹⁻⁵ (Figure 1). OBS architectures typically do not buffer bursts in the all-optical core, and intermediate burst switches provide no IP-like *network* layer services. OBS architectures typically do not require a *link* layer protocol in the optical domain. Traditional link layer services are either not provided, or are provided by an end-to-end protocol, or are severely constrained. Service differentiation in IP/datagram networks typically requires resource reservations and/or packet queuing and active queue management at intermediate nodes. In contrast, service differentiation in OBS networks is performed at OBS edge nodes (except for burst-level priority-based preemption, and experimental mechanisms like burst aging and burst segmentation with head drop^{6,7}). The OBS transport layer architecture will have to take on some traditional network layer, link layer, and service quality functions.

(2) OBS networks inherently support a range of circuit- and packet-like services. OBS is technologically positioned between all-optical circuit switching and all-optical packet switching²⁻⁴.

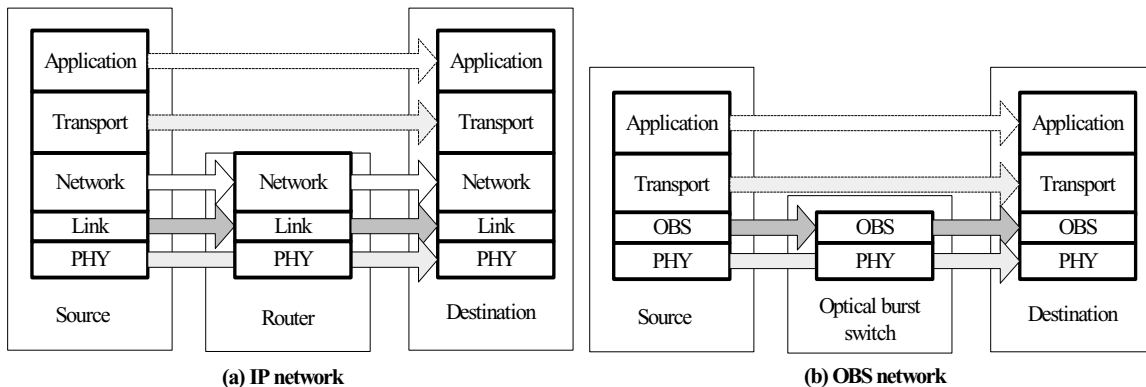


Figure 1. Generic (a) IP and (b) OBS protocol architectures.

Most OBS variants can support circuit-like and packet-like *services* by varying burst lengths, and/or by establishing long-lived ‘light paths’. Light paths are analogous to provisioned circuits, and support pseudo wire and circuit emulation services. Some OBS variants support ‘persistent paths’ (pinned routes), which are analogous to burst-multiplexed fixed-route tunnels. Single bursts, and burst streams that are individually and dynamically routed are roughly analogous to a bufferless optical packet-switched service (with burst overhead). OBS ‘circuits’ and ‘tunnels’ can be provisioned in tens of microseconds (depending on switch configuration times), and have lifetimes from a few milliseconds to many hours.

[†] Corresponding author. Contact information: {abragg, ibaldin, stevenso} @ anr.mcnc.org .

The OBS transport layer architecture will greatly benefit from OBS' triform (circuit-, tunnel-, packet-like) character.

(3) Some transport layer services can be provided by OBS services. There is little value in requiring the transport layer to provide redundant services. *E.g.*, OBS pinned routes guarantee sequenced delivery without transport-layer sequence numbering and reassembly mechanisms. End-to-end flow control may not be required over circuit-like OBS light paths. Sources receive nearly instantaneous indications of channel blocking over the OBS signaling channel, so there is no need for TCP-like timeouts or other congestion avoidance mechanisms. The OBS transport layer architecture will likely map some transport layer services directly onto OBS layer services.

(4) Many transport protocols have performance limitations when used in optical WANs, especially lossy protocols optimized for noisy, low-bandwidth links^{8,9}. These limitations have spawned a great deal of research on high-performance transport protocols for IP networks[†]. Feature sets include support for time-critical and scheduled transfers, high-performance network interfaces, OS bypass, network-aware OSs, service differentiation across administrative domains, *etc.* The OBS transport layer architecture must support a number of new features.

We discuss functions, services, features, issues, and requirements for a transport layer architecture for OBS networks. We focus on the OBS transport layer and on ancillary functions above and below the transport layer that may be required to implement both traditional and novel transport layer services within the architecture. These collectively comprise a reference model, a suite of transport services, and a conceptual transport protocol.

2. APPROACH

The OBS transport layer architecture focuses on services between applications and the OBS and physical layers in Figure 1b. For downward flows, it maps applications' requirements into services provided by the transport and/or OBS layers, and into information elements required by the OBS and physical layers. For upward flows, it traps and processes transport-service alarms, exceptions, error conditions and other indications originating from lower layers at the source, from intermediate OBS switches, and from the destination. The architecture must:

- Capture and convey an application's QoS requirements – *e.g.*, 'sequenced payload delivery'.
- Map QoS requirements onto OBS connection attributes – *e.g.*, 'use an OBS persistent path'.
- Map expedited delivery requests – *e.g.*, 'preempt bursts with priorities less than 0x011'.
- Manage transport layer 'connections' if a connection-oriented service is requested – *e.g.*, establishment, normal and abnormal termination, error traps.
- Provide input to OBS session, path, and data transport components.
- Augment OBS status reporting and exception handling when relevant to the transport layer.
- Utilize well-defined interfaces (API, RPC, message passing, DMA, RDMA, *etc.*).
- Provide input to physical layer components – *e.g.*, 'utilize Reed-Solomon FEC'.
- Support unique OBS features – *e.g.*, 'utilize adaptive methods to mitigate retransmission'.
- Support typical transport layer services – *e.g.*, 'utilize end-to-end flow control'.
- Support hardware-based acceleration (*e.g.*, for FEC calculations) and hardware/software partitioning to optimize performance.
- *Etc.*

The architectural framework consists of a reference model and a set of design goals. **(1)** The reference model includes applications, the OBS transport layer (OTL), the OBS and physical layers, and optional services above and below the OTL. The optional services do **not** compose an architectural layer, adaptation layer, or protocol layer *per se*; they appear in the reference model (Figure 2) but not in the OBS protocol architecture (Figure 1b). **(2)** The design goals include assumptions about the OBS network, a conceptual view of the OTL, and requisite features.

2.1 Reference model

The reference model (Figure 2) has five components:

(1) An OBS source (or edge node) hosts one or more concurrent **applications**. An application has one or more active traffic 'streams'. Streams may have different QoS requirements and/or different destinations. Streams may use one or more transport layer connections and/or connectionless services.

[†] A short list includes protocols optimized for performance, throughput, service quality, and/or reliability: High Speed TCP¹⁰, Caltech FAST¹¹, various parallel TCPs¹², Grid TCP¹³, TCP Westwood¹⁴, Stream Control Transmission Protocol¹⁵, Scheduled Transfer Protocol¹⁶, Xpress Transport Protocol¹⁷, Reliable Blast UDP¹⁸, Streaming Reliable Blast UDP¹⁹, Forward Error Corrected UDP²⁰, Reliable and Scalable Striping Protocol²¹, the Web100 project²², eXplicit Control Protocol²³, Network Block Transfer²⁴, the Simple Available Bandwidth Utilization Library²⁵, and Tsunami²⁶. See also³⁰.

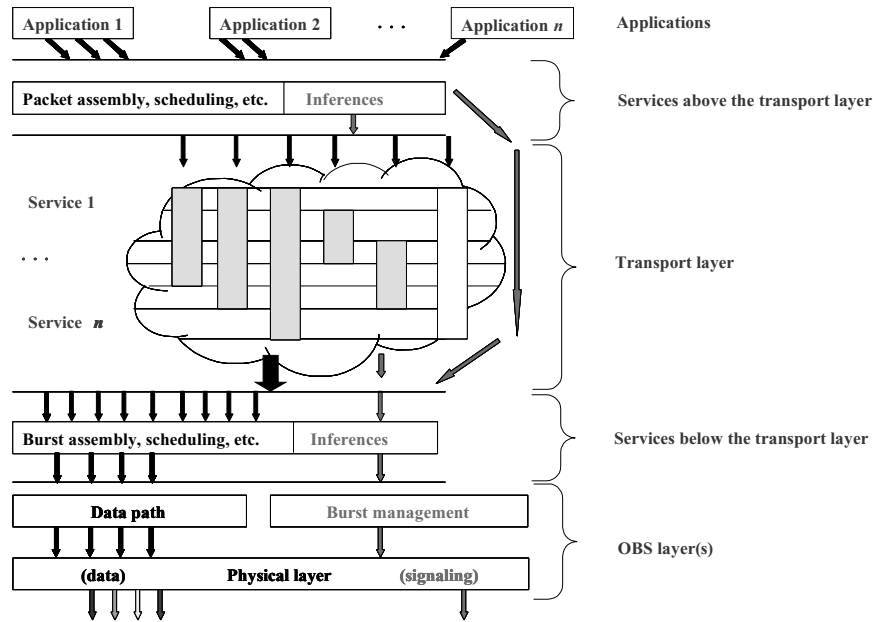


Figure 2. OBS reference model.

(2) Some applications may require additional packet- or datagram-oriented **services above the transport layer**. *E.g.*, an application may not prioritize traffic from its streams, so a shim that marks packets may be useful. The shim might also map stream requirements to transport layer services and/or OBS functions; *e.g.*, it may infer that a traffic stream is sensitive to latency and jitter but not to loss or severe error. This inference bounds the set of transport layer services and OBS functions required by the traffic stream. This is discussed briefly in Section 3.

(3) The OBS **transport layer** and transport protocol services, methods, mechanisms, and the (optional) adaptive control of services are discussed in Section 4.

(4) Every stream will require additional burst-level **services below the transport layer**. *E.g.*, an application may require absolute or quasi-absolute delay and jitter assurances for real-time streams. Note that some OBS mechanisms (burst assembly, burst scheduling, admission control, traffic shaping/policing, burst framing/formatting, deadline scheduling, other traffic conditioning) *may* reside here. This is discussed briefly in Section 5.

(5) The **OBS layers** comprise the OBS connection management, signaling, burst transmission, other OBS mechanisms, other OBS services, and the physical layer.

2.2 Design goals

Design goals include assumptions about the OBS network and the OTL, and requisite features:

(1) **Intelligent edge, simple core**. The OBS network and its network elements are functionally quite simple. Intelligence resides at the source and the destination, and/or within the ingress and egress OBS nodes. OBS core switches do not buffer bursts or condition traffic. If applications require service differentiation, it must be provided at the edge (with a few exceptions like preemption and experimental priority-based mechanisms). This contrasts with architectures such as IETF **diffserv** in which packets are marked at the source or edge, and service-level decisions are made in the core using packet markings and active queue management mechanisms.

(2) **Transport services**. The transport layer architecture *must* provide a set of standard transport services (comparable to those defined for ISO OSI layer 4), and a set of novel transport services to meet new requirements^{10-26,30}. The OTL only needs to provide transport layer services. As noted, ancillary transport services *may* be provided above and below the OTL to augment the OTL, and/or to support traditional network layer, link layer, or service quality functions.

(3) **Services vs. implementation of services**. There must be a clear distinction between transport services and the implementation of those services. *E.g.*, error-free delivery is a transport layer service. Reed-Solomon FEC is one mechanism for implementing an error-free transport service. Error detection (*e.g.*, CRC-32) and retransmission (*e.g.*, ARQ) is another (more robust) implementation mechanism. The architectural framework defines requisite services, but not how those services are to be implemented.

(4) **Transport protocol**. Generally accepted transport layer services are well known. However, a specific transport protocol uses only a small subset of services, and the subset has usually been tuned to perform well in a specific

control is requested, the OTL uses measurements to determine which error detection mechanism to use to guarantee QoS (Figure 4). In this example, measurements suggest that a long checksum (CRC-32) scheme is sufficient to guarantee the requested QoS. Future measurements may move the arrow left or right.

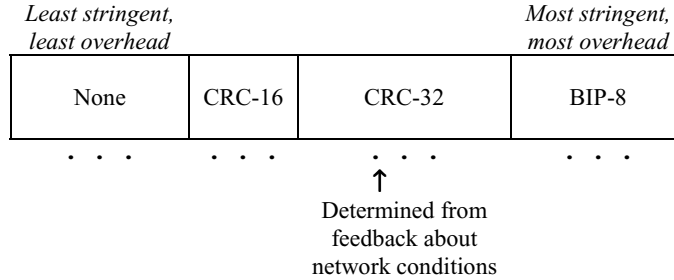


Figure 4. Adaptive control for an error detection service.

Adaptive control requires two components for each relevant service: (i) **monitoring ON/OFF**, and (ii) **control ON/OFF**. Adaptive control for service *i* requires both to be **ON**:

Control	Monitoring	Operation
control OFF	monitor OFF	No adaptive control for service <i>i</i>
control OFF	monitor ON	Typical monitoring for network management (alarms, exceptions, OAM, <i>etc.</i>) with no direct control action
control ON	monitor OFF	Locks implementation in the current control state
control ON	monitor ON	Adaptively controls service <i>i</i>

Table 1. Adaptive control state space.

(7) **Zero copy and kernel bypass**. Zero copy allows the OBS interface to transfer data to/from application memory without an intermediate copy to/from the OS kernel. Kernel bypass allows applications to issue commands directly to the OBS interface without an OS kernel call. This reduces the number of context switches required, and should greatly improve device and interface throughput. These techniques are intended for high performance, large-scale systems (typically multiprocessor clusters or servers) that transfer data in bulk over high bandwidth, low loss, low latency networks. They are particularly sensitive to memory bottlenecks as connection rates exceed processing and memory bandwidth. Copy avoidance significantly reduces I/O overhead and memory bandwidth. A fundamental design goal is to utilize zero copy and kernel bypass, and to build those features into the OTL and ancillary services.

The OTL conceptual building block approach (Figure 3) differs from other suites proposed for high-performance networks. *E.g.*, Quanta^{27,28} is a portfolio of individual, fully operational, standard and experimental transport protocols designed for high bandwidth and QoS-guaranteed flows. Applications explicitly or implicitly specify the transport protocol required for their stream(s). We believe the virtual suite approach to architectural design, when compared to other suites, is:

- Is more finely grained with respect to services.
- Is more amenable to hardware/software partitioning.
- Does not duplicate services; *e.g.*, several Quanta protocols provide reliable delivery, while the OTL has a single reliable delivery service (though it may have several implementations).
- Is more easily modified to support new services.
- May be more amenable to resource monitoring, adaptive control, and QoS provisioning.
- May be a realistic implementation option if coupled with a Quanta-like portfolio (Section 6).

3. SERVICES ABOVE THE TRANSPORT LAYER

3.1 Applications

Applications are at the top of the reference model (Figure 2) and have these characteristics:

- An OBS source (or edge node) hosts one or more concurrent applications.
- Each application generates one or more data ‘streams’. *E.g.*, a single virtual meeting application might have at least one audio stream, at least one video stream, a white board control stream, streams for one or more avatars (virtual participants), a background file transfer stream, a stream for directory services, a stream for instant messaging among participants, *etc.*
- Streams from the same application might have different QoS requirements. *E.g.*, audio and video streams are sensitive to latency and jitter; background file transfer is sensitive to loss and error; a directory service requires

reliable transport; instant messaging is low priority.

- Streams from the same application might have different destinations. *E.g.*, the directory services stream will go to a third party.
- Different applications will likely have different QoS requirements and destinations.

An application's streams might utilize a mix of transport connections. *E.g.*, a virtual meeting application might spawn M concurrent connections, or some mix of M connections and connectionless transfers, or a single connection in which packets from M streams are interleaved. An application might not support service differentiation. *E.g.*, some low-end video conferencing tools interleave packets from multiple streams onto one transport layer connection.

3.2 Other services above the transport layer

Applications might require (but lack) other services. A service (or 'shim') layer resides between the applications and the transport layer of the reference model (Figure 2). As the name implies, this is not a formal architectural layer, protocol layer, or adaptation layer. Service layers are not uncommon in protocol design. The IETF's Stream Control Transmission Protocol (SCTP) with Direct Data Placement utilizes a shim directly above SCTP. The shim allows messages to be placed in user workspace without (re-)assembly buffering, signals message boundary conditions and actions, bridges disparate technologies, and abstracts services from the transport protocol without affecting its behavior. The IETF's Blocks Extensible Exchange Protocol (BEEP) is another example of a service layer. BEEP is an application framework that supports the multiplexing of independent request/response streams over a single transport layer connection. The service layer has three functions:

(1) QoS assurance – the service layer might augment a naïve application by multiplexing and scheduling packets from that application; *e.g.*, a virtual meeting application that does not prioritize or differentiate traffic. It might provide other datagram services – prioritization, shaping, policing, scheduling via a timing table, *etc.*

(2) Mappings – the service layer might map applications' requirements onto OTL services or OBS dimensions; *e.g.*, an application might be sensitive to delay and jitter but not to loss or error. This requirement can be translated to OTL services and perhaps to OBS dimensions.

(3) QoS classes – the service layer can be used to define ATM-like application QoS classes; *e.g.*, a 'reliable low latency' class that is sensitive to delay, loss and error but not to jitter[†].

The service layer has the following characteristics:

- It is not required; it might be null.
- It might augment the OTL and/or services below the transport layer (Figure 2).
- It might serve one application by multiplexing, prioritizing, shaping, policing, ordering or scheduling packets from that application's stream(s).
- It might serve more than one application by multiplexing, ordering or scheduling packets from the applications' streams.
- It might act as a virtual source or a proxy source.

4. TRANSPORT LAYER

As noted, the transport layer provides functions or services (rows in Figure 3) to traffic streams (columns in Figure 3). The leftmost traffic stream in Figure 3 requires two services – ordered delivery and expedited transfer. In general, a stream requires s services ($s \in \{0, 1, 2, \dots, n\}$). The transport layer has the following characteristics:

- It is required for OBS networks; it is **not** a null layer (although s might be zero).
- It works in concert with services above and/or below the transport layer.
- Some of it can be implemented in hardware.
- It need not vary with different OBS implementations (*e.g.*, LAN, WAN). Ideally, it should not vary with different OBS implementations.

4.1 Nomenclature – services, methods, mechanisms

Transport layer functions require a precise terminology and taxonomy (Figure 5):

- The OTL provides n transport layer **services**. *E.g.*, sequenced delivery is a service.
- A service i can be provided by one of m service **methods**. *E.g.*, a sequenced delivery service can be provided by at least two different methods – adding sequence numbers to the transport layer wrapper and having the transport layer reorder payloads at the destination, or using a OBS pinned route that guarantees payloads arrive in the order they

[†] This service class approach is similar in concept to the ITU-T's ATM service classes specified in ITU-T I.356, I.362 (discontinued), and I.371.

were sent.

- A service method j can be implemented by one of p service method **mechanisms**. As noted, an error detection service can be implemented via short or long checksums (CRC-16 or CRC-32), or a more robust bit-interleaved parity checking mechanism (e.g., BIP-8).
- A service method (i, j) may be amenable to **adaptive control** such that the service method's mechanism k changes in response to monitored conditions; e.g., use BIP-8 on noisy links.
- Each service method that is amenable to adaptive control has a **monitoring** component with a parameter **monitor** (i, j) , and a **control** component with a parameter **control** (i, j) . Parameters indicate whether the component are **ON** or **OFF**.

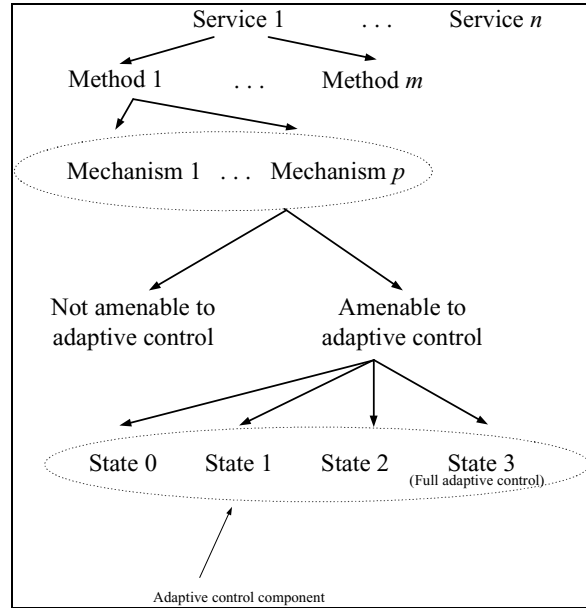


Figure 5. Transport layer nomenclature and taxonomy.

Adaptive control induces the four states at the bottom of Figure 5.

4.2 Services

A representative set of transport services for OBS networks is provided in Table 2. This is not an exhaustive list. As noted, the set of services is open-ended; some will be provided by the OTL, some will be provided by the OTL in concert with the service layers above and below the OTL, and some will be mapped directly onto OBS dimensions.

Category	Service	Amenable to adaptive control
Connection management services	Connection-oriented transport	No
	Connection termination	No
	Restoration and recovery	No
	Inactivity control	No
	Mapping application attributes	Not applicable
	Mapping OBS attributes	Not applicable
	Connection ID	No
	Duplex connections	Not applicable
	Multiplexed connections	No
Connectionless services	Inverse multiplexed connections	No
	Connectionless transport	No
	Mapping application attributes	Not applicable
Unicast data transfer services	Mapping OBS attributes	Not applicable
	Flow control	Yes
	Rate control	Yes
	Push	No
	OTL wrapper	Not applicable
	Segmentation, reassembly, framing, formatting	Depends

Category	Service	Amenable to adaptive control
Multicast data transfer services	(Similar to unicast services)	Yes
Services that guarantee QoS	Sequenced delivery	No
	Loss detection	Yes
	Duplicate detection	Yes
	Error detection	Yes
	Retransmission	Depends
	Error correction	Yes
	Error control	Yes
	Bandwidth control	Yes
	Priority, preemption, expedited delivery	Depends
Status reporting services	Error logging	Yes
	Urgent indication	No
	Exception reporting	No
Interfaces to services above/below the OTL	API, RPC, message passing, DMA, <i>etc.</i>	No
Hybrid services	New service composed of two or more other services	Yes

Table 2. A representative set of OTL services.

4.3 Adaptive control

As noted, a monitoring component and a control component are required for each service that is subject to adaptive control. The monitoring scheme depends on the capabilities of the monitoring component. It may support different sampling disciplines (periodic, random, stratified), aggregations (events over threshold, cumulative events), triggers, *etc.*

Feedback controllers are an important part of the *implementation* of a service that supports adaptive control, although they are not a part of the OTL *per se*. A full implementation might have a suite of controllers and use different ones for different transport layer services; *e.g.*:

- A two state ('bang bang') controller. These monitor an input's state relative to one or two thresholds, and turn the control element fully up or down. They provide coarse control and are easily implemented in hardware.
- A state and δ -state controller. These monitor an input's state relative to 3-4 thresholds, estimate the rate of change (δ), and derive a proportional control action. They are more finely grained, and are easily implemented in hardware.
- A proportional integral derivative (PID) controller. These compute a control action based on the input state and feedback gain multipliers that control stability, error and response. They are powerful but can be difficult to tune.
- A fuzzy controller. These adjust control actions based on imprecise definitions of performance thresholds, control points, and control point tolerances. Empirical values are convolved to obtain a proportional control action.
- A neural controller. These are often used to refine and improve the operational performance of fuzzy controllers. They require training exemplars and a training regime, and may be implemented in hardware.
- A sequential decision stochastic controller²⁹. These are reward/loss-driven controllers that seek a sequence of actions to optimize system performance (minimize cost or maximize reward) over some decision-making horizon. They typically use dynamic programming or other non-linear optimization techniques, and are less likely to be implemented in hardware.

Adaptive control also requires mechanisms to detect, report, and react to situations arising when the requested QoS can no longer be guaranteed under any service method mechanism.

5. SERVICES BELOW THE TRANSPORT LAYER

5.1 Other services below the transport layer

There is a second service layer between the OTL and OBS layers (Figure 2). This layer will typically provide some OBS functions, including burst assembly and burst scheduling. Like the service layer above the OTL, this is not a formal architectural layer, protocol layer, or adaptation layer. OBS networks do not have a network layer at intermediate switches to handle service differentiation and assure QoS, so it must be provided at the edge of the network; *viz.*:

- Some applications require absolute, deterministic, ATM-like QoS delay or jitter bounds.
- Some applications require quasi-absolute QoS assurances comparable to IETF **diffserv** pseudowire (PW) or expedited forwarding (EF) per-hop behaviors (PHB).
- Some applications require relative QoS assurances comparable to the **diffserv** assured forwarding (AF) PHB.

The service layer has the following characteristics:

- It is required for OBS sources or edge nodes; it is not null.
- It might augment the OTL and/or services above the transport layer.

- It typically serves all of the outbound transport-layer connections.
- Connections are concurrent, they might have different QoS requirements, and they almost certainly have different destinations.
- Some services can be implemented in hardware.
- It will likely vary with different OBS implementations (*e.g.*, LAN, WAN). As noted, the OTL need not vary with different OBS implementations.

The OBS components, if provided at this conceptual layer, may include:

- Multiple tributary inputs, where an input may be a transport-layer connection.
- A classifier, which is usually a function of the destination and the QoS class and priority.
- One or more burst assemblers.
- At least one class-based queue per destination, *i.e.*, per-destination per-class queuing.
- One or more burst schedulers (work conserving or non-work conserving).

Some implementations may provide additional components: admission control, adaptive burst assembly, forecasting, burst shaping, traffic conditioning, services related to burst size distributions, policing, rate control (leaky or token bucket).

5.2 OBS layer(s)

The OBS layer(s) reside at the bottom of the reference model (Figure 2). They provide OBS connection management services and physical layer services and functions: signaling, transmitting and receiving optical bursts, O/E conversion of control packets, framing bursts, generating offsets, generating burst control packets, wavelength multiplexing and demultiplexing and assignment, resource reservations (*e.g.*, TAW, TAG variants), *etc.*

6. IMPLEMENTATION

As noted, implementation details are outside the scope of the architectural framework. However, some general comments and guidelines are appropriate, and are provided in this section. There are at least four ways to implement the OTL:

(1) As a single **monolithic** transport protocol containing some/many/most/all of the services described in Section 4. In theory, a monolithic transport protocol is the union of every required transport protocol service embedded in an operating system-like kernel (Figure 6a). One would use options, parameters, calls, *etc.* to define the protocol's behavior. The advantages are:

- Service duplication – an implementation would not duplicate services.
- Control – tightly integrated services might be more amenable to resource monitoring and adaptive control.

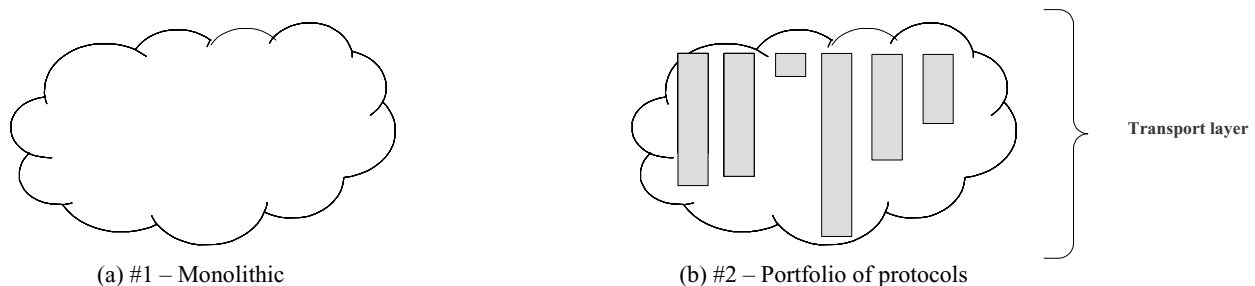


Figure 6. (a) Monolithic, and (b) portfolio of protocols implementation options.

- Interfaces – an implementation would have a single standard interface to services/layers above and below the transport layer.
- Partitionability – an implementation might be more amenable to hardware/software partitioning than most other alternatives.

A monolithic transport protocol is probably unrealistic for the following reasons:

- Size – an implementation (a transport protocol kernel or '**tpkernel**') would be quite large, perhaps as large as a small operating system kernel.
- Environment – a different implementation would almost certainly be required for each environment (LAN, WAN, tributary type, *etc.*).
- Performance – an implementation might be difficult to tune. The **tpkernel** also runs counter to kernel bypass and other performance enhancement concepts.

- Extensibility – adding new services would require `tpkernel` modifications and integration.
- Flexibility – an implementation may not easily support unforeseen service combinations.
- Granularity – an implementation would not likely support as many service methods or service method mechanisms as some other alternatives.
- Acceptance – an implementation’s acceptance by the industry would likely depend on how the kernel was defined. The design issues and tradeoffs are similar to those proposed for operating systems (monolithic, kernel, micro-kernel, exo-kernel, *etc.*)

(2) As a **portfolio** of individual, fully operational, production-grade transport protocols that collectively provide every requisite service. *E.g.*, the OTL could be a set of N fully operational transport protocols sandwiched between applications and the OBS layers (Figure 6b). The set collectively provides every requisite service. One would explicitly or implicitly invoke one transport protocol at a time from the set. (This is very similar to the Quanta approach^{27,28,29}.) The portfolio could include any number of standard and experimental transport protocols. In Figure 6b, each ‘silo’ represents one fully operational transport protocol. The advantages are:

- Size – the set would be the size of the N individual transport protocols plus the wrapper, but modular (unlike the monolithic protocol), so size is less an issue than with other alternatives.
- Environment – a different set would almost certainly be required for each environment, but building a different set simply requires swapping set elements in much the same way that TCP variants are swapped when satellite links are utilized.
- Performance – an implementation’s performance and tunability depends on the individual protocols in the set. However, most production-grade transport protocols are peak performers in the environment for which they are intended.
- Extensibility – new services are added by adding protocols to the set.
- Acceptance – an implementation would almost certainly be accepted by the industry.

The disadvantages are:

- Service duplication – an implementation (*i.e.*, a set of N transport layer protocols) would duplicate some services nearly N -fold.
- Control – services would not be amenable to resource monitoring and adaptive control because each individual protocol would have to be modified.
- Interfaces – an implementation would have many different interfaces to services/layers above and below the transport layer; some type of middleware wrapper will be required.
- Flexibility – an implementation would not likely support unforeseen service combinations.
- Partitionability – hardware/software partitioning would likely require modifications to each individual protocol.
- Granularity – an implementation would not likely support many different service methods or service method mechanisms unless the set were quite large.

(3) As a set of individual **services** that are combined (when required) to form a virtual transport protocol instance. *E.g.*, the OTL is a set of N standard transport *services* (Figure 7a.) The set includes the services outlined in Section 4. Services can be combined more easily than with other alternatives. One would use options, parameters, calls, *etc.* to assemble a virtual transport protocol by explicitly or implicitly invoking one or more services from the set. In Figure 7a, each ‘silo’ represents one virtual transport protocol instance comprising some subset of OTL services.

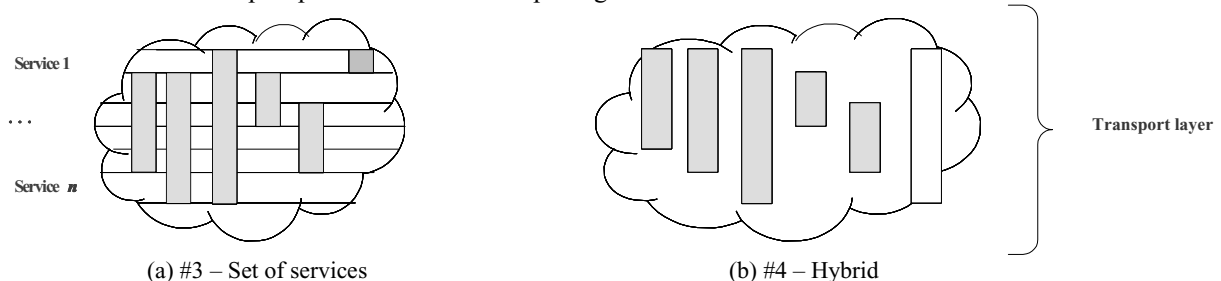


Figure 7. (a) Set of services, and (b) hybrid implementation options.

The advantages are:

- Environment – one set can be used for every environment by changing only the services provided below the transport layer.
- Extensibility – by definition, new services are easily added.

- Service duplication – by definition, an implementation would not duplicate services.
- Control – by definition, services are amenable to resource monitoring and adaptive control.
- Interfaces – by definition, an implementation has a single interface to services/layers above and below the transport layer.
- Flexibility – by definition, an implementation supports any logical combination of services, including unforeseen service combinations.
- Partitionability – hardware/software partitioning is a fundamental design goal.
- Granularity – by definition, an implementation can support multiple service methods and multiple service method mechanisms.

The disadvantages are few in number, but significant:

- Size – an implementation would be quite large, perhaps larger than the monolithic alternative. It would almost certainly be the most complex implementation because of the ‘building block’ design of the services.
- Performance – an implementation’s performance and tunability would depend on how individual services interoperate and interact. It is unlikely that one implementation of a service (*e.g.*, retransmission) would be optimal for every application, so adaptive control with multiple service methods and service method mechanisms would be required. Performance and performance optimization would be major concerns.
- Acceptance – this approach is unlikely to be accepted by the industry.

(4) Some **hybrid** of alternatives #2 and #3. *E.g.*, the OTL is a portfolio of $N+1$ protocols: N fully operational, production-grade, standard or experimental transport protocols, plus the virtual protocol architecture (Figure 7b). In Figure 7b, each shaded silo represents a fully operational transport protocol (#2), and the unshaded silo represents the virtual protocol architecture (#3).

The major advantage of this approach is the evolutionary path it provides. The virtual protocol architecture could be implemented service by service, and its perceived disadvantages – performance, size, complexity – would be measurable at an early stage. The advantages – extendibility, control, interfaces, flexibility, partitionability, granularity – would also be more easily evaluated vis-à-vis production-grade protocols. *E.g.*, one could directly compare Reliable Blast UDP¹⁸ with a reliable connectionless bulk transfer transport protocol instance. The likely result is that virtual instances would not perform as well as widely-implemented and exquisitely tuned production protocols, but *may* perform as well as experimental protocols.

Any comparison of the four implementation options is subjective. Acceptance, performance and size probably should account for at least half of any realistic weighing regime.

7. CONCLUSIONS

An important conclusion is that the architectural design framework is not necessarily a blueprint for a realistic OTL implementation.

7.1 Design issues

Tradeoffs. A challenge is to ensure that the OTL is clearly defined, nonproprietary, secure (supporting authentication, authorization, integrity, privacy), fast and efficient, robust (to system failures), and easily extendible (technically, architecturally). Some of these goals are in competition with others.

Service set. Another challenge is selecting a ‘best’ set of (nearly) orthogonal transport protocol services from the scores of standard and experimental transport protocols proposed over the last two decades. Each service must be individually enhanced to support OBS signaling, optical networks, hardware/software partitioning, adaptive control, and hardware acceleration.

Boundaries. In theory, services typically provided by the application, and services typically residing above or below the transport layer will not be a part of the OTL. In practice, some non-transport services *may* be added to an implementation of the OTL. If so, those services must be clearly distinguished from other transport protocol services.

OBS focus. The OTL is closely coupled with optical networks in general and OBS in particular. It need not be overly flexible to support transport over other architectures (*e.g.*, wireless). As noted, if flexibility is required, it should be provided by the conceptual service layer below the OTL, the OBS layer, by a MAC layer, and/or by the PHY layer.

Interfaces. Precise definitions are required for the interface between applications and services above the OTL, and the OTL itself; for the interface between the OTL and services below the OTL, including the OBS burst assembler and scheduler; and where OBS functions reside – software-based functions likely reside in the ‘services below the OTL’ conceptual layer; hardware-based functions likely reside in the OBS layer(s) proper.

7.2 Implementation issues

Approach. The approach that one might use to build a proof-of-concept OTL has not been determined. Alternatives 1 and 3 are least likely. Alternative 2 is close to the Quanta approach. Note that #3 is closest to the architectural design framework. Alternative 4 is an attempt to combine the strengths of #2 and #3 while minimizing the limitations of each.

Effort. Full implementations of production-grade transport protocols typically require several person-years of effort (design, implementation, laboratory testing, testbed deployment, documentation, performance tuning, *etc.*). Alternative 3 would require significantly more effort. Alternative 2 requires the least, but has limitations for OBS networks as noted.

Reuse. An advantage of building from an orthogonal services design (#3 and parts of #4) is that some (and perhaps most) of the requisite transport-layer functionality can be inherited from production-grade protocol components. It may require significantly less debugging and testing, and provide significantly more testbed functionality than building the OTL in some other way. It will also require access to quality source code.

ACKNOWLEDGMENT

This work was supported by NASA through a grant with the MCNC Virtual Collaborative Center (NAG2-1467). The authors gratefully acknowledge the contributions of Gigi Karmous-Edwards (MCNC RDI), Steve Thorpe and Kevin Gamiel (MCNC), Linden Mercer (US Naval Research Laboratory), and Ray McFarland to this research effort.

REFERENCES

1. Yao S. *et al.*, "Advances in photonic packet switching: an overview", *IEEE Communications*, **38**(2), 84-94 (2000).
2. Turner J., "Terabit burst switching", *J. High Speed Networks*, **8**(1), 3-16 (1999).
3. Qiao C. and M. Yoo, "Optical burst switching (OBS) – a new paradigm for an optical Internet", *J. High Speed Networks*, **8**(1), 69-84 (1999).
4. Xu L., H. Perros and G. Rouskas, "Techniques for optical packet switching and optical burst switching", *IEEE Communications*, **39**(1), 136-142 (2001).
5. Verma S. *et al.* "Optical burst switching: a viable solution for terabit IP backbone." *IEEE Network Mag.*, November 2000. 6 pp.
6. Vokkarane V. and J. Jue. "Prioritized routing and burst segmentation for QoS in optical burst-switched networks." *Proc. OFC Conf. 2002*. 3 pp.
7. Vokkarane V. *et al.* "Burst segmentation: an approach for reducing packet loss in optical burst switched networks." *Proc. IEEE ICC. 2002*. 5 pp.
8. "High-Performance Networks for High-Impact Science", *Report of the August 13-15 2002 Workshop on High-Performance Network Planning*, 116 pp., http://doecollaboratory.pnl.gov/meetings/hpnpw/finalreport/high-performance_networks.pdf.
9. Bunn J. *et al.*, "Ultrascale Network Protocols for Computing and Science in the 21st Century", White paper to US Department of Energy's USS initiative, September 2002.
10. High Speed TCP, 'draft-floyd-tcp-highspeed-02.txt', IETF, February 2003, work in progress.
11. Caltech FAST, <http://netlab.caltech.edu/FAST/overview.html>.
12. Various parallel TCPs, *e.g.*, Hsieh, H-Y. and R. Sivakumar, "pTCP: An end-to-end transport layer protocol for striped connections", *Proc. 10th IEEE Intl. Conf. Network Protocols*, November 2002, Paris.
13. Grid TCP, <http://netlab.caltech.edu/FAST/meetings/2002july/GridTCP.ppt>.
14. TCP Westwood, <http://www.cs.ucla.edu/NRL/hpi/tcpw/>.
15. Stream Control Transmission Protocol, IETF RFC 2960, "Stream Control Transmission Protocol", October 2000.
16. Scheduled Transfer Protocol, ANSI standard transport protocol (ANSI INCITS 337-2000 and 343-2001).
17. Xpress Transport Protocol, <http://www.ca.sandia.gov/xtp/>.
18. He, Eric *et al.* "Reliable Blast UDP: Predictable High Performance Bulk Data Transfer", *Proc. IEEE Cluster Comput. 2002*, Chicago, September 2002, 8 pp.
19. Streaming Reliable Blast UDP, Leigh, J., June 2001, unpublished.
20. Forward Error Corrected UDP, in Leigh, J. *et al.*, "Adaptive networking for tele-immersion", *Proc. Immersive Projection Technology/ Eurographics Virtual Envir. Workshop (IPT/EGVE)*, Stuttgart, May 2001.
21. Adishesu, H. *et al.* "A reliable and scalable striping protocol", *Proc. ACM SIGCOMM*, **26**(4), October 1996, 11 pp.
22. The Web100 project, <http://www.web100.org/>.
23. eXplicit Control Protocol, Katabi D. *et al.*, "Congestion Control for High Bandwidth Delay Product Networks." *Proc. ACM SIGCOMM '02*, August 2002.
24. Network Block Transfer, IETF RFCs 998 and 1030, 1987.
25. The Simple Available Bandwidth Utilization Library (SABUL), <http://www.dataspaceweb.net/sabul.htm>.
26. Tsunami, <http://www.indiana.edu/~anml/>.
27. Leigh, J. *et al.*, *op. cit.*
28. Quanta project, UIC EVL, <http://www.evl.uic.edu/cavern/teranode/quanta.html>.
29. He, Eric *et al.* "Quanta: a toolkit for high performance data delivery over photonic networks", in *Future Generation Computer Systems*, Elsevier Science Press (to appear).
30. "Survey of Transport Protocols Other than Standard TCP", <http://www.evl.uic.edu/eric/atp/>.