XMIT Design Notes

alpha9 - 2013-08-27

This document complements the XMIT specification by explaining design goals and potential applicability of XMIT.

Contents

1 References

XMIT	http://blinkprotocol.org/spec/XmitSpec-alpha9.pdf
GITHUB	https://github.com/pantor-engineering/xmit
BLINK	http://blinkprotocol.org
RFC4122	http://tools.ietf.org/html/rfc4122
RFC4689	http://tools.ietf.org/html/rfc4689
RFC5218	http://tools.ietf.org/html/rfc5218
RFC5405	http://tools.ietf.org/html/rfc5405
HOURGLASS	http://tools.ietf.org/html/draft-tschofenig-hourglass-00
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
IPC	Inter-Process Communication
RPC	Remote Procedure Call
QUIC	http://en.wikipedia.org/wiki/QUIC

2 Introduction

The purpose of this document is to give some context to XMIT design decisions and explain some tacit properties of XMIT. To a large extent, XMIT builds upon existing best practices.

XMIT was designed to extract and generalize upon the best characteristics of existing session layer protocols that support electronic trading in the capital markets.

Among key requirements in this space is to support asynchronous peer-to-peer and one-to-many communications in the data center and in wide area networks. XMIT leaves to the user to synthesize any other more involved communications pattern from these two basic patterns. Synchronous request and response or [RPC] is one example of a pattern that is easy to build from the XMIT peer to peer primitive.

XMIT was designed to make it easy for applications to take advantage of UDP and its better than TCP realtime characteristics. An XMIT implementation should be able to present a unified interface to the application, regardless of which transport protocol that is used.

XMIT is a protocol and not a programming interface with a protocol inside.

Another design goal has been to avoid weak aspects of existing protocols. Specifically, the FIX session layer is a widely used standard that has a significant performance overhead. The FIX session is improperly layered, confusing application message flow with the session control flow, and has unfortunate lifetime semantics. A simple error in carrying out a standard operational procedure may silently void the guarantee that an operation is only carried out once. XMIT and the FIX session layer are significantly different.

XMIT is meant to be more generic than the purpose built protocols that its design draws upon. This would allow XMIT to be more widely applicable, at a minimal increase in complexity or processing overhead.

Some of the existing protocols that XMIT draws upon, or relates to, are described in Section 9 (page 3) .

Sample application patterns using XMIT can be found in Section 10 (page 3) .

3 Transfer Encoding

XMIT explicitly does not specify a transfer encoding. The data center requirement includes in-process and in-server communications and high performance networking. In an environment sometimes constrained by processing power instead of bandwidth, the optimal transfer encoding may be different than in other scenarios. This applies to XMIT as well as other protocol layers. To get some background on transfer encoding independence that is application transparent, please see the design of [BLINK].

XMIT shares a design philosophy with Blink, in that the same message protocol should be used across layers where possible. The session and the application layers should be independent of a particular transfer encoding. By first liberating the layers in the stack, it becomes possible to select a particular encoding across the layers for a given message exchange. This enables vertical access through the message layers, i.e., integrated layer processing.

4 Session Identifier Allocation

XMIT relies on *UUID* generation to create unique identifiers in a distributed system, without requiring any communication in the allocation of identifiers. It may take a leap of faith to accept that this allocation system is safe. If you feel uncomfortable with this method, please bear in mind that this is a practice established in other settings long before XMIT was created.

5 Implementation Independency

XMIT is simple to implement and is not tied to a reference implementation or platform. It was first implemented in *node.js*, using a foreign function interface towards a Blink *C++* implementation. The second and third XMIT implementations were written in *C++* and *Java*, respectively. The Java version is open sourced and can be found at [GITHUB]. It is based on the Pantor Engineering open source Blink implementation that is available at the same location.

Some messaging technologies are defined in terms of a common programming interface that provides easy to use access to message patterns, with bindings in multiple programming languages. There is sometimes a stronger emphasis on the programming interface than on the underlying session level protocol being used. The specification of XMIT is strictly protocol oriented.

6 Tacit Properties

The purpose of this chapter is to highlight or reiterate properties of XMIT, some of which are otherwise tacit.

- XMIT performs optimally over an unreliable datagram transport (UDP), if there is sufficient bandwidth. If the network is sometimes congested, the optimal transport is to use TCP that is designed to handle such scenarios.
- XMIT enables the flow in each direction to be either sequenced or unsequenced, independently of the flow type used in the other direction.
- XMIT allows sessions to be multiplexed over a single transport session or in a datagram. The latter enables fine-grained topics of data to be multicast using coarse-grained multicast group addressing.
- XMIT does not provide flow control or rate pacing.
- This version of XMIT does not include acknowledgment of receipt; such acknowledgment could potentially allow a producer to prune data it has created.
- XMIT provides transparency into and control over the behavior in congestion and packet loss scenarios, where a reliable but opaque transport would introduce latency and unconditional retransmisson.

7 Transport Protocols

A key aspect of the design is to take advantage of the nuances in communicating over UDP or TCP in the best way possible.

UDP provides better realtime and latency characteristics. UDP provides control to the user where the use of TCP would hide and translate network problems into unconditional retransmission and resulting delays. A UDP based stack is simpler and therefore easier to implement in a new, different way for better performance.

On the server side, UDP may significantly reduce the per-session state footprint compared to TCP. A processor is typically constrained by its caches, so a reduced footprint allows more sessions to be handled per processor core.

XMIT and TCP partially overlaps but XMIT does not provide some more complex TCP features that are often desirable. Using TCP with XMIT adds congestion control and flow control while the reliability and ordering provided by TCP overlaps with similar functionality in XMIT.

Traditionally, TCP has been used as the safe or default choice, despite UDP having been a potentially better match. XMIT is meant to make it easier to use UDP exactly where it makes sense to.

TCP is more capable but also more complex, opaque, and difficult to control. For a realtime application, TCP provides little value over UDP in the data center, or along a network path that has predictable bandwidth characteristics, TCP does provide the congestion control mechanism required to optimize the utility of a network path, where a naive UDP implemention might under utilize the available bandwidth for fear of packet loss.

It could be attractive to create a hybrid of TCP and UDP, like combining the congestion control of TCP with the application option to intelligently drop data that would get stale and irrelevant if queueing kicks in. There would be little value in trying to reinvent the algorithms used by TCP in XMIT, unless it could be done at a low complexity penalty. If anything, XMIT should leverage emerging alternatives to TCP. See [QUIC] for a recent example on an intermediary network protocol that leverages existing UDP (and TCP obviously) centric network infrastructure to provide TCP like features.

8 Performance Notes

8.1 Efficient Session Mapping

The UDP or TCP addressing tuple identifies one established XMIT session unless multiplexing is being used. This tuple can be bound to a file descriptor in the sockets programming model. An implementation of XMIT can therefore use a low cost lookup from the file descriptor to the only XMIT session for the file descriptor.

8.2 Efficient Topic Filtering

The receiver can perform topic filtering on the XMIT level, without involving the application. This means that filtering can be offloaded with ease, for instance to an FPGA-based device.

9 Other Session Layers

9.1 FIX

Model wise, the FIX session layer most closely resembles peer to peer XMIT with the use of the sequenced mode in both directions. FIX and XMIT differs significantly in that the application layer is decoupled from the session layer in XMIT.

In FIX, all messages from both layers use the same sequencing. The FIX model is seemingly simple, but results in unwanted behavior. A producer of a FIX application flow cannot determine its own sequence. Instead, the sequence will depend on the way it is transmitted to the counterparty. For instance, if the session is logged off and logged on again, the relogon message exchange will allocate an outgoing sequence number. This means that no application message will have a predetermined sequence number, but one that depends on events at the session level.

If the exact same application flow is shared among multiple parties using a FIX session each, the producer cannot use a shared message sequence. If one of the parties log off and on, the sessions carrying the same data will have a different sequence numbering. This property of the FIX session layer limits the scalability of an market data application. It is also more difficult to replicate the exact sequence in a separate copy of the producer, to create a highly available service.

The tight coupling of layers in FIX means that there is a need for a gap fill mechanism to weed out session messages during recovery. XMIT does not have this need.

To deal with the problems in the FIX session layer, it is common to use FIX application sequencing and turn off the sequencing at the FIX session level. These fields are defined at the FIX application level but there is no cleaned up FIX session layer that integrates the new mechanism. At the time of writing this text, there is no wide adoption of a reformed FIX session and application layer that would allow an implementor of a general platform to avoid avoid dealing with the shortcomings of the original FIX session layer.

9.2 NASDAQ Soup and UFO

Soup is similar to the XMIT peer to peer mode, with an unsequenced flow going from the client to the venue, and using a sequenced flow in the other direction. The UDP based UFO protocol maps to XMIT in the same way as Soup. XMIT unifies the models used by Soup and UFO.

9.3 NASDAQ Mold

Mold UDP is similar to the XMIT one to many mode. XMIT adds the option to create topics, logical sessions. The model for recovering messages in Mold resembles the peer to peer XMIT recovery model. In Mold UDP, there is an implicit convention that a datagram of payload is sent in each retransmission round, automatically imposing rate pacing by making the client send a new rerequest message for each incoming datagram. In XMIT, the rate pacing has been made more explicit to support multiple datagrams to be in transit at a time, and to better function over TCP that has no datagram boundaries. In XMIT, there is a *RetransmitRequestResponse* message that instructs the counterparty that it will need to issue a new resend request.

10 Applications

10.1 FIX/XMIT

To layer the FIX application messages on to XMIT, XMIT would be used in a bidirected sequenced mode. XMIT enables FIX to be transported over UDP in addition to TCP. XMIT introduces session layer decoupling.

A challenge with the lack of layering in the FIX stack is that it makes it more difficult to migrate off the existing FIX session layer to an alternate session protocol.

10.2 High Performance Electronic Trading

Order and quote routing messaging matches the use of the unsequenced mode, as proven by the popularity of OUCH/Soup. It makes good sense to blend the application and session layer, not by coupling them like in FIX, but by moving responsibilites to the application layer. To make an operation on an order (or a quote) idempotent, a unqiue application level key is coupled to each operation.

The unsequenced mode does not provide automatic transmission. This is a feature and not a limitation as the application benefits from controlling the behavior when a message would need to be resent. An order that is not processed immediately is likely to be stale by the time it can be reprocessed.

10.3 Multicast Market Data

One important consideration in multicasting market data is to weigh the different filter preferences among clients against the cost to produce a fine grained selection of logical channels.

By defining fewer logical channels with more content per channel, messages that has been generated at an instant across topics can be combined into fewer datagrams, thereby reducing network header overhead. Also the cost to setup and maintain IP Multicast groups is held back by having fewer of them.

By using coarse grouping, network bandwidth will obviously be wasted on its way to clients and also add filtering processing overhead. If topic addressing is made in application layer terms, the input filtering has to be applied at the application level. It may be costly processing wise for the client to carry out the filtering at the application level, after having first decoded and passed all of the data through the stack.

XMIT addresses the application level filter problem by allowing clients to perform filtering already in the session layer, without involving the application.

The use of one session for each topic or logical channel, in combination with the use of multiplexing, allows XMIT to decouple topic definition at the application level from multicast addressing at the networking level. A large set of topics can be multiplexed over a smaller set of multicast groups as there is one session per topic and the producer is free to define the granularity of a topic.

10.4 Subscribed Market Data

An option to multicasting market data is to perform filtering at the head end, and to allow clients to subscribe to a subset of the data that is available to them. XMIT/TCP matches the traditional method of using TCP to transport subscribed market data and performing programmed multicast in software or using specialized hardware appliances. A potential benefit of using XMIT/UDP is that advanced data producers may be better able to control end to end rate pacing with the more transparent UDP stack. Using XMIT/UDP may also allow a producer to create a programmed multicast mechanism where the UDP stack allows data copying to be made in a more lean way across sessions, than is possible with the more complex TCP stack.