Checks on Connection Operations

The destination port and key must be verified.

Checks on Transfer Operations

The destination port, key, and T-Id must be verified.

Checks on block operations

The destination port, key, T-Id, and B_num must be verified.
For Fetchop and Get verify port, key, and F-id/G-id, respectively.

Checks on STU operations (i.e. Data messages).

The destination port, key, T-Id, B_num and STU_num must be verified.
Note that for Data messages in reply to a Fetchop or Get the F-id/G-id, respectively, replaces T-Id and B_num.

10.0 Appendix A: References

[94Stevens] - **TCP/IP Illustrated**, Volume 1, published 1994...

RFC 793 defines a Maximum Segment Lifetime (MSL) of 2 minutes. Common implementations are 30 seconds to 120 seconds. Restricts reuse of client and server ports, where C-IP:C-Port:S-IP:S-Port cannot be reused for 2*MSL. This is used for the "Quiet Time" at the end of a connection.

RFC 1323 Comments (but does not justify) in section 4.2.2 that the worst case MSL is 255 seconds, but claims that this constraint should be larger.

RFC 1323 Defines the use of a time stamp algorithm to prevent aliasing of TCP sequence numbers within a connection called Protect Against Wrapped Sequence Numbers (PAWS). One feature of the algorithm if 1 msec clocks are used is MSL is 24.8 days (section 4.2.2). The relaxed constraint is needed to prevent sequence number wrap on idle connections (where the time stamp is not being updated, but then data transfer starts again). If the timestamp wraps, all transfers are locked out until a wrap occurs again.

Because a remote address is not included in the connection triplet, the remote port is not unique to a host. Thus it is possible for two hosts connecting to the same destination port with the same local key to alias.

An example helps. Assume hosts A and B wish to connect to server S, where each wishes to establish an infinite Write sequence. Both A and B are using the same local port number. A establishes a connection at port 1457, with Akey. Sometime later, B establishes a connection to S on port 1457. Assume that enough time has elapsed so the key has wrapped such that Akey is assigned to B's connection. It is now possible for an RTS to be aliased to either connection.

This is easily solved if we restrict a destination port to never use the same key for two simultaneously open connections, or we disallow 2 simultaneous connections to the same destination port.

7.2 Proposed Change

If a host allows multiple remote hosts to connect to the same port, the local host must control key allocation such that if a second RC is received on a specific port, the returned key must be unique for that destination port.

8.0 Protecting against Programmatic Errors

8.1 Problem Definition

It is desirable that if an implementation of ST on host A is incorrect that it not corrupt (alias to) an existing connection between hosts B and S. The ST specification requires the destination to verify the key before the operation is accepted. This algorithm is sufficient only if key's are unique to a destination host. If the key's have wrapped, this is insufficient.

If the constraint was modified to not accept a packet if the key:d_port is not correct (and the Proposed changes above are approved), then this aliasing is impossible.

8.2 Proposed change

No other changes required, because enforcement of checks defined in this document provide a super-set of the above protection.

9.0 Summary

This paper outlined restrictions for allocating STU_Num, Max_STU's, Max_Block, STU_Num, Blk_Num, T-id, and key's. To enforce this protection, the recipient of a message must check ***every message*** for valid id’s. The number of id’s to check increases per layer, where an STU has the most checks required.

Local-Port : Local-key : Remote-Port
16 bits : 32 bits : 16 bits = 64 bits

Because the connection triplet is included in every ST message, for connection aliasing to occur the connection triplet must alias.

The key is initialized to any value after a reboot, and currently has no restrictions for key allocation. Because of this, there is no protection against key aliasing, and consequently connection aliasing.

Two methods are considered here:

1) Implementing a Quiet Time after connection teardown for at least 2*MSL.
2) Restricting the key to monotonically increase for each new connection.

If a Quiet Time is implemented, it means that from when the last connection was torn down the triplet can not be reused for a period of time. This could be implemented as an additional state in the connection teardown sequence, similar to the TCP state TIME_WAIT. Because of sampling issues due to the discrete timers typically implemented on computers, simply waiting MSL time is insufficient, thus 2*MSL is used. This method is considered undesirable because of the added complexity to the connection state machine. If the key were restricted to monotonically increase for each new connection, a bound can be put on the maximum number of connections per second (MCPS) before connection aliasing could occur:

$$\mathrm{MCPS} < \frac{2{*}{*}32}{\mathrm{MSL}} \text{ ~= 35 million connections per second.}$$

6.2 Proposed change

Require the locally assigned key to monotonically increase for each new connection on a specific host.

7.0 Connection Aliasing Many Hosts to a Single Host

7.1 Problem Definition

As mentioned above, an ST connection is defined by the connection triplet:

Local-Port : Local-key : Remote-Port

Memory_Region_Available (MRA)
Request_State (RA)
Request_State_Response (RSR)
Request_Answer (For RMR, RTS, RTR)
End
End_Ack

Block Operations

Clear_To_Send (CTS)
Put
Get
Fetchop
Fetchop_Complete
Request_Answer (only for Put, Get, and Fetchop)

STU operations

Data

Currently the transaction-id is 16 bits. This allows 64K transactions before the transaction-id wraps. The specification does not restrict how T-id's are allocated, so aliasing of transactions is possible. If T-id's were restricted to being allocated monotonically, an upper bound on the maximum transactions per second (MTPS) would be:

MTPS < 2**16/MSL ~= 546 transaction per second

This is clearly insufficient. Extending the T-id to 32 bits would allow:

MTPS < 2**32/MSL ~= 35 million transaction per second

However the ST message format as currently defined can not support this extension.

5.2 Proposed Change

Redefine the ST header to support a 32 bit transaction ID.

6.0 Connection Aliasing Single Host to Single Host

6.1 Problem Definition

An ST connection is defined as the triplet (which I'll call the connection triplet):

Since a transfer can be infinite length, the B_num can wrap. However, the minimum size a Block can be (except the first and last) is 2**8 bytes. Thus the channel speed calculation becomes:

2**32 * (2**6 + 2**8) ~= 2**40 ~= 1 terabyte/sec

It is still possible to alias a Block for a Write or Read transfer. Forinstance if a Write transfer of 2**64 is requested, and a CTS returns the first and last block, the two could have the same block number. To prevent this the Responder could be required to transmit CTS's in order (which seems burdensome), or CTS_Req could be changed to a required maximum number of outstanding Blocks. The later seems reasonable, and starts to provide a mechanism to implement a sliding window protocol for B_nums to allow reliable delivery, if so chosen.

If CTS_req is defined as the maximum number of outstanding CTS's, then to avoid aliasing the B_num:

CTS_req < 2**31

Because CTS_req is a 16 bit field, this restriction is naturally enforced.

4.2 Proposed Change

Extend the F_id and G_id fields to 32 bits.

Require F_id and G_id to monatomically increase per Block into all persistent memory regions open on a single port (solves Block and RMR aliasing).

Restrict the responder for an RTS to not allow the B_num in outstanding CTS’s to alias to a prior Block. If the responder is implementing a sliding window, this effectively limits the sliding window to 2**31 Blocks outstanding. One way of implementing this would be to change CTS_req to be CTS_Max, and require the Responders to never have more than CTS_Max CTS’s outstanding. Since CTS_Max would be a 16 bit field, a maximum number of 65536 Blocks could be outstanding at any one time.

5.0 Transfer/Persistent Memory Aliasing

5.1 Problem Definition

All messages that control and send data for a transfer and/or persistent memory region must be examined for aliasing. This includes the following messages:

Transfer Operations

Request_to_Send (RTS)
Request_to_Receive (RTR)
Request_Memory_Region (RMR)

Fetchop
Fetchop_Complete
Request_Answer (only for Put, Get, and Fetchop)

STU operations

Data

CTS, Put, and Data have a 32 bit B_num in the header. Get, Fetchop, Fetchop complete, and Request_Answer do not, but G_id and F_id serve the same purpose, except that they are only 16 bits.

Because each transfer is unique, only Block aliasing within a single transfer need be examined.

For Persistent memory operations, the block number for Put's is allowed to increment indefinitely and wrap. Because each block could be 1 byte, the sequence number would wrap after 2**32 bytes. To avoid aliasing blocks, the maximum blocks per second (MBPS) is:

MBPS = 2**32/MSL ~= 35 million blocks/sec

Thus to alias a block, the channel speed must consume the MAC header, ST header, and one byte data payload at a rate greater than MBPS. The ST header is 40 bytes, and the MAC header is somewhere between 22 and 24 bytes (Ethernet is 22 bytes and HIPPI-6400 is 24). Let's assume 64 bytes of overhead plus one data byte per block. Thus to alias a block the channel data rate must be greater than:

2**32 * (2**6 + 1) ~= 2**38 bytes ~= 256 gigabytes/sec

Thus delayed duplicates can safely be detected for Puts.

This same analysis can be applied to GET, Fetchop, and Fetchop complete, using G_id and F_id, but the sequence number is only 16 bits.

2**16 * (2**6 + 1) ~= 2**22 bytes ~= 4 megabytes/sec

If we assume that a channel can be saturated with 1 byte Get or Fetchops, then delayed duplicate aliasing could occur on a channel operating at just 32 mbits/sec! If the G_id and F_id were 32 bits, then this is not a problem for any channel rate in the foreseeable future.

The Put analysis can also be applied to Writes and Reads, with the restriction from Section 6.2.5:

> *"All of the Blocks of a Transfer shall be the same size, except for the first and/or last STU of a Block which can be smaller".*

Get_size <= (2**16) * min(I-Max_Block_size, I_Max_STU_size, I_Bufsize)

It does not have the required state to perform this calculation.

Note that Fetchop is defined to be 8 bytes in length, and a single STU, thus calculating the Block_size does not apply.

If the Max_Block_size was transmitted with the RTS and RMR, then the opposite end could calculate the block correctly.

3.3 Proposed Change

If tiling is used,

STU_size = min(Max_STU_size, Bufsize)

Max_blk_size <= (2**16) * STU_size

2**8 <= Bufsize <= 2**32 (Offset address all of Buffer)

To make Max_STU_size symmetric with Bufsize, change it to:

2**8 <= Max_STU_size <= 2**32

This causes Max_blk_size to be restricted to:

2**8 <= Max_blk_size <= 2**48

To ensure proper calculation of Blocksize for CTS and Put_size for Persistent Memory, modify RTS and RMR to include Max_Block, where:

I-Max_Block <= min(I-Max_Block_size, I-Max_STU_size, I-Bufsize)

I-Max_Block can vary on a per Operation Sequence basis.

To cover the case when tiling is not used, add text to Section 6.2.6 stating:

> *"STU_num shall not wrap."*

4.0 Block Aliasing

4.1 Problem Definition

All messages that control and send data for block operations must be examined for aliasing. This includes the following messages:

Block Operations

- Clear_to_Send (CTS)
- Put
- Get

Table 1 summarizes the state of each end after a connection setup. Note that the Request_Connection operation specifies the local I-Max_Block_size and I-Max_STU_size, but this value can be modified by intermediate devices. Thus the Initiator does not know the final negotiated I-Max_Block_size and I-Max_STU_size. These values are used when sending Data operations from the Responder to the Initiator. Another way of putting it is that ***I***-Max_Block_size and ***I***-Max_STU_size specifies how to send data to the Initiator.

A Data transfer operation can be initiated by either end node, but for consistency I'll assume the Initiator begins the sequence.

Table 1: Known State After Connection Setup

Initiator State	Responder State
I-Bufsize	R-Bufsize
R-Bufsize	I-Bufsize
R-Max_Block_size	I-Max_Block_size
R-Max_STU_size	I-Max_STU_size

For a Read or Write Sequence, the side which generates the CTS must calculate the Block size. If tiling is not used, then the end node will probably have to choose a block size of <= 64 KB to ensure the source does not wrap STU_num.

If tiling is used, the Blocksize must be:

Blocksize <= (2**16) *min(Max_Block_size, Max_STU_size, Bufsize)

If the Responder is calculating Blocksize for a CTS, the equation is:

Blocksize <=(2**16)*min(R-Max_Block_size, R-Max_STU_size, R-Bufsize)

This is because the Data STU's will be going into the Responder's buffers.

Thus the Responder does not have enough state to calculate Blocksize such that it will not violate the above restriction.

A Put message must obeys a simliar constraint. Assume the Initiator has negotiated the Persistent Memory region and wished to Put data into the Responders exposed buffers. Again, if tiling rules are obeyed, the maximum size Put is:

Put_size <= (2**16) *min(R-Max_Block_size, R-Max_STU_size, R-Bufsize)

The Initiator has the required state to size the Put.

If the Initiator wishes to perform a Get which can span multiple STU's, it does not have enough state to correctly size the maximum Get. The Get must obey the constraint:

If each STU was 2**8 bytes, then wrapping would occur in 2**24 bytes or 16 Mbytes. At HIPPI-6400 rates the wrap would occur in 20 msec. Thus it is possible to corrupt a data stream if MSL > 20 msec. If Block sizes were restricted to not allow STU's to wrap, then no aliasing could occur (assuming blocks can not alias).

The maximum number of STU's in a block if tiling rules are followed is:

STU_size = min(Max_STU_size, Bufsize)

$$1)\qquad \frac{\text{Max\_Blk\_size}}{\text{STU\_size}} \quad <= \quad \text{Max\_STU\_num}$$

The specification defines the parameters as:

2**8 <= Max_Blk_size <= 2**63

Max_STU_size <= Bufsize

Max_STU_size <= Max_Blk_size

2**8 <= Max_STU_size <= 2**31

2**8 <= Bufsize <= 2**63

Using 1) to solve for Max_blk_size (assuming tiling):

Max_blk_size <= (2**16) * STU_size

On a side note, if Bufsize were 2**63, because Offset is only 2**32 the upper half of the buffer is not addressable.

The specification does not place an explicit limit on the minimum STU size, stating (Section 6.2.6):

> *"There is no requirement that STU sizes be consistent throughout a Block or Transfer".*

Note that if tiling rules are not obeyed, the above quote means that filling a block with 1 byte STU's is acceptable. To ensure STU_num does not wrap within a Block, this effectively limits Blocksize to <= 64 KB.

3.2 Calculating Blk_size Such That STU_num Will Not Wrap

A consequence of requiring the STU_num to not wrap is that each type of Data message must be analyzed to make sure that the Blk_size can be calculated correctly if tiling is used. For Read or Write Sequences, the Blocksize parameter must be bounded correctly. For Put and Get Sequences, an upper bound must be put on the length of the Put or Get. Fetchop is restricted to a single STU, so there is no issue.

Disconnect_Complete (DC)

Transfer Operations

Request_To_Send (RTS)
Request_to_Receive (RTR)
Request_Memory_Region (RMR)
Memory_Region_Available (MRA)
Request_State (RA)
Request_State_Response (RSR)
Request_Answer (For RMR, RTS, RTR)
End
End_Ack

Block Operations

Clear_to_Send (CTS)
Put
Get
Fetchop
Fetchop_Complete
Request_Answer (only for Put, Get, and Fetchop)

STU operations

Data

To simplify implementations, it is assumed that if a machine is rebooted its reboot time is greater than MSL, or the machine must implement a quiet time to ensure all delayed duplicates have disappeared. Typically machines reboot slower than an MSL delay, so it is not envisioned that a reboot quiet time is required. (This simplification is typical in the TCP community).

3.0 STU Aliasing

3.1 Problem Definition

STU aliasing can only occur for a data message. Each data STU has a 16 bit STU_num in the header. Because each block is unique, just aliasing within a single block need be examined. The ST specification states that STU_num must start at zero, and increment by one for each STU. Thus aliasing could occur if STU_num was allowed to wrap. The specification does not restrict this.

dent, and can interface to the lower layers at either the network or data link layers. Because of this, proving the existence of an MSL must be done in the context of a bridged LAN and a routed WAN. Examples of each are HIPPI-6400 encapsulation and IP encapsulation, respectively.

For an MSL to be calculated for a LAN, several things must be examined

1) presence of loops
2) storage in the medium and congestion

1) is assumed to be solved by the bridges/switches, usually by some form of a spanning tree algorithm such as 802.1d.

2) An analysis of bridge/switch congestion is not attempted here, and is assumed to be no worse than the routed case (but without the protection of a hop count). This is reasonable because typically switches or bridges have little storage.

For a network layer interface, I'll assume we're using IP encapsulation. Because IP has a hop count, it is assumed that the hop count value allows determination of an MSL.

RFC 793 defines the MSL to be 120 seconds. RFC 1387 and [94Stevens] states that implementations typically set MSL to be between 30 and 120 seconds. I will assume the goal is 120 seconds.

Because of the complexity and versatility of ST sequence interactions, the approach I've taken is to ensure that at each layer delayed duplicates can be prevented by using an orthogonal algorithm from the layer below (e.g. non-overlapping sequence numbers, or completely separate sequence numbers). Using this logic it is possible to greatly simplify the analysis. Forinstance, if STU's can be shown to not alias within a Block, then one need only prove that Blocks can not alias by using an orthogonal proof to also show that STU's can not alias between Blocks.

Messages in ST tend to be unique to a specific layer of the protocol. Below is a categorization of the hierarchy used for this paper and what messages apply to each. Messages are grouped by shared common id fields.

Connection Creation Operation

Request_Connection (RC)

Connection Operations

Request_Connection (RC)
Connection_Answer (CA)
Request_Disconnect (RD)
Disconnect_Answer (DA)

To ensure Blocks can not alias:

Extend the F_id and G_id fields to 32 bits.

For a specific port, require F_id and G_id to monotomically increase per Block into all persistent memory regions open on a single port (solves Block and RMR aliasing).

Restrict the responder for an RTS to not allow the B_num in outstanding CTS’s to alias to a prior Block. If the responder is implementing a sliding window, this effectively limits the sliding window to 2**31 Blocks outstanding. One way of implementing this would be to change CTS_req to be CTS_Max, and require the Responders to never have more than CTS_Max CTS’s outstanding.

To ensure Transfers can not alias:

Redefine the ST header to support a 32 bit transaction ID.

To ensure that Connections can not alias:

Require the locally assigned key to monotonically increase for each new connection on a specific host.

If a host allows multiple remote hosts to connect to the same port, the local host must control key allocation such that if a second RC is received on a specific port, the returned key must be unique for that destination port.

2.0 The Approach

Aliasing errors can occur for many reasons, but the most insidious of the version of an aliasing errors occurs when delayed duplicates are present. Solving delayed duplicates solves all other forms of aliasing, so this paper will focus specifically on ensuring no aliasing errors in the presence of delayed duplicates.

Delayed duplicates can happen because the network has the ability to store packets. I am assuming that the network does not replicate packets. Packet replication occurs because the source times out and retransmits a packet. Delayed duplicates can break the protocol or cause data corruption.If, for example, a sequence number wrapped such that the delayed duplicate data message from a previous connection can be interpreted as a valid data message for the current connection corruption of the datagram stream would occur.

The key to solving the delayed duplicates is to be able to assume that a packet has a maximum lifetime in the network, which I'll call (for compatibility with RFC 793) the Maximum Segment Lifetime (MSL). ST is network indepen-

Avoiding Aliasing Errors in Scheduled Transfers

James Pinkerton
Silicon Graphics

- DRAFT -

1.0 Executive Summary

This document outlines several recommended clarifications to the ST specification to provide constraints for sequence numbers that either wrap or are reused. It was found that when analyzing the protocol for aliasing errors in the context of maximum sizes, lifetimes, and rules for wrapping there were several cases which could cause aliasing errors. These errors can be shown to lead to protocol errors or data corruption.

This is a summary of the proposed changes:

To ensure STU's can not alias:

If tiling is used, STU_size = min(Max_STU_size, Bufsize). Changes are:

> Max_blk_size <= (2**16) * STU_size
> 2**8 <= Bufsize <= 2**32 (Offset address must address all of Buffer)
> 2**8 <= Max_STU_size <= 2**32
> 2**8 <= Max_blk_size <= 2**48

To ensure proper calculation of Blocksize for CTS and Put_size for Persistent Memory, modify RTS and RMR to include Max_Block, where:

I-Max_Block <= min(I-Max_Block_size, I-Max_STU_size, I-Bufsize)

I-Max_Block can vary on a per Operation Sequence basis.

Add text to Section 6.2.6 stating:

> "STU_num shall not wrap."