NGTCP: Next Generation Transmission Control Protocol

A Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Bachelor of Technology

by
Ambarish Narayan Gupta
Sandeep Gupta

to the
Department of Computer Science & Engineering
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
April, 2000

To our parents

Certificate

Certified that the work contained in the report entitled *NGTCP : Next Generation Transmission Control Protocol*, by Amabarish Narayan Gupta and Sandeep Gupta, has been carried out under our supervision and that this work has not been submitted elsewhere for a degree.

(Dr. Dheeraj Sanghi) (Dr. Rajat Moona)

Abstract

This project's aim is to implement an experimental extension for TCP, the NGTCP standard, for the Linux operating system. This document describes the design and implementation of NGTCP , and presents some comparative analysis between NGTCP and TCP based on the number of packets per session and the response time for short transaction-oriented connections.

Acknowledgements

We would like to thank our supervisors, Dr. Dheeraj Sanghi and Dr. Rajat Moona for encouraging us at every step and giving us valuable insight solving critical problems. We would also like to thank the members of the Linux mailing list for helping us with the implementation of NGTCP when we had problems. We would also like to thank the CSE lab staff for providing us with facilities to run our software. And we are thankful to our batch mates for all the cheers and love they gave to us.

Contents

Chapter 1

Introduction

The TCP/IP reference model is a specification for a networking stack on a computer. It exists to provide a common ground for network developers. This allows easier interconnection of the different vend or supplied networks, reducing the cost of installing completely new networks in order for one to work with another.

The most popular implementation of the transport layer in the reference model is the Transmission Control Protocol (TCP). This is a connection-oriented protocol. Another popular implementation is the User Datagram Protocol (UDP), which is a connectionless protocol.

Both of these protocols have advantages and disadvantages. The two main aspects of the protocols make them useful in different areas. UDP is a connectionless protocol. UDP always assumes that the data was received correctly. The application layer above it looks after error detection and recovery. Even though UDP is unreliable, it is quite fast, and is useful for applications such as DNS (Domain Name System), where speed is preferred over reliability. TCP on the other hand, is a reliable, connection oriented protocol. It does a 3-way handshake before connection establishment to avoid false connections. It looks after error detection and recovery. Retransmission of data is done automatically if a problem is detected. As a result of being more reliable, TCP is a slower protocol than UDP. It introduces delays of one round trip time before data-packets can be exchanged. Also it's small window size acts as bottleneck for large data transfers over gigabit networks.

The current transport protocols are thus either too reliable or too unreliable.

They lie at either end of the scale in taking into account speed and reliability. TCP has reliability at the cost of speed, whereas UDP has speed at the cost of reliability. Moreover applications have become a lot more sophisticated and have started demanding quality of service. This calls for need of a new protocol which can support a variety of reliability options spanning the range from unreliable delivery like UDP to extremely reliable delivery like TCP. The protocol should be fast, and should not impose lot of overheads for short transactions. Also it should perform well over gigabit networks.

TPng[1] is envisioned as a successor protocol satisfying all above requirements. It also presents many other novel features in addition. TCP for Transactions T/TCP[3] is another experimental protocol which gives an insight of how to incorporate transactions in TCP. In following chapters we study the development of a new protocol NGTCP which is based on TPng and T/TCP.

Chapter 2

Background

We describe UDP and TCP in following sections.

2.1 User Datagram Protocol

User Datagram Protocol is an unreliable connectionless protocol defined in. It is useful for applications that do not require or want TCP's sequencing or flow control. It is used for one-shot, request-reply applications where prompt delivery is important. Examples of these types of applications would be DNS (Domain Name System) and transmission of speech or video. UDP minimizes the overhead associated with message transfers because no network connection is established before transmission.

UDP can be likened to the postal service. A message is sent to someone else by putting the address on the envelope and dropping it into the letter box. The sender has to rely on the underlying system, in this case the postal service, to deliver the letter. The letter can traverse countries and continents, where each different country has a different system, different stamps and charges, providing there is a reliable service, the letter will be delivered. UDP is similar, it drops the datagram onto the underlying architecture, the Internet Protocol, and hopes that the message is delivered. It has no way of verifying that the datagram was delivered. It does not do any error checking and it has no way of recovering data that was incorrectly delivered.

2.2 Transmission Control Protocol

Transmission Control Protocol is a reliable connection-oriented protocol that allows a byte stream, originating on one machine, to be delivered without error to any other machine. It fragments the message into discrete packets and passes them onto the internet layer.

TCP has the ability to handle flow control. This prevents the source machine from swamping a slower destination machine with data. If the destination machine's buffer becomes full with incoming packets, the TCP will send a control signal to the source machine indicating that it cannot handle any more information

at the moment and to slow down the transmission.

TCP has the ability to handle sequencing. When packets are being sent out, not all of them will take the same route. This may result in packets being delivered out of sequence. TCP has a way of reordering the segments to avoid the need of the sender resending all the segments again.

In the previous section, it was noted that UDP could be compared to the postal service. In a similar analogy, TCP can be compared to the telephone system. When a call is made, a direct connection is made between the two people involved in the conversation.

The operation of the TCP protocol can be divided into three distinct sections:

1. Establishment of Connection

2. Transmission of Data

3. Termination of Connection

2.2.1 Establishing the Connection

The connection is established between two hosts by a method known as the 3-Way Handshake. Three segments are transmitted before the two hosts are fully synchronized and ready to transmit the data.

With reference to figure 1 and figure 2, a host that wishes to make a connection sends out a TCP segment with the SYN flag set and the proposed initial sequence

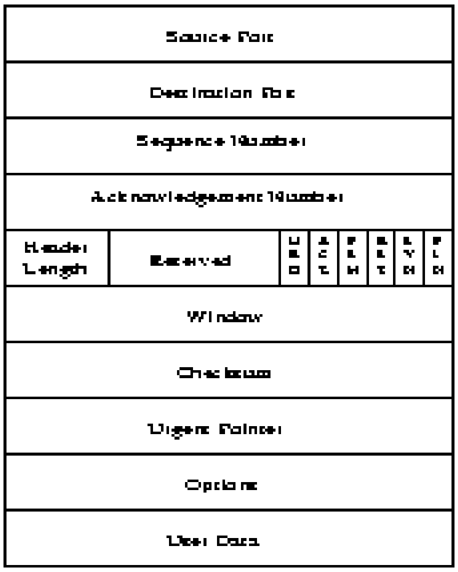

Figure 1: TCP Header Structure

number in the sequence field, say sequence=X. The TCP on the destination machine notes the sequence number X, and returns a segment with the SYN and the ACK flags set. It also populates the sequence number field with its own value, say Y, and the acknowledgment field with the value X+1.

The source machine receives the segment, notes the value Y, and returns a segment with the ACK flag set and the acknowledgment field set to Y+1.

The two hosts have now established connection, and the transfer of data may be started.

2.2.2 Data Transmission

The transfer of data is dominated by two mechanisms, acknowledgements of data and sequence numbers in a segment to allow for re-assembly.

When a segment of data is transmitted, the host that transmitted it expects to receive an acknowledgment within a certain time period. If the acknowledgment is not forthcoming, the host retransmits the data. This is how TCP ensures that the data is delivered.

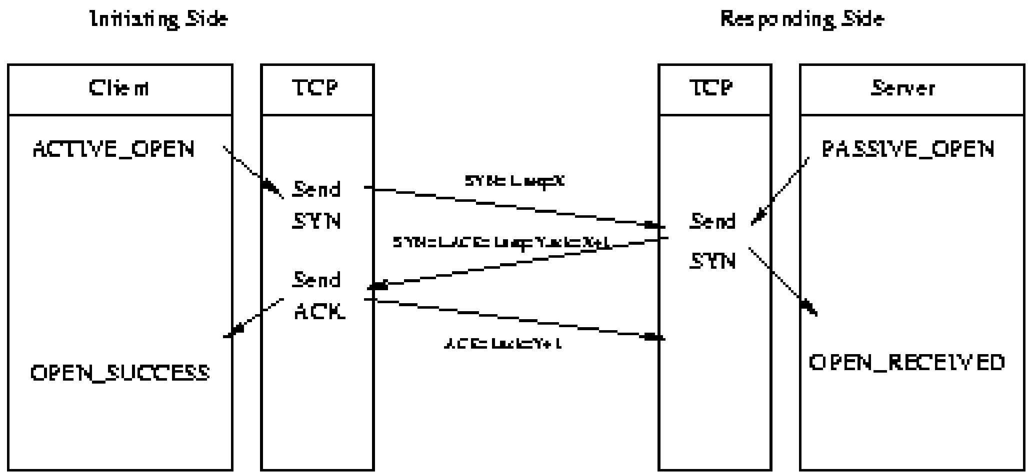

Figure 2: The 3 Way Hand Shake

Sequence numbers allow the receiving host to reassemble any out of order packets received from a host. The sender sets the sequence field to a predetermined value. The receiver takes this sequence number, and adds to it the number of bytes in the segment, this is calculated from the amount of data received, plus one byte for each of the SYN or FIN flags that are set. This is the next sequence number the receiver expects to see in a packet from the sender. This is also the acknowledgment number the receiver sends to the other host. If a packet arrives that doesn't have the correct sequence number, the receiver can determine whether it is an old duplicate or if it is a packet that has been delayed in the network. The receipt of a duplicate packet allows the host to discard it, thus making sure that the receiving process only gets the data once.

2.2.3 Termination of Connection

Keeping in mind that TCP connections are full duplex, we can view them as being two independent pipes of communication between the two host computers. When an application program has no more data to send, it informs the TCP service. The TCP closes its half of the connection by sending the rest of the data that may be buffered and then sends a segment with the FIN flag set. The receiving TCP acknowledges the receipt of the FIN segment, and informs its own application that there will be no more data received. The TCP that is still open may continue to send data onto the original TCP until it terminates the connection itself, the application may still have data to send even though the other TCP has finished. When both connections are closed, the connection is deleted.

2.3 Summary

A transaction with UDP takes 2 segments, the request and the reply. With TCP a transaction takes 10 segments. As will be seen in chapter 3 NGTCP can complete a transaction in a minimum of 3 segments. A reduction in the case of TCP, but an increase for UDP. The advantage that NGTCP holds over UDP is the reliability.

Chapter 3

Next Generation Transmission Control Protocol

3.1 Motivation

The current TCP has reached its limit of operation. This is evident by following facts:

3.1.1 Limitation due to insufficient number spaces.

1. The 16 bit window size allows at most 64 KB of data to be in transit at a time, which is too little for today's gigabit networks.

2. The 32 bit sequence space can easily wrap around over gigabit networks, leading to protocol breakdown.

3. The 16 bit port numbers can easily be exhausted by an active server serving a large number of clients.

4. The 4 bit header length is too small to support a good number of options.

3.1.2 Limitations due to protocol semantics.

1. The 3 way handshake introduces a time lag of one round trip time before data can be sent. This certainly leads to performance problems in high delay networks like satellite channels. It also acts an overhead in terms of bandwidth

for short-lived connections (Transactions) in which only one or two packets are exchanged, with an accompanying poor response time.

2. TCP is excessively reliable (at an excessive cost) for applications like RPC which actually do not need such levels of reliability. On the other hand UDP is too unreliable.

3.2 A Vision of General Purpose Transport Protocol

In last two decades, there have been a large number of transport protocols proposed by various researchers like NETBLT, VMTP, DTP etc. But most of these protocols were designed for specific environments. We aim for a general purpose protocol that can take care of very basic requirements.

Although of the problems cited above, have been solved but they make current TCP too patched up. Many of these patches will take a long time to distribute leaving the network in a highly heterogeneous state. They also make implementation very complicated and therefore open to bugs and security holes. Moreover, TCP can have only limited number of options, which means that we can improve the protocol only upto a limit (because there is not sufficient space in the header). Therefore a new protocol becomes very important.

In our new version of transport protocol which we hence forth refer to as NGTCP, we would not only improve on the shortcomings of TCP, we would also like to add more features. We would like to provide new types of reliability (no acks, no flow control, losses acceptable) and support for transaction based applications.

3.3 NGTCP Features

3.3.1 Scaled Number Spaces

64 bit Sequence Numbers

With the assumption that the network speeds do not exceed 1Gbps (implying a clock granularity of 2ˆ-30 secs) the 64 bit sequence will wrap in approxmately 6 centuries. This is too huge a duration for any connection to persist.

In TCP, a danger with fast connections is that they may end up using the 32 bit sequence space faster (and thus wrap), so that older packets of the same connection may interfere with later packets if the sequence space wraps in less than MSL (Maxmum Segment Lifetime) duration. TCP dealt with this problem by using 32 bit timestamps options, effectively extending the sequence space to 64 bits.

The 64 bit sequence space of NGTCP provides a much cleaner solution. It makes every connection slow and short (as sequence space never wraps). Successive connections between same host and port pair use disjoint sets of sequence numbers as ISN increases at a faster rate than the sequence numbers get used up in data transfer. The possibility of old duplicates reappearing is thus negligible. An old packet from a host will always bear a sequence number less than the latest packet from that host. In the event a a host crashes and reboots and starts using sequence numbers from a random start point, then their won't be any problems. This is because it must wait for atleast MSL duration before rebooting so that all old packets from this host have disappeared.

32 bit Windows

With 32 bit windows a maximum of 4 gigabytes of data can be in transit which is by far sufficient for all purposes. It can easily comply with the jumbograms (huge packets) of IPv6.

32-bit ports

The 32-bit ports with can provide an enormous transaction rate. It would amount to 2ˆ32 / (Duration in TIME-WAIT) as opposed to 268 Tps in TCP. The duration in TIME-WAIT is also reduced from 240 seconds in TCP to some multiple of RTT in NGTCP (explained later).

3.3.2 Transaction-oriented Service

Currently, a transaction-oriented Internet application must choose to suffer the overhead of opening and closing TCP connections or else build an application-specific transport mechanism on top of the connectionless transport protocol UDP. Hence greater convenience, uniformity, and efficiency would result from widely-available kernel implementations of a transport protocol supporting a transaction service model [RFC- 955].

Transaction characteristics

- The fundamental interaction is a request followed by a response.
- An explicit open or close phase would impose excessive overhead.
- At-most-once semantics is required; that is, a transaction must not be "replayed" by a duplicate request packet.
- The minimum transaction latency for a client is RTT + SPT, where RTT is the round-trip time and SPT is the server processing time.

Transactions Using Standard TCP

Consider a simple transaction in which client host A sends a single segment request to server host B, and B returns a single-segment response. Current TCP implementations use at least ten segments (i.e., packets) for this sequence:

- 3 for the three-way handshake opening the connection,
- 4 to send and acknowledge the request and response data, and
- 3 for TCP's full-duplex data-conserving close sequence.

These ten segments represent a high relative overhead for two data-bearing segments. However, a more important consideration is the transaction latency seen by the client: 2*RTT + SPT, larger than the minimum by one RTT. As CPU and network speeds increase, the relative significance of this extra transaction latency also increases.

The TCP close sequence also poses a performance problem for transactions: one or both end(s) of a closed connection must remain in "TIME-WAIT" state until a

4 minute timeout has expired . The same connection (defined by the host and port numbers at both ends) cannot be reopened until this delay has expired. Because of TIME-WAIT state, a client program should choose a new local port number (i.e., a different connection) for each successive transaction. However, the TCP port field of 16 bits provides only 64512 available user ports. This limits the total rate of transactions between any pair of hosts to a maximum of $64512/240 = 268$ per second. This is much too low a rate for low-delay paths, e.g., high-speed LANs. A high rate of short connections (i.e., transactions) could also lead to excessive consumption of kernel memory by connection control blocks in TIME-WAIT state.

Hence to perform efficient transaction processing in TCP, we need to suppress the 3-way handshake and to shorten TIME-WAIT state in our new protocol.

Transactions Using NGTCP

Bypassing 3-way handshake

To avoid 3-way handshakes for transactions, we introduce a new mechanism for validating initial SYN segments, i.e., for enforcing at-most-once semantics without a 3-way handshake. We refer to this as the NGTCP Accelerated Open, or TAO, mechanism.

NGTCP Accelerated Open

NGTCP uses cached per-host information to immediately validate new SYNs. If this validation fails, the procedure falls back to a normal 3-way handshake to validate the SYN. This is Accelerated Open which simulates a 3-way handshake. Thus, bypassing a 3-way handshake is considered to be an optional optimization.

What do we cache and how it helps?

We cache the last sequence number (lsn) of a host from a latest connection from that host. So when host B receives from host A an initial SYN segment host B compares it against cache[A].lsn , the latest value that B has cached for A. The validation fails if there either the the new sequence number is less than or equal to the stored one or there is no current cached state. Else the validation succeeds.

A point to note is that we need not worry about duplicate or old SYN segments, as it is taken care of by our 64 bit sequence space which practically never wraps. All segments from a host successively occupy the ever monotonic sequence space, so by design there won't be any old duplicates . In case a host crashes and reboots, it would ask for cache re-synchronization.

Considering a transaction in which client host A sends a single segment request to server host B, and B returns a single-segment response.

The number of segments required in a favorable situation are:

- 1 for the accelerated open and sending data and initiating close,
- 1 to acknowledge the request and to send response data and initiate close,
- 1 for completing the full-duplex close sequence.

More than 3 segments may be transferred in case acks are not piggybacked.

Thus we see that we have improved on:

1. Response time which is RTT+SPT in this case.
2. A saving of 66% of packets being transferred as compared to TCP (obviously in cases where large amount of data is being transferred, there will be more packets transmitted and hence less in the percentage saving).

3.3.3 Truncation of TIME-WAIT

The TIME-WAIT state is a state that all TCP connections enter into when the connection has been closed. The length of time for this state is 240 seconds (twice the maximum segment lifetime), which is to allow for any duplicate segments still in the network from the previous connection to expire. Since we can always detect old duplicate segments, (because the sequence space spanned by every connection is disjoint owing to the huge 64 bit sequence space), we need not wait this long. We propose to remain in TIME-WAIT for some suitable multiple of round trip time so that a graceful close is ensured.

3.4 Examples

NGTCP can be beneficial to some of the applications that currently use TCP or UDP. At the moment there are many applications that are transaction based rather than connection based, but still have to rely on TCP along with the overhead. UDP is the other alternative, but not having time-outs and retransmissions built into the protocol means the application programmers have to supply the time outs and reliability checking themselves.

3.4.1 HTTP and RPC

HTTP is the protocol used by the World Wide Web to access web pages. HTTP is the classic transaction style application. The client sends a short request to the server requesting a document or an image and then closes connection. The server then sends on the information to the client.

With TCP, the transaction is accomplished by connecting to the server (3-Way Handshake), requesting the file and then closing the connection (sending a FIN segment). NGTCP would operate by connecting to the server, requesting the document and closing the connection all in one segment (TAO). It is obvious that bandwidth has been saved and response time enhanced.

Remote Procedure Calls also adhere to the transaction style paradigm. A client sends a request to a server for the server to run a function. The results of the function are then returned in the reply to the client. There is only a tiny amount of data transferred with RPC's.

3.4.2 DNS

Domain Name System is used to resolve host names into the IP addresses that are used to locate the host.

To resolve a domain name, the client sends a request with the IP address or a host name to the server. The server then responds with the host name or the IP address where appropriate. This protocol uses UDP as its underlying process.

As a result of using UDP, the process is fast, but not reliable. Furthermore, if the response by the server exceeds 512 bytes of data, it sends the data back to the client with the first 512 bytes and a truncated flag. The client has to resubmit the request using TCP. The reason for this is that there is no guarantee that the receiving host will be able to reassemble the IP datagram exceeding 576 bytes. For safety, many protocols limit the user data to 512 bytes.

NGTCP is the perfect candidate for the DNS protocol. It can communicate at speeds approaching that of UDP, and it has the reliability of TCP.

3.5 NGTCP Header Structure

3.5.1 Design Goals

The header structure for new protocol is completely overhauled along following lines.

- Any information not needed in majority of packets is not kept in the mandatory part of header.
- Sizes of various fields have been decided keeping in mind the future extensibility.
- All fields are to be aligned on natural boundaries.

3.5.2 Header Fields

- Version - The protocol should have a version number in the header. This will permit future extension of the protocol easily. A 4 bit field is proposed.
- Empty - A field of 28 bits. This is for future extensions.
- Header Size - A 16 bit field that will facilitate incorporating huge number of options as , the limit being 0.25 MB.
- Port Numbers - 32 bit fields, as proposed earlier.
- Sequence Number - A 64 bit field, as proposed earlier.
- Acknowledgment Sequence Number - A 64 bit field.

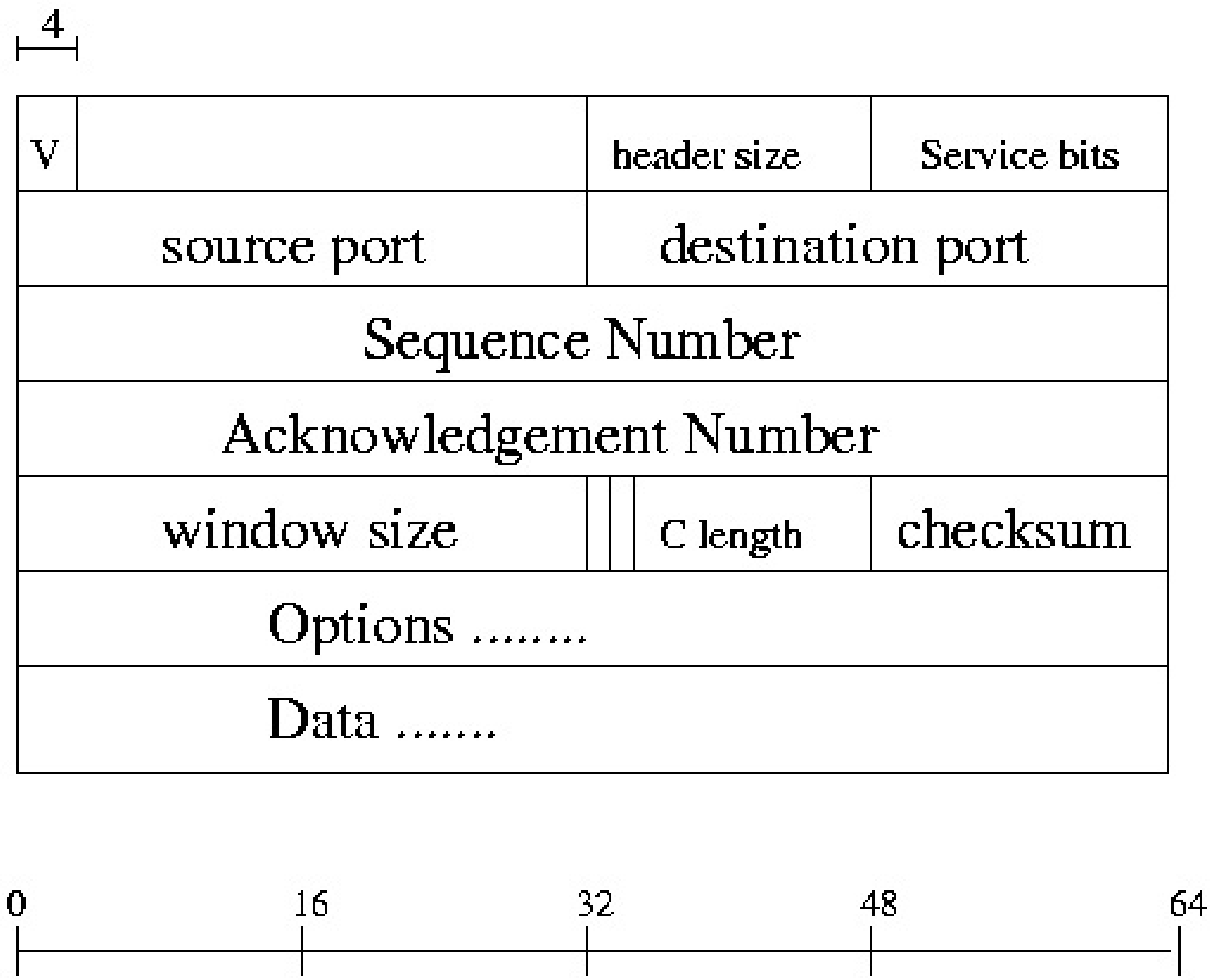

Figure 3: NGTCP Header Structure

- Window size - A 32 bit field, as proposed earlier.
- Checksum - Following fields are proposed:
- Checksum - A 16 bit field for sending the computed value of checksum using CRC-16 algorithm.
- Checksum length - A 14 bit field indicating length checksum coverage. So if an application wants more than 16 KB of packet to covered, then the entire packet will be covered.
- Checksum Flags - A 2 bit field to indicate following options:

1. No checksum present.
2. Checksum covers header only.
3. Partial checksum.
4. Checksum covers entire packet.

Service Bits

A 16 bit field for service flags.

- SYN, RST, FIN, ACK have similar meaning as in TCP.
- SAK - If set then acknowledgment is selective.
- REC - If set then then the protocol is record based (the peer should give the entire packet to the application as one). Otherwise the protocol is byte stream oriented.
- FLW - If set then flow control is to be used. The window size assumes significance. Otherwise no flow control is used.
- ORD - If set then peer should give packets to the application in sequence only. Otherwise packets may be delivered out of order.

- LOS - If set then the protocol should recover from losses. Otherwise application is asking for best-effort service.
- TRX - If set then transaction-based service is desired.

Options

There can be 0 or more options each in TLV (type-length-value) format. Each option is assigned a 8-bit number. The second byte will indicate the length of option in multiple of 4 bytes, limiting an option to a maximum of 1 KB. The remaining part is data associated with option.

Chapter 4

Design And Implementation

4.1 Design

4.1.1 Approach

The TCP state diagram has been modified to accommodate Accelerated Open required for transactions in the experimental protocol T/TCP. We started with this state diagram and simplified it by eliminating the highly unlikely paths to be traversed by the protocol endpoints. This included simultaneous Accelerated Open at client by server when client is in SYN_SENT* state.

The basis for design of NGTCP implementation is the state machine (figure x). The starred states are the new states introduced.

Half Synchronized Connections

TCP has always allowed a connection to be half-closed. TAO makes a significant addition to TCP semantics by allowing a connection to be half-synchronized i.e. to be open for data transfer in one direction before the other direction has been opened. Thus, the passive end of connection (which receives the initial SYN) can accept data and even a FIN bit before its own SYN has been acknowledged.

For half-synchronized connections we have following enhancements in NGTCP.

1. The passive end must provide an implied initial data window in order to accept data. The minimum size of this window is a parameter in the specification. Suggested is 4K bytes.

2. New connection states and transitions are introduced into the FSM at both ends of the connection. At active end, new states are required to piggy-back the FIN on the initial SYN segment. At passive end, new states are required for a half-synchronized connection.

4.1.2 State diagram

The FSM described by the state diagram is intended to be applied cumulatively; that is, parsing a single packet header may lead to more than one transition. Each new state (in addition to states of TCP) in NGTCP is indicated by standard state followed by a star.

There is a simple correspondence between these and their equivalent original states. States SYN_SENT* and SYN_RECEIVED* differ from corresponding unstarred states in recording the fact that a FIN has been sent. The other new states with starred names differ from the corresponding unstarred states in being half-synchronized (hence, a SYN bit needs to be transmitted).

Figure 5 shows an example of minimum transaction, highlighting the states traversed by client and server.

4.2 Implementation

The project is implemented on linux 2.2.5 kernel.

We have been able to implement the core functionality of the protocol which includes transaction-oriented service and 64-bit sequence numbers. We have also done Window sclaing.

Implementation of other services will be an incremental work on the platform we have developed.

The current implementation of TCP in the kernel is modified to handle new state transitions. A brief picture of implementation is as follows:

```
     TCP A   ( Client )                                   TCP B  (Server)

0    CLOSED                                               LISTEN

1    SYN_SENT*      --> < SYN, data1, FIN> -->            CLOSE_WAIT*
                                                          ( TAO test okay =>
                                                            data1 --> User_B)

                                                          LAST_ACK*

              <-- <SYN, ACK(FIN), data2, FIN> <--

2    TIME_WAIT
     (data2 --> User_A)


3    TIME_WAIT         --> <ACK(FIN)> -->                 CLOSED
```

Figure 4: An Example of Minimum Transaction

4.2.1 Application Program Interface

NGTCP sits as another transport protocol in the kernel like TCP and UDP. The interface is defined as :

- Application programs just need to specify the protocol identifier IPPROTO_NGTCP, to use the protocol.
- Clients wishing to use the transaction-service feature may use setsockopt call to set the NGTCP_TRX option, with option value as 1. For this they need to include the file '/usr/include/netinet/ngtcp.h'.
- To enable NGTCP to optimize on the number of segments transferred clients can optionally set the MSG_LAST flag in the their last send system call, which has a value of 0x8000.

4.2.2 Implementation Details

We started with the code of TCP, as the NGTCP has lot in common with TCP. NGTCP specific implementation required introduction many new data-structures and supporting routines and modifications in existing routines. Here is a brief description of important constituents relevant to by-passing 3-way handsahke.

Variables

- We remember that a handshake has yet not occurred by maintaining a variable 'handshake' in the control struct 'ngtcp_opt' . This variable is initialized to 0 (handshake pending) and is set to 1 whenever handshake is over (either 3-way or 1-way in case of TAO).
- We remember that transaction oriented service is desired by maintaining a variable 'trx_service' in the control struct 'ngtcp_opt'. This variable is initialized to 0 and is set to 1 if user sets the option for transaction-oriented service using setsockopt.

Caches

We introduced caches for storing the last sequence numbers as proposed in the protocol specification. These are set-associative caches. The caches are updated along following lines.

- Cache entries are always synchronized at the end of every connection (when a FIN is received from the other end) both at server and client, independent of whether transaction oriented service is requested or not.
- Caches entries are always invalidated soon after a handshake is done both at server and client, so that if a connection is reset in the middle due to unknown reasons, the caches do not have stale values, for the next connection.

4.2.3 Snapshot of Processing Done at Client and Server

Processing at Client

setsockopt

If the option is NGTCP_TRX then we set 'trx_service' to 1.

Connect system call

If 'trx_service' is set {

All processing related to sending an initial SYN packet is done, but the packet is held back from going to the IP layer.

We also do a cache lookup to find if the entry for the remote host is valid. The idea is that if we have invalid entry, then probably our host has rebooted and is now out of synchronization, implying that transaction-oriented service should not be requested. This is because the remote host's cache will now (if at all) have a stale cache entry, which now should not be used for TAO test. So 'trx_service' is reset to 0 for an invalid entry.

}

if 'trx_service' is not set {

Routine TCP processing is done.

}

Send system call

This involves very crucial processing. The data packet is build here, and appropriate flags are set. Depending on whether the 'handshake' is pending or not, we set the appropriate bits (SYN, TRX, FIN, ACK) in the packet during the processing of send system call.

If handshake is pending {

SYN bit is set and ACK bit not set.

if trx_service' is set, TRX bit is set.

if 'trx_service' is set and the user application has supplied appropriate flags (MSG_LAST) in the send system call, FIN bit is set.

}

If handshake is done {

ACK bit is set.

If the the user application has supplied appropriate flags (MSG_LAST) in the send system call, FIN bit is set.

}

Finally the packet is given out to IP layer for transmission.

Processing at Server

Listen system call

The server checks for the flags in the incoming packet.

If TRX bit is set {

TAO test is triggered:

A cache lookup is done. If entry is valid, and the ISN in the packet is greater than the cached sequence number then TAO test is passed.

So, if the test is passed 'handshake' is set to 1 (handshake over).

}

else {

Routine TCP processing is done }

A SYN-request (a struct) is created and queued to the listening socket.

If handshake is pending, then a SYN-ACK segment is sent to client, acking the SYN

If handshake is over{

A socket is created from the SYN-request.

Cache entry for the remote host is invalidated.

Data, if any is queued.

If the FIN bit is also set then the cache entry for this client host is updated.

}

Accept system call

All requests for which handshake is over are dequeued.

Following processing is meaningful only if TAO has been crossed or handshake is over.

Receive system call

Here data is transferred from the receive queues to the server application.

Send system call

Data is packeted and sent to the client. The packet carries an ack for SYN+number of data bytes+[FIN]. The FIN bit is set if the server application has supplied appropriate flags (MSG_LAST) in the send system call.

Processing at Client

'handshake' is set to 1.

If trx_service is set{

If the ack from the server acks only the SYN, then TAO has failed at the server else if it acks all the data bytes sent, then TAO has succeeded at server.

If TAO has failed then the old data segment is retransmitted with the flags SYN and TRX stripped off and flag ACK added.

else a simple ACK segment is sent.

}

else {

Routine TCP processing is done.

}

Receive system call

Here data received from server queued into into receive queues is transferred to the user application.

Now server and client can choose to follow a chain of sends and receives until a full duplex close is done.

Chapter 5

Performance Comparison Between NGTCP and TCP

5.1 Environment

- Machines : csews34 , csemt80
- OS : Linux 2.2.5 on both
- Processors : csews34 : Pentium 2 MMX at 233 Mhz with 32 MB main memory.
- csemt80 : Pentium 2 MMX at 233 Mhz with 64 MB main memory.
- Server is running on csemt80
- Client is running on csews34
- Intermediate network is LAN.

Transaction Definition

The transaction here comprises of client sending a data packet to the server and the server echoing it back. The client finally closes down after receipt of response packet.

5.2 Statistics Comparison for 1000 Transactions

Each of following statistics is taken with the help of 'time' command. The elapsed time represents the total time for a client to make 1000 consecutive (iterative) connections to server. This emulates the case where an active server is serving large number of transactions from different clients.

5.2.1 NGTCP running between csews34 and csemt80

1. 0.01user 0.21system 0:01.30elapsed 16%CPU
2. 0.01user 0.13system 0:01.30elapsed 10%CPU
3. 0.00user 0.14system 0:01.36elapsed 10%CPU
4. 0.00user 0.10system 0:01.36elapsed 7%CPU
5. 0.01user 0.13system 0:01.35elapsed 10%CPU
6. 0.00user 0.17system 0:01.34elapsed 12%CPU
7. 0.02user 0.17system 0:01.39elapsed 13%CPU
8. 0.01user 0.18system 0:01.30elapsed 14%CPU
9. 0.03user 0.13system 0:01.35elapsed 11%CPU
10. 0.01user 0.12system 0:01.31elapsed 9%CPU
11. 0.01user 0.13system 0:01.40elapsed 9%CPU
12. 0.00user 0.10system 0:01.36elapsed 7%CPU
13. 0.01user 0.12system 0:01.31elapsed 9%CPU
14. 0.01user 0.13system 0:01.30elapsed 10%CPU
15. 0.00user 0.15system 0:01.38elapsed 10%CPU
16. 0.01user 0.16system 0:01.34elapsed 12%CPU

17. 0.01user 0.15system 0:01.33elapsed 12%CPU

18. 0.01user 0.16system 0:01.31elapsed 12%CPU

Average elapsed time for NGTCP = 0:01.3383 seconds

Variance in elapsed time for NGTCP = 0:00.0316 seconds

5.2.2 TCP running between csews34 and csemt80

1. 0.00user 0.11system 0:01.69elapsed 6%CPU

2. 0.00user 0.09system 0:02.25elapsed 3%CPU

3. 0.00user 0.10system 0:02.47elapsed 4%CPU

4. 0.00user 0.11system 0:01.69elapsed 6%CPU

5. 0.00user 0.09system 0:02.25elapsed 3%CPU

6. 0.00user 0.10system 0:02.47elapsed 4%CPU

7. 0.00user 0.11system 0:01.69elapsed 6%CPU

8. 0.00user 0.09system 0:02.25elapsed 3%CPU

9. 0.00user 0.10system 0:02.47elapsed 4%CPU

10. 0.00user 0.16system 0:01.61elapsed 9%CPU

11. 0.02user 0.13system 0:02.31elapsed 6%CPU

12. 0.00user 0.16system 0:01.61elapsed 9%CPU

13. 0.02user 0.13system 0:02.31elapsed 6%CPU

14. 0.02user 0.16system 0:01.61elapsed 11%CPU

15. 0.00user 0.19system 0:02.05elapsed 9%CPU

16. 0.02user 0.16system 0:01.61elapsed 11%CPU

17. 0.00user 0.19system 0:02.05elapsed 9%CPU

18. 0.01user 0.15system 0:01.86elapsed 8%CPU

19. 0.01user 0.15system 0:01.90elapsed 8%CPU

20. 0.01user 0.15system 0:01.86elapsed 8%CPU

21. 0.01user 0.15system 0:01.90elapsed 8%CPU

Average elapsed time for TCP = 0:01.9052 seconds

Variance elapsed time for TCP = 0:00.3233 seconds

5.3 Statistics Comparison for Individual Transactions

The statistics is generated by tcpdump. Protocols are running between machines csews34 and csemt80.

The software tcpdump tells the protocol-id of the protocol when generating statistics of packets going in and out of machine. The protocol id of NGTCP is 7 while for TCP it is 6. This the way we distinguish the packets belonging to NGTCP and TCP.

In case of NGTCP, the response time is calculated as difference of timestamps of first packet which the request and the fifth packet which is the ack of the response of the server from the client.

NGTCP

1. 05:00:40.691444 csews34 > csemt80: ip-proto-7 77 (DF)

2. 05:00:40.691552 csemt80 > csews34: ip-proto-7 52 (DF)

3. 05:00:40.691686 csemt80 > csews34: ip-proto-7 44 (DF)

4. 05:00:40.691847 csemt80 > csews34: ip-proto-7 69 (DF)

5. 05:00:40.692061 csews34 > csemt80: ip-proto-7 44 (DF)

6. 05:00:40.692304 csews34 > csemt80: ip-proto-7 44 (DF)
7. 05:00:40.692359 csemt80 > csews34: ip-proto-7 44 (DF)
8. 05:00:40.692794 csews34 > csemt80: ip-proto-7 44 (DF)

Response time = 692061 - 691444 = 617 micro seconds

TCP

1. 05:00:42.461326 csews34.1424 > csemt80.15000: tcp 0 (DF)
2. 05:00:42.461405 csemt80.15000 > csews34.1424: tcp 0 (DF)
3. 05:00:42.461815 csews34.1424 > csemt80.15000: tcp 0 (DF)
4. 05:00:42.461929 csews34.1424 > csemt80.15000: tcp 25 (DF)
5. 05:00:42.461983 csemt80.15000 > csews34.1424: tcp 0 (DF)
6. 05:00:42.462081 csemt80.15000 > csews34.1424: tcp 25 (DF)
7. 05:00:42.462492 csews34.1424 > csemt80.15000: tcp 0 (DF)
8. 05:00:42.462542 csemt80.15000 > csews34.1424: tcp 0 (DF)
9. 05:00:42.462563 csews34.1424 > csemt80.15000: tcp 0 (DF)
10. 05:00:42.462596 csemt80.15000 > csews34.1424: tcp 0 (DF)
11. 05:00:42.462949 csews34.1424 > csemt80.15000: tcp 0 (DF)

Response time= 462492 - 461326 = 1166 micro seconds

NGTCP

1. 05:05:44.060335 csews34 > csemt80: ip-proto-7 77 (DF)
2. 05:05:44.060449 csemt80 > csews34: ip-proto-7 52 (DF)
3. 05:05:44.060575 csemt80 > csews34: ip-proto-7 44 (DF)

4. 05:05:44.060740 csemt80 > csews34: ip-proto-7 69 (DF)
5. 05:05:44.060955 csews34 > csemt80: ip-proto-7 44 (DF)
6. 05:05:44.061198 csews34 > csemt80: ip-proto-7 44 (DF)
7. 05:05:44.061254 csemt80 > csews34: ip-proto-7 44 (DF)
8. 05:05:44.061690 csews34 > csemt80: ip-proto-7 44 (DF)

Response time = 060955-060335=620 micro seconds

TCP

1. 05:05:46.547293 csews34.1426 > csemt80.15000: tcp 0 (DF)
2. 05:05:46.547375 csemt80.15000 > csews34.1426: tcp 0 (DF)
3. 05:05:46.547788 csews34.1426 > csemt80.15000: tcp 0 (DF)
4. 05:05:46.547900 csews34.1426 > csemt80.15000: tcp 25 (DF)
5. 05:05:46.547957 csemt80.15000 > csews34.1426: tcp 0 (DF)
6. 05:05:46.548058 csemt80.15000 > csews34.1426: tcp 25 (DF)
7. 05:05:46.548473 csews34.1426 > csemt80.15000: tcp 0 (DF)
8. 05:05:46.548525 csemt80.15000 > csews34.1426: tcp 0 (DF)
9. 05:05:46.548545 csews34.1426 > csemt80.15000: tcp 0 (DF)
10. 05:05:46.548580 csemt80.15000 > csews34.1426: tcp 0 (DF)
11. 05:05:46.548929 csews34.1426 > csemt80.15000: tcp 0 (DF)

Response time = 548473-547293=1180 micro seconds

NGTCP

1. 05:07:22.527383 csews34 > csemt80: ip-proto-7 77 (DF)
2. 05:07:22.527493 csemt80 > csews34: ip-proto-7 52 (DF)
3. 05:07:22.527622 csemt80 > csews34: ip-proto-7 44 (DF)
4. 05:07:22.527791 csemt80 > csews34: ip-proto-7 69 (DF)
5. 05:07:22.528005 csews34 > csemt80: ip-proto-7 44 (DF)
6. 05:07:22.528252 csews34 > csemt80: ip-proto-7 44 (DF)
7. 05:07:22.528309 csemt80 > csews34: ip-proto-7 44 (DF)
8. 05:07:22.528745 csews34 > csemt80: ip-proto-7 44 (DF)

Response time = 528005-527383=622 micro seconds

TCP

1. 05:07:28.278774 csews34.1427 > csemt80.15000: tcp 0 (DF)
2. 05:07:28.278864 csemt80.15000 > csews34.1427: tcp 0 (DF)
3. 05:07:28.279276 csews34.1427 > csemt80.15000: tcp 0 (DF)
4. 05:07:28.279390 csews34.1427 > csemt80.15000: tcp 25 (DF
5. 05:07:28.279447 csemt80.15000 > csews34.1427: tcp 0 (DF)
6. 05:07:28.279554 csemt80.15000 > csews34.1427: tcp 25 (DF)
7. 05:07:28.279965 csews34.1427 > csemt80.15000: tcp 0 (DF)
8. 05:07:28.280016 csemt80.15000 > csews34.1427: tcp 0 (DF)
9. 05:07:28.280036 csews34.1427 > csemt80.15000: tcp 0 (DF)

10. 05:07:28.280068 csemt80.15000 > csews34.1427: tcp 0 (DF)
11. 05:07:28.280421 csews34.1427 > csemt80.15000: tcp 0 (DF)

Response time = 279965-278774=1191 micro seconds

NGTCP

1. 05:09:22.004346 csews34 > csemt80: ip-proto-7 77 (DF)
2. 05:09:22.004762 csemt80 > csews34: ip-proto-7 52 (DF)
3. 05:09:22.004889 csemt80 > csews34: ip-proto-7 44 (DF)
4. 05:09:22.005057 csemt80 > csews34: ip-proto-7 69 (DF)
5. 05:09:22.005272 csews34 > csemt80: ip-proto-7 44 (DF)
6. 05:09:22.005511 csews34 > csemt80: ip-proto-7 44 (DF)
7. 05:09:22.005567 csemt80 > csews34: ip-proto-7 44 (DF)
8. 05:09:22.006001 csews34 > csemt80: ip-proto-7 44 (DF)

Response time = 005272-004346=926 micro seconds

TCP

1. 05:09:23.804243 csews34.1429 > csemt80.15000: tcp 0 (DF)
2. 05:09:23.804622 csemt80.15000 > csews34.1429: tcp 0 (DF)
3. 05:09:23.805036 csews34.1429 > csemt80.15000: tcp 0 (DF)
4. 05:09:23.805150 csews34.1429 > csemt80.15000: tcp 25 (DF)
5. 05:09:23.805207 csemt80.15000 > csews34.1429: tcp 0 (DF)
6. 05:09:23.805310 csemt80.15000 > csews34.1429: tcp 25 (DF)
7. 05:09:23.805720 csews34.1429 > csemt80.15000: tcp 0 (DF)

8. 05:09:23.805771 csemt80.15000 > csews34.1429: tcp 0 (DF)
9. 05:09:23.805791 csews34.1429 > csemt80.15000: tcp 0 (DF)
10. 05:09:23.805822 csemt80.15000 > csews34.1429: tcp 0 (DF)
11. 05:09:23.806179 csews34.1429 > csemt80.15000: tcp 0 (DF)

Response time = 805720-804243=1477 micro seconds

NGTCP

1. 05:11:40.484971 csews34 > csemt80: ip-proto-7 77 (DF)
2. 05:11:40.485079 csemt80 > csews34: ip-proto-7 52 (DF)
3. 05:11:40.485210 csemt80 > csews34: ip-proto-7 44 (DF)
4. 05:11:40.485382 csemt80 > csews34: ip-proto-7 69 (DF)
5. 05:11:40.485597 csews34 > csemt80: ip-proto-7 44 (DF)
6. 05:11:40.485840 csews34 > csemt80: ip-proto-7 44 (DF)
7. 05:11:40.485898 csemt80 > csews34: ip-proto-7 44 (DF)
8. 05:11:40.486333 csews34 > csemt80: ip-proto-7 44 (DF)

Response time = 485597-484971=626 micro seconds

TCP

1. 05:11:42.672570 csews34.1430 > csemt80.15000: tcp 0 (DF)
2. 05:11:42.672658 csemt80.15000 > csews34.1430: tcp 0 (DF)
3. 05:11:42.673071 csews34.1430 > csemt80.15000: tcp 0 (DF)
4. 05:11:42.673191 csews34.1430 > csemt80.15000: tcp 25 (DF)
5. 05:11:42.673255 csemt80.15000 > csews34.1430: tcp 0 (DF)

6. 05:11:42.673367 csemt80.15000 > csews34.1430: tcp 25 (DF)
7. 05:11:42.673782 csews34.1430 > csemt80.15000: tcp 0 (DF)
8. 05:11:42.673835 csemt80.15000 > csews34.1430: tcp 0 (DF)
9. 05:11:42.673854 csews34.1430 > csemt80.15000: tcp 0 (DF)
10. 05:11:42.673891 csemt80.15000 > csews34.1430: tcp 0 (DF)
11. 05:11:42.674239 csews34.1430 > csemt80.15000: tcp 0 (DF)

Response time = 673782-672570=1212 micro seconds

NGTCP

1. 05:15:01.302558 csews34 > csemt80: ip-proto-7 77 (DF)
2. 05:15:01.302975 csemt80 > csews34: ip-proto-7 52 (DF)
3. 05:15:01.303106 csemt80 > csews34: ip-proto-7 44 (DF)
4. 05:15:01.303296 csemt80 > csews34: ip-proto-7 69 (DF)
5. 05:15:01.303510 csews34 > csemt80: ip-proto-7 44 (DF)
6. 05:15:01.303755 csews34 > csemt80: ip-proto-7 44 (DF)
7. 05:15:01.303813 csemt80 > csews34: ip-proto-7 44 (DF)
8. 05:15:01.304250 csews34 > csemt80: ip-proto-7 44 (DF)

Response time = 303510-302558=952 micro seconds

TCP

1. 05:15:03.400909 csews34.1431 > csemt80.15000: tcp 0 (DF)
2. 05:15:03.401288 csemt80.15000 > csews34.1431: tcp 0 (DF)
3. 05:15:03.401701 csews34.1431 > csemt80.15000: tcp 0 (DF)

4. 05:15:03.401814 csews34.1431 > csemt80.15000: tcp 25 (DF)
5. 05:15:03.401872 csemt80.15000 > csews34.1431: tcp 0 (DF)
6. 05:15:03.401980 csemt80.15000 > csews34.1431: tcp 25 (DF)
7. 05:15:03.402395 csews34.1431 > csemt80.15000: tcp 0 (DF)
8. 05:15:03.402447 csemt80.15000 > csews34.1431: tcp 0 (DF)
9. 05:15:03.402466 csews34.1431 > csemt80.15000: tcp 0 (DF)
10. 05:15:03.402504 csemt80.15000 > csews34.1431: tcp 0 (DF)
11. 05:15:03.402852 csews34.1431 > csemt80.15000: tcp 0 (DF)

Response time = 402395-400909=1486 micro seconds

Average Response Time for NGTCP = 727.167 micro seconds

Variance in Response Time for NGTCP = 150.001 micro seconds

Average Response Time for TCP = 1285.333 micro seconds

Variance in Response Time for TCP = 139.413 micro seconds

Chapter 6

Future Work

Implementing Functionality of Service bits

Due to limited time we have not been able to implement the proposed services. A future work would be to implement all the services envisioned with this protocol.

Compatibility

Making NGTCP compatible with TCP is a big issue which we have not addressed. Finding ways to make it compatible would be a huge work in itself.

More Testing

The protocol is tested only on LAN, which doesn't bring out many facets of the protocol. A more suitable testing bed is required, so that more realistic performance measures can obtained.

Bibliography

[1] Sanghi D. Issues in Designing Next Generation Transport Protocol.

[2] Beck M, Bohme H, Dziadzka M, Kunitz U, Magnus R, Verworner D; Linux Kernel Internals; Addison-Wesley, 1996

[3] Braden R T, RFC1644 T/TCP - TCP Extensions for Transactions Functional Specification, Network Working Group, 1994

[4] Braden R T, RFC1379 Extending TCP for Transactions - Concepts, Network Working Group, 1992

[5] Braden R T, Jacobson V, Borman D, RFC1323 TCP Extensions for High Performance, Network Working Group, 1992

[6] Braden R T, RFC1337 TIME-WAIT Assassination Hazards in TCP, Network Working Group, 1992

[7] Postel J, RFC793 Transmission Control Protocol, Defense Advanced Research Projects Agency, 1981

[8] TCP/IP Illustrated Volume 2, The Implementation, Wright and Stevens.

[9] Internetworking with TCP/IP, Volume II, Design, Implementation and Internals.