DTL: Dynamic Transport Library for Peer-To-Peer Applications

Rakesh Gopinath Nirmala

Arunakumari Yedurupaka

I. Motivation

With the emergence of P2P computing, the need for advancements in transport layer protocols increased significantly. Many P2P applications such as video conferencing, content distribution etc., requires prioritization, low packet jitter and aggressive traffic control. Current TCP congestion control algorithms do not cater these diverse needs. In addition, most of the proposed solutions are scattered each solving specific issues. This makes it difficult to cherish the benefits of all the latest developments increasing the need for the integration into a single library for quicker and flexible applications development. DTL addresses this issue by putting all these at one place.

II. Contributions

The authors were the first to design a complete solution that provides a reliable, application level library implementing both LEDBAT and MulTCP. The two algorithms are used to provide varying levels of prioritizing traffic (less-than-best-effort to high) using a runtime configurable parameter. This ensures that the current data flows are not disrupted avoiding connection re-establishments. Also, the library is portable developed using JAVA NIO and could be deployed across multiple platforms. This obviates the need to look for platform specific solutions.

III. Solution

DTL inherits its design from Selective Acknowledgement enabled TCP and is implemented using JAVA over UDP. It tags each datagram with a TCP-like header consisting of fields - receiver advertised window, sequence number and ACK number to provide reliability and flow control. The sender keeps track of the congestion window, advertised window and threshold values. These values along with the bytes in traversal are used to compute the number of bytes to be sent next. To ensure in-order delivery and packet loss detection, sequence numbers are used. The packet loss is detected by the sender when either a packet times out or when a selective acknowledgement is received. The receiver sends a selective acknowledgement on reception of three subsequent out-of-order packets. The SACK informs the sender of already received segments and helps in retransmissions. In DTL, the steps to be taken in case of packet loss or ACK reception is defined on the server-side.

The aggressiveness of the flow is controlled based on two modes-Polite and Variable employing two algorithms-LEDBAT (Polite) and modified MulTCP (variable mode). LEDBAT uses one-way delay variation to calculate congestion. The authors include timestamps on every packet to measure this delay. On the sender side, minimum one-way delays for every minute is computed and stored in different queues. They are used in queuing delay and actual network delay measurements. To avoid errors in measurements and to ensure intra-protocol fairness, modified TCP slow-start with Additive increase & Multiplicative Decrease (AIMD) is applied to the initial LEDBAT flows forcing other active connections to drop. In this way, LEDBAT maintains the delay within a specified threshold (a.k.a. fixed or target delay). For variable mode, MULTCP is modified to take in a priority input, N, and virtualize a single flow into N parallel TCP flows. MulTCP congestion control closely resembles that of TCP in each virtual flow. To avoid packet loss and increase reliability, authors have implemented smooth slow start and MulTCP2 improvements that offers a better network control. The evaluations proves the claims presented and validates the traffic prioritization and intra-protocol fairness offered by DTL.

IV. Strong Points

1) The authors provide an all-in-one reliable and portable solution to address the congestion control issues in P2P applications.
2) The library manifests both LEDBAT and MulTCP that are considered to be the best algorithms to control congestion and reduce latency and packet loss within the network.
3) It eliminates the disruptions to the running flows and connection re-establishment costs with its on-the-fly priority specification and tuning.
4) Based on UDP, DTL eliminates the connection establishment overheads (3-way handshake) and easily pass through NATs. With application-level support, it obviates kernel dependency fostering faster adoption.

V. Weak Points

1) The evaluations are conducted mostly on the emulated hardware. This does not provide much insight into the performance of DTL in the real-world deployments.
2) P2P applications generally operate on large networks. But, the authors do not discuss anything about the scalability of DTL.
3) DTL is designed to be portable using a high-level language. But, the paper does not present any results to demonstrate its performance on different platforms.

VI. Questions

A. What is TCP slow start? How does Internet applications get around the TCP slow start?

Slow Start (SS) is one of the two algorithms used by TCP to address congestion control in a network. To start with, TCP transmit packets at a slow rate size of the sender window (CWND). For every acknowledged packet from the receiver, the CWND is increased with Maximum Segment Size (MSS) effectively doubling the sender capacity at each round. The process continues until enters the Congestion Avoidance phase i.e., one of the following happens SS threshold is reached or the receiver advertised window (RWND) is full.
However, TCP SS limits the full usage of the available bandwidth thus restricting faster data transmissions and resulting in poor performance. To avoid this, most of the Internet applications implement the following optimizations.

- Re-use the established TCP connections and avoid new connection establishments.
- Set the initial CWND and RWND to higher values and adjust them accordingly.