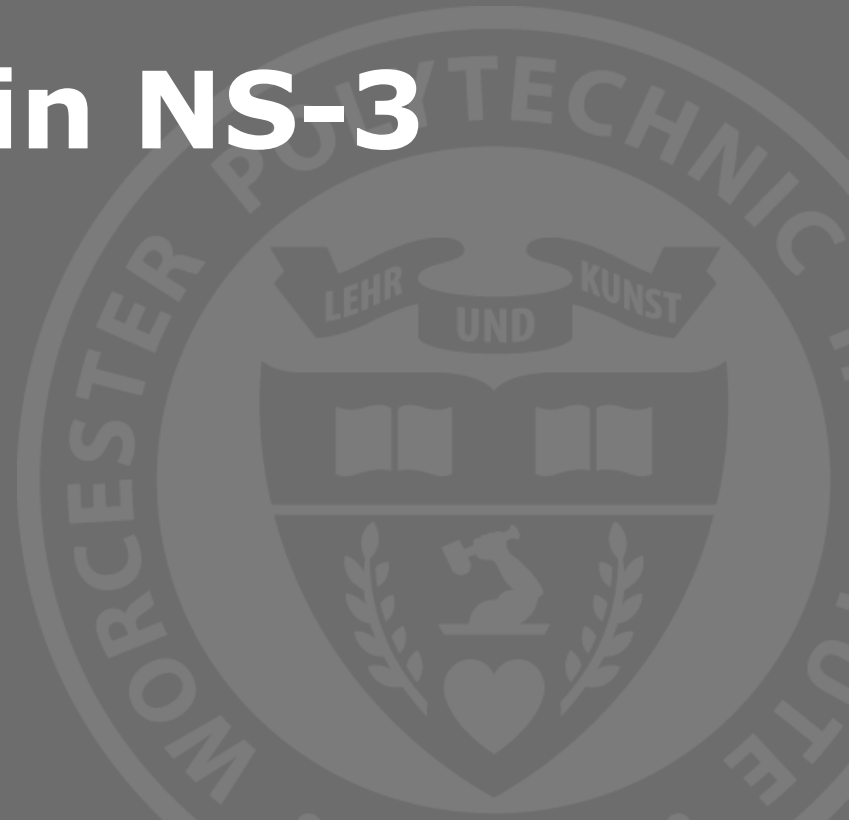




WPI

Compound TCP in NS-3

Keith Craig

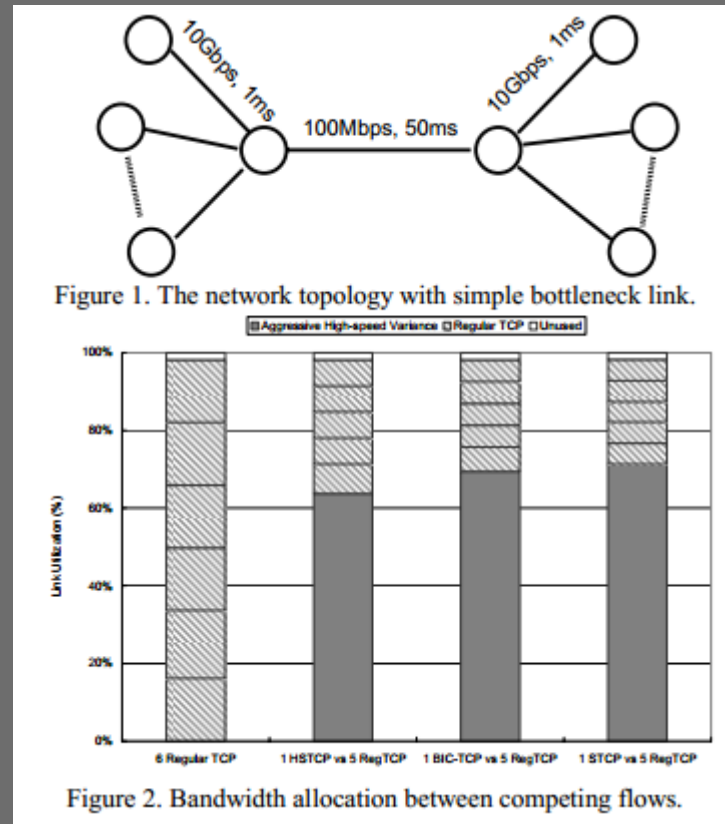


What is Compound TCP?

- As internet speeds increased, the long 'ramp' time of TCP Reno became an increasingly large issue.
- According to an IETF paper in 2003, a 10Gbps line would require no more than 1 drop every 100 minutes to achieve maximal throughput.
- A class of TCP algorithms known as "high speed" TCP algorithms attempt to alleviate this problem.
- These algorithms ramp up and recover after loss more quickly in order to more efficiently saturate available bandwidth.

The Trouble with BIC and Cubic

- BIC and Cubic (and loss-only based algorithms in general) exacerbate unfairness in scenarios with competing streams.



A Hybrid Approach

- Compound TCP attempts to mitigate the unfairness issues with BIC and Cubic by introducing a 'delay based' component to the congestion window, in the style of TCP Vegas.
- Broadly, Compound TCP calculates the total congestion window as the sum of a loss-based window (*cwnd*) that tracks packet drops, and a delay-based window (*dwnd*) modified by a moving RTT average.

Starting Up

- At the start of a new connection, CTCP uses the same slow-start behavior that TCP Reno uses.
- The `dwnd` is set to 0, disabling it during slow-start.
- For each packet ACKed, the `cwnd` is incremented by 1.
- The algorithm exits slow-start at the first dropped packet.

Loss-Based Window

- CTCP uses the AIMD approach from TCP Reno.
- When a packet is successfully ACKed, the cwnd increases by $1/\text{cwnd} + \text{dwnd}$.
- When a packet loss is detected, the cwnd is decreased by half.

Delay-Based Window

- A delay-based window attempts to predict oncoming congestion by tracking RTT variations.
- CTCP's algorithm for delay is based on TCP Vegas, the 'standard' form of delay-based window TCP algorithms.
- Broadly, given knowledge of the 'best-case' RTT and the current (or more typically, exponential moving average) RTT, decrease the *dwnd* if the difference between the two exceeds a defined parameter.

Delay-Based Numbers

- $throughput_{expected} = \frac{window}{bestRTT}$
- $throughput_{observed} = \frac{window}{movingRTT}$
- $backlog = bestRTT * (throughput_{expected} - throughput_{observed})$
- Here, backlog is then the number of additional packets added delay has 'stuck' in the link.
- If the packet backlog exceeds a threshold γ , delay-based throttling occurs.

Window Behavior Goals

- In order to set the throttling behavior of the delay-based window, the creators of CTCP decided what the overall behavior of CTCP should be and then set the *dwnd* behavior to 'fill in the gaps'
- When neither delay nor drops are occurring, the CTCP window expands exponentially:
- $window(t + 1) = window(t) + \alpha * win(t)^k$
- When loss occurs, multiplicatively decrease the window.
- $window(t + 1) = window(t) * (1 - \beta)$

Dwnd Fills in the Gaps

- Since dwnd is just filling in the gaps, $dwnd(t+1)$ is $window(t+1) - cwnd(t+1)$
- So, with backlog $< \gamma$,
- $dwnd(t + 1) = dwnd(t) + (\alpha * win(t)^k - 1)$
- When loss is detected,
- $dwnd(t + 1) = (win(t) * (1 - \beta) - \frac{cwnd}{2})$
- Finally, some new behavior has to be defined for when backlog $> \gamma$, but no loss has yet occurred.

CTCP Under Delay Conditions

- CTCP needs to back off under delay conditions proportionally to the amount of 'backlog' it estimates in the link.
- Thus, when $\text{backlog} > \gamma$,
- $\text{dwnd}(t + 1) = \text{dwnd}(t) - \zeta * \text{backlog}$
- The dwnd cannot go below zero, so in extreme delay conditions, CTCP degrades down to its cwnd behavior, TCP Reno.

CTCP Implementations

- CTCP is the default implementation in Windows systems, beginning with Windows Vista and Windows Server 2008.
- A Linux implementation was added to the kernel, but no longer compiles in versions 2.6.17 or later, due to changes to the TCP stack implementation.
- The closed nature of Windows and the current state of the Linux implementations means there may be no open source implementation of CTCP available.

Windows CTCP

- The Windows CTCP implementation, owing to the closed nature of Windows, is itself proprietary.
- The original version appearing in Windows Vista, however, was written by the paper authors, and the original version of it was used for testing in the paper.
- A few implementation optimizations are suggested in the paper:
 - Sampling only M RTTs per segment, where M is proportional to the RTT itself, since TCP flows only change their sending rate as their RTT
 - Setting ' k ' to be $\frac{3}{4}$ instead of $\frac{5}{6}$, as it is faster to calculate.

Linux CTCP

- The implementation of CTCP in Linux (2.6.16) is similar, but not identical to the original CTCP paper.
- Has parameters alpha, beta, gamma, and zeta as in the paper.
- $\alpha=3$, $\beta=1$, $\gamma=30$, $\zeta=1$
- K (the exponent in cwnd growth) is defined at a fixed 0.75.

Linux CTCP

- The current RTT value is not set based on a moving average, but rather is the last seen RTT.
- Additionally, RTTs are sampled. This is an optimization mentioned in the original CTCP paper to reduce packet handling overhead.
- The CTCP paper recommended $k = 5/6$ (based on emulating the HSTCP response slope). The linux implementation uses 0.75 as an approximation, allowing the use of Newton-Raphson for quartic roots.

CTCP In NS-3

- The ns-3 simulator is the latest version of the ns family of network simulators originally created and used in the RED paper.
- Many newer algorithms (some of which were implemented in ns-2) are not yet implemented in ns-3.
- CTCP is one of the algorithms not yet implemented; Cubic was only recently implemented in ns-3.

Which Version of CTCP?

- Since both the implied implementation of CTCP on Windows and the verifiable implementation of CTCP on Linux are similar in their modifications from 'paper CTCP', both should provide similar performance.
- The Linux version of CTCP was thus used as the primary implementation reference for CTCP in ns-3.

Implementation in ns-3

- The underlying TcpSocketBase class in ns-3 provides the core TCP functionality.
- TcpSocket provides NewAck() and DupAck() virtual methods to override to modify the cwnd, dwnd and window parameters.
- TcpSocket also provides the slow-start functionality that CTCP uses.

Implementation in ns-3

- NewAck() is called from the lower layers whenever a new acknowledgement is received.
- When this happens, we can increment our cwnd value according to additive increase.
- We also update the RTT estimates and IncreaseDwnd() or ThrottleDwnd() as necessary.
- DupAck() is called whenever a packet has been dropped.
- When this happens, cwnd is halved and dwnd is reduced by the $(1-\beta)$ factor.

Verification

- In order to verify the correctness of the implemented algorithm in ns-3, it should be tested against the behavior of real-world CTCP implementations – Windows and Linux.

Questions?
