TCP flavors for mobile and high-speed environments

Freeze-TCP, TCP-Probing and Compound TCP

Giovanni Mazzocchin

Wireless Networks
Department of Mathematics

Università degli Studi di Padova

September 20th, 2018

Università degli Studi di Padova

Outline

Scenario

Freeze-TCP

TCP-Probing

Compound TCP

Scenario

UNIVERSITÀ DEGLI STUDI DI PADOVA

- **TCP** wasn't designed to cope with mobility issues and high-speed networks.
- Several enhancements to standard TCP have been devised:
    - mobility and power consumption is now taken into account (*I-TCP*, *M-TCP*, *Freeze-TCP*);
    - some enhancements tackle problems posed by high-speed and long-distance networks (*FAST TCP*, *Compound TCP*).

Previous approaches

- Many approaches involve base stations in flow and congestion control. Besides, they often split the connection (*I-TCP*, *M-TCP*, *Snoop* etc...).
- **Freeze-TCP** is an end-to-end scheme and doesn't require the involvement of any intermediaries for flow control.
- In order to achieve *Freeze-TCP*'s goals, changes in TCP code don't go beyond the mobile client.

Standard window management

- When the receiver advertises a zero-size window, the sender enters the **persist mode**:
  - *Zero Window Probes* are sent until the receiver's window opens up;
  - eventually, the receiver sends back a non-zero window size, and the sender will open its sending window.

Issues in mobile environments

- Even if a single packet is dropped due to a short disconnection, standard TCP wrongly thinks that the loss was caused by congestion and chokes the transmission.
- Thus, standard TCP's sender unnecessarily holds back, (slow window growth), even though the receiver often recoups quickly from a short disconnection.

Freeze-TCP approach (1)

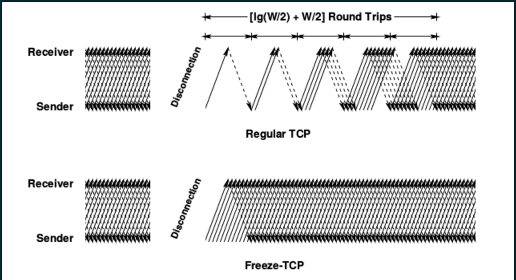
- The mobile client should signal any impending disconnection:
  - this is done via **signal strength monitoring**;
  - after detecting a disconnection:
    1. the mobile client advertises a zero window size;
    2. the sender switch to *ZWP* mode and doesn't shrink its window.

Freeze-TCP approach (2)

- How much in advance of the disconnection should the receiver advertise a window size of zero?
- There should be a *warning period* prior to disconnection.
- They figured out that a sensible choice is the *Round-Trip-Time*:
  1. if it is too long, there will be idle time prior to the disconnection;
  2. if it is too small, the sender's window could drop due to packet losses.

Freeze-TCP approach (3)

- A relevant issue: the *ZWP*s are exponentially backed off, so there could be an idle time after a reconnection.
- Trick: soon after the reconnection, the receiver sends 3 copies of the ACK for the last data segment it received before the disconnection (**TR-ACKs**).



Freeze-TCP approach (4)

Università degli Studi di Padova

It can be shown that the (approximate) number of extra packets transferred by *Freeze-TCP* is given by:

$$\frac{W^2}{8} + WlgW - \frac{5W}{4} + 1$$

Open issues

- In order to apply this protocol, the network stack should be aware of mobility.
- Is it reasonable to restart transmission at the full rate with the old window size upon entering a new environment?
- The receiver must predict impending disconnections.

Improving energy efficiency

- Energy efficiency is becoming paramount in communication protocols (a cross-layer issue).
- The error control mechanism should be friendly both to **throughput** and **low power consumption**.
- Energy-conserving capabilities in standard TCP:
  after segment drops, it shrinks the window so as to save transmission effort.

Improved Error Recovery for better Energy Efficiency

- The Error Recovery mechanism is not always efficient: in fact, standard TCP thinks that packet losses always happen due to *congestion*.
- In case of *infrequent* and *transient* errors, standard TCP strategy leads to:
  1. unneeded effective throughput degradation;
  2. increase in overall connection time.
- Moreover, monitoring network conditions only by means of packet losses causes major energy wastage.

TCP-Probing approach

- In **TCP-Probing**, when a segment is dropped, the sender initiates a **probe cycle** (described later).
- The *probe cycle*'s duration is naturally extended according to the error condition.
- Random losses trigger short probe cycles.

Probe Cycle (1)

- **Immediate Recovery**:
  if network conditions detected when the probe cycle is over are acceptable, the protocol simply restarts from the state before the timeout event.
- Otherwise, TCP-Probing opts for *Slow-Start* (conservative approach).

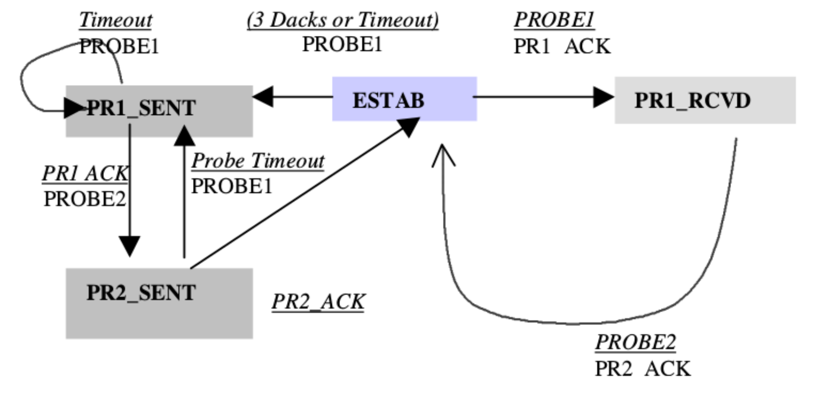
Probe Cycle (2) - Implementation

1. The sender transmits *PROBE1*, to which the receiver immediately responds with *PR1_ACK*. After receiving the latter the sender transmits *PROBE2*.
2. The receiver acknowledges this second probing with a *PR2_ACK* and returns to the *ESTAB* state.
3. A critical part of the probing mechanism is the protocol's behavior at the end of the cycle.

Probe Cycle (3) - Implementation

- A *measurement timer* is used to measure the two RTTs from the probe cycle.
- Upon exiting the probe cycle, the two measured RTTs are compared.
    1. if both lie in the range `[best RTT, last RTT]`, *Immediate Recovery* is applied;
    2. otherwise, the sender enters a *Slow-Start* phase.

Probing State Transition Diagram



Open issues

- Probe cycles cause more transmission effort.
- The decision-making criteria are conservative because *Immediate Recovery* is entered at the end of probing only in some cases.
- Simply stated, *TCP-Probing*'s behavior is insufficiently aggressive.

Issues in High-speed and Long Distance networks

- Standard TCP can't fully utilize the network capacity because of its conservative approach.
- In **Compound TCP**, *delay-based* and *loss-based* approaches coexist.
- In *Compound TCP*, a scalable delay-based component is plugged into the *TCP Reno* congestion avoidance algorithm, which is loss-based.

Background

- In a **high-speed** and **long delay** network, only a very large window can fully utilize the link capacity.
- In real-life networks, a standard TCP sender may never open its window enough to leverage the high-speed resource.

Loss-based vs Delay-based

- **Loss-based** strategies modify the increase/decrease parameters in order to become more aggressive. Pitfalls:
  1. an aggressive behavior causes more packet losses on bottleneck links;
  2. the throughput of the regular TCP flows is pushed back.
- **Delay-based** strategies base their decisions on RTT variations (e.g. *FAST-TCP*). They have:
  1. higher utilization;
  2. less self-induced packet losses;
  3. better RTT fairness and stabilization.

The Compound TCP (1)

- Pure delay-based approaches are not competitive to loss-based approaches.
- This happens because they reduce their sending rate when bottleneck queue is built. However, this behavior will make loss-based flows increase their sending rate since they notice less packet losses.

The Compound TCP (2)

- *Compound TCP* incorporates a scalable delay-based component into the standard TCP congestion avoidance algorithm.
- Delay-based component's features:
  1. rapid window increase rule when the network is under-utilized;
  2. it reduces the sending rate once the bottleneck queue is built.

The Compound TCP (3)

- A new scalable delay-based component in the TCP congestion avoidance algorithm is added.
- A new state variable is introduced: `dwnd` (*Delay Window*), which controls this delay-based component.
- The `cwnd` remains the same, controlling the loss-based component.
- Thus, the sending window is controlled by both `cwnd` and `dwnd`.

The Compound TCP (4)

- `Sending window = min(cwnd + dwnd, awnd).`
- The `cwnd` is updated in the same way as in regular TCP's congestion avoidance.
- The *Slow-Start* behavior of regular TCP is kept at the start-up of a new connection. In fact, *Slow-Start* is quick enough also for fast and long distance environments.
- The delay-based component comes into play in the congestion avoidance phase.

Delay-based component design (1)

- A state variable (`baseRTT`) is maintained as an estimation of the delay of a packet over the network path.
- A the start of a connection, `baseRTT` is updated to the minimal observed RTT.
- An exponentially smoothed current RTT (`sRTT`) is also maintained.

Delay-based component design (2)

The number of backlogged packets can be estimated by means of these formulas:

1. `expected = win / baseRTT`;
2. `actual = win / RTT`;
3. `diff = (expected - actual) * baseRTT`.

An early congestion is detected if the number of packets in the queue is larger than a fixed threshold γ (*diff* $> \gamma$).

Delay-based component design (3)

1. Without packet losses:

$$win(t+1) = win(t) + \alpha * win(t)^k$$

2. With packet losses:

$$win(t+1) = win(t) * (1 - \beta)$$

Parameters α, β and k should be tuned.

Delay-based component design (4)

The delay-based component `dwnd` is updated following the rules below.

$$dwnd(t+1) =$$

$$\begin{cases} dwnd(t) + (\alpha * win(t)^k - 1)^+, diff < \gamma \\ (dwnd(t) - \zeta * diff)^+, diff \geq \gamma \\ (win(t) * (1 - \beta) - cwnd/2)^+, loss \end{cases}$$

Open issues

- The γ parameter could be set adaptively.
- Early congestion should be detected by means of constant buffer requirements regardless of the number of *CTCP* flows.

Bibliography

UNIVERSITÀ DEGLI STUDI DI PADOVA

- T. Goff, J. Moronski, D.S. Phatak, V. Gupta, *Freeze-TCP: a true end-to-end TCP enhancement mechanism for mobile environments*.
- V. Tsaoussidis, H. Badr, *TCP-probing: towards an error control schema with energy and throughput performance gains*.
- Kun Tan, Jingmin Song, Qian Zhang, Murari Sridharan, *A Compound TCP Approach for High-speed and Long Distance Networks*.

Thank you

Any questions?