Towards Handoff Tolerance in TCP: Link Layer Handoff Detection for Optimized Data Transport

Vishwa Goudar, Michael Cohen, M.Y. Sanadidi, Mario Gerla
Computer Science Department
University of California, Los Angeles, CA

Francesco Zampognaro
Department of Electronics Engineering
University of Rome, Italy

ABSTRACT

With the proliferation of network management tools that adjust link and path characteristics to optimize network performance and health, TCP’s are frequently incorporating path characteristic detection mechanisms to quickly adapt to the effects of these changes. Optimal TCP performance, including rapid adaptability, depends on, among other things, an accurate view of the Bandwidth-Delay Product (BDP) of the path. Towards the goal of prompt detection of large changes in the BDP, we propose a detection scheme that can be applied as a sender-side only modification. Specifically, we focus on link layer (L2) handoffs at the bottleneck link of the path. Simulation results show that current TCP variants, such as TCP-Reno and TCP-Westwood, are unable to efficiently capitalize on L2 handoffs at bottleneck links that result in substantial changes in the BDP. Such a handoff can occur, for example, during a TCP session over a multi-hop path involving a bottleneck link between two aerial vehicles, wherein a degradation of this link causes a handoff to a backup satellite relay. By making use of BDP and congestion heuristics, our mechanism detects such BDP changes and allows for quick adaptation to the observed change, thereby efficiently addressing this issue. Based on simulated experiments, we observed rapid detection of handoff (up to 7 times faster that TCP Reno), including fairly accurate estimation of the changes in BDP.

1. INTRODUCTION

The ubiquity of the internet and the surge in related applications that require reliable and in-order delivery of data has engendered the Transmission Control Protocol (TCP) a de-facto standard for many types of data communication. Further, the maturity and pervasiveness gained by network management tools that achieve optimal performance by monitoring the health of the network, has allowed networks to balance loads effectively under a variety of constraints, by choosing the most promising paths for data flows. TCP performance, under constraints presented by the recalibrating network, is required to optimize capacity utilization, while simultaneously affording an effective congestion control/avoidance mechanism, as well as providing fairness among traffic flows sharing the network. Posed as an optimization problem, TCP must function under a variety of trade-offs. As a result, it has undergone a variety of modifications and iterative improvements giving rise to different variants that achieve optimal performance under varied assumptions regarding network parameters, behavior and traffic characteristics.

An important component of a TCP’s performance is how well it maximizes channel utilization, while maintaining fairness. The amount of data that a TCP source puts on the channel depends on its view of the BDP of the path traversed by the data, wherein the BDP is the product of the available bandwidth, BW, and delay offered by the path. While we would like TCP to rapidly adapt to significant changes in the BDP offered by the path, current TCP variants suffer high delays until they fully adapt to the change. These delays are proportional to the extent of the change due to the underlying AIMD mechanism of TCP. Therefore, substantial changes in the BDP lead to larger stabilization delays. Although work exists to adjust to change in network conditions, these efforts focus on congestion as the source of the change. A large body of work also exists that allows TCP to quickly stabilize given that the network conditions are unknown [8]. To bridge the gap, we would like to present a solution towards the detection of a change in network conditions. Specifically, we focus on large changes in the BDP, caused by link layer handoffs. We would like this mechanism to work as a sender-side only modification, with no cross-layer signaling or optimization.

We view detection of handoff as a ternary classification problem, wherein we would like to distinguish between no change in path characteristics, a change in the BDP, and a change in the congestion level. Further, when a change in the BDP is detected, we would like to estimate the level of change to allow an appropriate rescaling of TCP parameters. This detection mechanism must provide high sensitivity and specificity in distinguishing between a change in BDP and a change in congestion, while doing so as soon as possible. Finally, we would like it to be robust to a variety of congestion models, including path-persistent and non path-persistent congestion.

Towards this goal, we make use of two BDP heuristics and one congestion heuristic. When neither changes, we
classify this as no change in path characteristics; when the BDP heuristics change without a change in the congestion heuristic, it is classified as a change in BDP; and finally, when the congestion heuristics change with/without changes in the BDP heuristics, it is classified as a change in congestion levels. On detection of a BDP change, the BDP heuristics also provide an estimate of the level of this change. There is much to be gained by TCP, in terms of efficiency in throughput, from the detection of and adaptation to abrupt changes in the BDP.

2. BACKGROUND

2.1 Related Work

Whereas vertical handoffs involve a change in the channel medium, a horizontal handoff involves no change to the link layer medium. Since horizontal handoffs, such as a change in WiFi access points, don’t yield large changes in the BDP, we choose not to focus on this problem. It has, however, been extensively studied. In [3], the authors study the impact of horizontal handoff on TCP performance and observe the confusion over packet drops resulting from handoff and those resulting from congestion. They propose a cross-layer signaling mechanism to allow for TCP to revert to slow start on handoff. TCP Snoop [4], on the other hand, improves TCP performance in wireless networks by caching packets at each packet forwarder, to ensure retransmissions in the face of packet drops due to the lossy nature of the channel. Horizontal handoffs are handled by transferring caches from old to new packet forwarders. Similarly, S-MIP [5] and I-TCP [6] optimize for lossy link and mobility related inefficiencies at lower layers of the network stack.

In [2], TCP optimizations for vertical L2 handoffs focus only on handoffs at the TCP sender’s outbound link, with cross-layer signaling to detect the handoff and freezing of TCP parameters to overcome instability during the change. Other works deal with the more general problem of vertical L2 handoffs at any link along the multi-hop path between the TCP sender and receiver, with cross-layer optimizations such as ECN and FEC mechanisms to detect and overcome the effects of handoff-related packet loss [12], or, explicit notification to TCP from the link experiencing handoff. We would like to address the generalized version of the problem that yields substantial changes in the BDP, while doing so both without cross-layer optimizations and as a sender-side only modification. By being able to estimate the level of change, our work should also allow for faster resolution.

Our work makes no distinction between hard handoffs, where one channel is completely severed before a separate channel is used for transmission, and soft handoffs, where both channels are used in parallel for a brief time during the transition. However, it is noteworthy that hard handoffs can cause the loss of a large number of packets and increase the time-to-detection of handoff.

Poor startup performance is identified as a crucial problem in large BDP networks. It arises from the lack of knowledge, at startup, of the optimal congestion window size ($cwnd$) or slow start threshold ($ssthresh$). To cope with a large BDP, [7] proposes a static solution wherein the initial congestion window is set to a large value when the BDP of the path is known apriori to be large. Commercial satellite data communication providers cope with the large BDP offered by the satellite link in a similar fashion, by using very large initial windows (LIW). Although these solutions mitigate the effects of a large BDP on startup performance, the authors of [8] show that such solutions may still be inefficient in some large BDP scenarios. They propose ASTART as an improved solution, wherein $cwnd$ is increased in mini linear and exponential phases during startup, to achieve optimal performance much sooner. Given our mechanism, a TCP that implements ASTART, such as TCP-AW [9], may utilize our detection mechanism to revert to ASTART after handoff detection.

2.2 Effects of Vertical L2 Handoff on TCP Performance

In order to examine the behavior and properties of TCP sessions over a vertical handoff, we implemented a handoff link abstraction in the ns-2 network simulator [1]. An L2 handoff is simulated in our abstraction, by switching, via a soft or hard handoff, between two component links that have distinct and independent properties such as bandwidth, delay and buffers.

![Figure 1: Network that may exploit a L2 Handoff](image)

Figure 1 depicts an example scenario, wherein a vertical L2 handoff occurs in the form of a change from a wireless point-to-point link, e.g. between C and E, to a link over a GEO Satellite relay. Consider a TCP session between C and E with the point-to-point wireless link supporting 300Kbps with a propagation delay of 10ms. Transmission over the satellite relay, on the other hand, supports 1Mbps at a delay of 500ms. Simulating a TCP-Reno session comprised of an FTP flow of 500B packets from C to E on ns2 for a period of 1500 seconds, with a
hard handoff to the relay link at 100 seconds, and a hard handoff back to the point-to-point link at 1400 seconds, we observe the congestion window and slow start threshold depicted in Figure 2. Over each hard handoff, the link stays severed for 0.2 seconds.

**Figure 2: cwnd and ssthresh reactions of TCP-Reno to L2 Vertical Handoffs**

Assuming no processing delays, each packet in this scenario experiences an end-to-end latency of 23.33ms, due to propagation and transmission delays, on the lower BDP path, and 504ms on the higher BDP path. This translates to BDPs of 875B and 63KB, respectively. Given TCP-Reno’s AIMD method of increasing \( cwnd \) by 1 for every Round Trip Time, or RTT, TCP-Reno would require about 200 seconds to adjust \( cwnd \) to the 62KB change in the BDP, on handoff to the higher BDP link. This is precisely what we observe in figure 2. Further, TCP-Reno would experience a delay in adjusting the re-transmission timeout, \( RTO \), to the abrupt change in the RTT. In our sample scenario this delay is observed to be 5 seconds long. In the meanwhile, depending on the level of congestion, the retransmission timer may timeout a lot sooner than it should resulting in un-necessary retransmission and unwanted, incorrect or slow adjustments to \( cwnd \) and \( ssthresh \). It is also assumed that link instability that results in handoff, but is commonly mistaken for congestion by most TCPs, is not observed for a very long period before the decision to handoff is taken. This period is related to the cycle length, defined in section 3.2, and the reason for this assumption is clarified in section 3.3. Finally, although it is not required for the detection mechanism to work correctly, we do not incorporate the lossy nature of wireless links in our simulations.

**Table 1: Handoff Classifier Decisions**

<table>
<thead>
<tr>
<th></th>
<th>Handoff</th>
<th>Congestion</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>AckRate Change</td>
<td>Yes+</td>
<td>Yes/No</td>
<td>No</td>
</tr>
<tr>
<td>RTT Change</td>
<td>Yes+</td>
<td>Yes/No</td>
<td>No</td>
</tr>
<tr>
<td>Jitter Change</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

+ either ackRate/RTT/both change

3. **HANDOFF DETECTION**

As stated earlier, we view the handoff detection as a ternary classification problem, wherein a TCP event is classified as a handoff, a change in congestion levels, or, no change. In solving this problem, we assume that a handoff and change in congestion levels cannot occur simultaneously. In other words, there must be a significant delay between the change in congestion levels and a handoff. It is also assumed that link instability that results in handoff, but is commonly mistaken for congestion by most TCPs, is not observed for a very long period before the decision to handoff is taken. This period is related to the cycle length, defined in section 3.2, and the reason for this assumption is clarified in section 3.3. Finally, although it is not required for the detection mechanism to work correctly, we do not incorporate the lossy nature of wireless links in our simulations.

### 3.1 Heuristics

We select a direct and easily measurable set of heuristics to ascertain current and stable measures of bandwidth, delay and congestion. We use an approach, similar to TCP Westwood's path characteristics estimation techniques [10], to estimate bandwidth by measuring the rate of returning acknowledgements (ackRate). In other words, the ackRate is measured as the reciprocal of the time between consecutive acknowledgements. We estimate congestion by the jitter observed in the rate of acknowledgements. Note that jitter is calculated as the difference between the current and previous measurement. AckRate jitter appears to be a good indicator of congestion, both logically and experimentally. When congestion levels change, the time between acknowledgements will fluctuate based on the distribution of the cross-traffic. With the help of the ackRate and RTT heuristics, we are able to detect changes in BDP caused by a change in bottleneck RTT takes some time to reflect in the RTO, resulting in sub-optimal delays before recovering from packet losses.

Clearly, a TCP stands to benefit significantly through the detection of a handoff event, estimation of the new BDP, and the recalibration of its \( ssthresh \) and \( RTO \) to the new conditions.
bandwidth and/or delay, as well as, characterize the direction and level of change. Based on the concurrence of the detection of such a change, with a change in congestion levels, as observed by the jitter heuristic, we are then able to attribute the change in BDP to congestion or to a handoff. Table 1 outlines the decisions of our classifier.

When handoff is detected, the heuristics allow us to estimate the level and direction of change in BDP that results from the handoff. Note that the ackRate and RTT are proportional to the bandwidth and delay respectively. Similarly, ssthresh is supposed to approximate the BDP share of a connection. Therefore, the new ssthresh may be estimated based on the observed change as follows:

$$ssthresh_{new} = ssthresh_{old} \times \left( \frac{RTT_{new} \times ackRate_{new}}{RTT_{old} \times ackRate_{old}} \right)$$  (1)

The RTO may also be reset in a similar fashion as a function of the new RTT. From table 1, it is also clear that our classifier must work under the assumption that congestion levels may not change at the same time that a handoff occurs. A handoff to a higher bandwidth link, with a similar propagation delay as the lower bandwidth link will lead to similar results. Therefore when handoff to a higher bandwidth link is combined with a decrease in congestion levels, the RTT and jitter may not provide much information as to the cause of the observed change. With no additional information, the two cases are difficult to distinguish. In such cases, therefore, our detection mechanism produces a false negative leaving TCP to readjust with its conventional mechanisms.

3.2 Averaging Heuristics

The classifier outlined in table 1 requires stable and accurate measures of the heuristics described above. But raw data for each of the required heuristics tend to fluctuate and may be distorted. A major source of such distortion comes from retransmitted packets. It is impossible to know whether an acknowledgement for a packet comes in response to the original packet or to one of the retransmissions. It is precisely the effects of this unreliability that Karn’s algorithm [11] serves to mitigate. In a similar fashion we only consider acknowledgements for non-retransmitted packets while collecting heuristic data. Note that this delays the time-to-detection as packet loss is common in hard handoff cases.

The collected heuristic data also fluctuates at steady state, based on the path characteristics and congestion levels. For example, under conditions of low to no congestion, the RTT tends to fluctuate between a minimum and maximum value characterized by the bottleneck buffer size. Congestion causes further fluctuation in all required heuristics due to the unpredictable traffic profiles of concurrent data flows. While TCPs use a smoothing mechanism in the form of exponential moving averages to mitigate these outliers, these smoothers do little to reduce the fluctuation observed. Our classifier solves this problem by averaging the heuristic over a “natural” cycle. A stable but accurate measure is obtained by averaging the heuristic over a dynamic interval that is based on the periodicity of the observed heuristic. To avoid ambiguity with cwnd, we refer to this interval as the cycle. To reduce computation, and without loss of accuracy, we compute the cycle length based on the periodicity of the RTT.

$$R(t, t-\tau) = \frac{E[(RTT_t - \mu)(RTT_{t-\tau} - \mu_{t-\tau})]}{\sigma_t \sigma_{t-\tau}}$$  (2)

where $E$ is the expected value operator. Given a history of $n$ RTT values, $(RTT_1, ..., RTT_n)$, whose mean and variance are computed as $\mu$ and $\sigma$, respectively, the $acf$ of $RTT_t$, at a lag $\tau$, $R(t, t-\tau)$ is defined as:

The $acf$ estimates for an experimental scenario described in section 4, is shown in Figure 3.
This scenario involves a TCP session between nodes 1 and 8 in the topology in figure 4. No congestion/cross traffic is simulated and the \( \text{acf} \) estimates are taken 10s into the session. Note that the estimates are high at very short lags, since consecutive RTT measurements are usually quite similar to each other. The dotted horizontal lines at \( \pm 0.013 \) represent the 99% confidence intervals of the estimates. The cycle length is then calculated from this function, based on lag at which we encounter the highest positive autocorrelation after the estimate has become negative. We must overlook small lags due to similarity of consecutive measurements and, therefore, wait until the \( \text{acf} \) estimates are negative. This happens at lag 923. We also require the highest \( \text{acf} \) estimate after lag 923 to be above the confidence interval. This happens at lag 2837 and is chose as the cycle length. Therefore, at time 10s, the average measure of a heuristics will be calculated as the average over the last 2837 measurements. Figure 5 shows the results of this method over the experimental scenario. We ran a TCP session between nodes 1 and 8, in figure 4 for 250s. We introduce cross traffic in the form of TCP sessions at times 25s to 75s, 100s to 150s, and, 175s to 225s. Figure 5(a) depicts the raw RTT measurements in grey and the corresponding average values in black; the corresponding cycle lengths are shown in figure 5(b). We observe that with path persistent congestion at times 25s to 75s, and, 100s to 150s, the congestion causes \( cwnd \) to reduce significantly, causing shorter cycles with higher variance. In the case of path non-persistent congestion at time 175s to 225s, the variance does not increase by the periodicity does, as can be observed from figure 5(a). We also see that due to the average over cycles, as with any smoothing function, there is a delay before changes are detected.

While averaging the heuristics over cycles makes them more stable, the cycle computation must be performed over a sizeable history. Maintaining a history involves buffer space, and hence it is important that we do not waste this resource. In table 2, we outline our cycle computation algorithm. Here, \( n \), is the length of history maintained for accurate calculation of the cycle. At time \( t \), the cycle is calculated by:

\[
\text{for } k \in \{n, n/4, n/2, 3n/4\} \\
\text{a = acf(\{RTT}_{t-k},...\;RTT_{t}\})} \\
l_k = \text{lag([a is maximum) and (a > conf. interval) and (lag > lag at which a first goes \(-ve\)])} \\
m_k = a[l_k] \\
\text{if } l_k \text{ exists for some } k \\
\text{cycle = } t - l_k \\
\text{else} \\
\text{cycle = cycle + 1} \\
\text{if } n > (\text{cycle} \times hf) \\
n = \text{cycle} \times hf
\]

where \( hf \), or history factor, is a tunable parameter that adjusts the length of history as a factor of the cycle length. We believe it should be greater than 3 to provide the \( \text{acf} \) with sufficient history. In our experiments, we set it to 3.5.

### 3.3 Detection

While the computed cycle allows for a robust and stable measure of the required heuristics, we would like to compute the average in a way that is unaffected by outliers, yet responds quickly to changes in network behavior. Calculating stable heuristics as the mean over the...
We evaluate our detection mechanism using ns-2, with the topology depicted in figure 4. An FTP session is run between nodes 1 and 8 on TCP-Reno. Vertical handoff is simulated as a change in the component links of the handoff link abstraction between nodes 4 and 5. We evaluate three scenarios in this situation: (a) a handoff to a higher bandwidth link at 15s and back down to the lower bandwidth link at 45s, with no change in delay and no cross-traffic; (b) a handoff to a higher delay link at 15s and back down to the lower delay link at 45s, with no change in bandwidth and no cross-traffic; and, (c) an increase in congestion levels due to cross-traffic without handoff, the details of which are outlined in section 3.3 above. The congestion at time 25s to 75s is forward and reverse traffic; at time 100s to 150s is only reverse traffic; and, at time 175s to 225s is non path-persistent forward and reverse traffic.

Figure 6 outlines the averaged heuristics in black, and the raw heuristic values in grey. The figures in column 1 are the ackRate, RTT and jitter heuristics for scenario (a); columns 2 and 3 correspond to scenarios (b) and (c), respectively. From column 1, we observe that the change in ackRate, on handoff to the higher BW link, is detected almost immediately. Without a change in the jitter by the end of the cycle, the ackRate change is detected in 1.21 seconds. While the new ackRate changes 10 times, the new RTT drops by half, leading to a 500% increase in the BDP. This is verified by a similar change in ssthresh produced by Reno, from 43 to 203, which requires 6.92 seconds. We are able to detect the handoff and estimate the new ssthresh almost 5 times faster. On handoff back to the lower BW link, the ackRate and delay heuristics detect the event in 3.84 seconds, just shy of the 4 seconds required by Reno to adjust to the new conditions.
Handoff to a higher delay link in column 2 of figure 6, takes the delay heuristic 5.73 seconds to detect. In this scenario, however, Reno is unable to arrive at a new ssthresh value. Re-running scenario (b) for a longer duration on the higher delay link shows that Reno only finds the new ssthresh value 42 seconds after the handoff. By this time, in the original scenario, a handoff to the lower delay link has already occurred. Our technique arrives at an ssthresh estimate over 7 times faster than Reno! Our estimate of the change in BDP, and hence the change in ssthresh, is a factor of 4.63. But Reno reaches steady state after an increase by a factor of 5.8. Despite the inaccuracy, we believe that it is better for us to under-estimate the change than over-estimate it. Further, while it takes Reno 8 seconds to readjust ssthresh on handoff back to the lower delay link, we are able to estimate the same in a little less than 2 seconds.

Finally, in the scenario with changing congestion levels and no handoffs, we correctly classify the changes in all 3 cases. For these experiments, we set cp to 10%. With the forward and reverse path persistent cross traffic, we observe a 13.6% decrease in RTT, with no significant change in the ackRate and a 1511% increase in jitter. These changes in RTT and jitter are observed within 0.27 seconds of each other, which turns out to be less than half a cycle length. In the case of reverse-only path persistent cross-traffic, we observe no significant change in the RTT, but a 46.1% drop in the ackRate, concurrent with a 24% increase in the jitter. Finally, in the case of non-path persistent congestion, we observe a 54% decrease in the RTT with no significant change in the ackRate and an 88% change in the jitter. These are observed within 0.86 seconds of each other, also less than half a cycle length.

5. FUTURE WORK

5.1 Post-Handoff Optimizations

We would like to augment TCP-AW with our detection mechanism and evaluate the resulting improvement in goodput. This would also involve a change in its behavior on detection of handoff. Specifically, we would have to reset TCP’s state variable and clear any explicit or implicit history. The most relevant variables are the ones that affect the retransmission timer, RTO, and the smoothed RTT, srtt. These must be reset to correspond to the newly estimated values of RTO and srtt. The task would also involve inducing a tempered growth of cwnd with the ASTART mechanism, to rapidly arrive at the optimal ssthresh without causing buffer overflows. Note that abruptly resetting cwnd and ssthresh to the newly estimated values, in cases of an increased BDP, would cause too many packets to be sent out simultaneously, causing TCP buffers to overflow.

5.2 Analysis

We would also like to analyze the proposed mechanism and the related modification to TCP-AW on multiple fronts. First, we would like to measure the sensitivity and specificity of our technique and study their implications. Next, we would like to compare our mechanism to those resulting from cross-layer signaling to fully assess the tradeoff. Finally, we would like to relax the assumption that there will be no simultaneous change in congestion levels and handoff. Such a mechanism is under investigation.

6. CONCLUSION

In this paper, we develop and experimentally evaluate a mechanism to detect vertical L2 handoffs that result in large BDP changes, as well as estimate the level of BDP changes. Such a mechanism is shown to be relevant to TCP’s seeking improved goodput and reaction times in the face of handoffs.

REFERENCES