10. Conclusions

Below, we provide conclusions in two general categories: conclusions (Sec. 10.1) about the congestion-control algorithms we studied and conclusions (Sec. 10.2) about the methods we applied. Along with each set of conclusions we also provide suggestions for related future work.

10.1 Conclusions about Congestion-Control Algorithms

The simulation and modeling studies reported here enabled us to draw a range of conclusions about the general utility and safety of seven proposed alternate congestion-control algorithms for the Internet. We were also able to characterize each of the congestion-control algorithms we studied. In the end, we developed some recommendations about whether it makes sense to deploy alternate congestion-control algorithms at large scale on the general Internet. Finally, though our study is quite comprehensive, we recognize the need for future work to investigate some questions that we did not tackle. We address these topics, in turn, below.

10.1.1 Utility and Safety of Alternate Congestion-Control Algorithms

Our simulation and modeling experiments showed that deploying alternate congestion-control algorithms can provide improved user experience under specific circumstances. As discussed below, the nature of such circumstances bound the utility that alternate congestion-control algorithms may provide. In addition, the experiments showed that some proposed algorithms can be deployed without driving large changes in macroscopic behavior throughout a network. On the other hand, other proposed algorithms altered behavior in undesirable directions under specific spatiotemporal situations. We address these topics in detail.

10.1.1.1 Increase Rate. One of the key questions for any data transport protocol is: How fast can the maximum available transfer rate be achieved? Assuming no congestion (i.e., no losses) protocols that can quickly converge to the maximum rate will spend the largest portion of a file transfer at that rate. Each TCP flow begins without any knowledge of the maximum available transfer rate. For this reason, TCP specifies an initial slow-start process where the source transmits slowly but then, as feedback arrives from a receiver, quickly increases the transmission rate until reaching a specified (initial slow-start) threshold or encountering a loss. This initial slow-start process is not altered by any of the proposed alternate congestion-control algorithms that we studied.

Assuming no (or low) congestion, the setting of the initial slow-start threshold can be quite important when comparing goodputs experienced by users on TCP flows with goodput for users on flows operating under alternate congestion-control algorithms.[1] When the initial slow-start threshold is set arbitrarily high, on average all flows achieve

[1] Note that in real TCP flows receivers may convey a receiver window (*rwnd*) that can restrict goodput quite severely because sources pace transmission based on the minimum of the congestion window (*cwnd*) and *rwnd*. Typically, the following holds: *rwnd* < *cwnd*. In our studies, we assume an infinite *rwnd* in order to compare the effects of congestion-control algorithms adjusting the *cwnd*. The goodput on many TCP flows in a real network might well be constrained by *rwnd*. In such cases, alternate congestion-control algorithms would provide little advantage over TCP congestion-control procedures.

maximum transfer rate with the same quickness. Under such situations, the goodput seen on TCP flows and flows running alternate algorithms appears quite comparable. Flows carrying short files (e.g., Web objects and document downloads) tend to complete while in initial slow-start, which means that alternate congestion-control procedures (restricted to the congestion-avoidance phase of a flow) do not operate. Even flows conveying long files can operate for extended periods under initial slow-start because such flows do not enter congestion-avoidance until encountering a loss.

When the initial slow-start threshold is set low (e.g., 64K Bytes) all of the alternate congestion-control algorithms that we studied increase transmission rate more quickly than the linear increase provided by the TCP congestion-avoidance phase. Thus, under low congestion, when the initial slow-start threshold is set low compared to the size of files transferred (and assuming the receiver window – *rwnd* – is not constraining transmission rate) users on TCP flows will see much lower goodput than users of alternate congestion-control algorithms. The larger the file sizes being transferred the larger the goodput advantage of the alternate algorithms. The alternate congestion-control algorithms provide different degrees of goodput improvement over TCP congestion-avoidance procedures. As discussed below (Sec. 10.1.2), these goodput differences can be tied directly to the speed with which the alternate algorithms reach the maximum available transmission rate.

Under conditions of heavy congestion the setting of the initial slow-start threshold matters less because initial slow-start terminates upon the first packet loss and then a flow enters the congestion-avoidance phase, which is where the alternate congestion-control algorithms differ from TCP procedures. In such situations, the main difference in goodput experienced by users relates to the loss/recovery procedures defined by the alternate algorithms. We turn to this topic next.

10.1.1.2 Loss/Recovery Processing. Two key questions arise when a data transport protocol experiences a packet loss. (1) How much should the protocol reduce transmission rate upon a loss? (2) How quickly should the protocol increase transmission rate after the reduction? TCP congestion-avoidance procedures reduce transmission rate by one-half on each packet loss. Subsequently, TCP congestion-avoidance procedures increase transmission rate linearly. The alternate congestion-control algorithms we studied specify various procedures for transmission rate reduction and increase following a lost packet.

One group of algorithms (Scalable TCP, BIC[2] and HSTCP) reduce transmission rate less than TCP after a packet loss. As a result, these algorithms tend to retain a higher transmission rate and associated buffers than is the case for TCP flows. Smaller rate reduction can allow these algorithms to provide established flows with higher goodputs following packet losses. We found this effect to increase with increasing loss rate and also file size. In addition, these algorithms can be somewhat unfair to algorithms (such as TCP) that exhibit a more reduced transmission rate following a loss, as well as to flows that have not had sufficient time to attain a high transmission rate prior to a loss.

[2] Note that on repeated losses occurring close in time, BIC can reduce *cwnd* substantially more than TCP congestion-avoidance procedures; thus, on paths with very severe congestion BIC can actually provide lower goodput than TCP and also occupy fewer buffers.

A second group of algorithms (CTCP, FAST and FAST-AT) reduce transmission rate in half following a loss. HTCP appears to be a hybrid, reducing transmission rate variably, mainly between 20% and 50%. The higher reduction occurs when transmission rate is increasing substantially in a round-trip time and the lower reduction occurs when transmission rate is less variable. To obtain higher goodput, these algorithms increase transmission rate more quickly than TCP flows following a rate reduction. As discussed below (Sec. 10.1.2), the rate of increase varies with the specific algorithm. Typically, HTCP and CTCP are less aggressive than FAST and FAST-AT when increasing transmission rate after a reduction. FAST-AT will be less aggressive when sufficient congestion exists to force a reduction in the α parameter. An aggressive rate increase following a rate reduction can induce additional losses. When such losses affect TCP flows, then linear recovery procedures lead to lower goodputs. Under severe congestion, CTCP and HTCP can provide better goodput than FAST and FAST-AT, which can underperform TCP.

In areas and at times of extreme congestion, most of the alternate algorithms we studied include procedures to adopt TCP congestion-avoidance behavior. These procedures appear motivated by the theory that when congestion is sufficiently severe then existing TCP behavior provides the best approach to fairly share the limited available transmission rate. The most typical technique employed is to set a low-window threshold. When the congestion window (*cwnd*) is below the threshold then TCP congestion-avoidance is used. When *cwnd* is above the threshold then alternate congestion-avoidance procedures are used. Specific values for the threshold vary among the alternate congestion-control algorithms. The combination of different thresholds and different file sizes can lead to modest differences in user goodputs.

HTCP handles adaptation to TCP procedures somewhat differently than most other alternate algorithms. After a loss, HTCP adopts linear rate increase for a time. The time period is an HTCP parameter, set in these experiments to one second. We found that HTCP then adapts to TCP linear increase after every loss, regardless of file size or *cwnd* value. For larger files, which tend to have higher *cwnd* and to experience more losses during transmission, this approach tends to lower goodput significantly relative to other alternate algorithms, which do not adopt linear increase after every loss.

FAST and FAST-AT do not use TCP congestion-avoidance procedures under any circumstances. In times and areas of heavy congestion, failure to adopt less aggressive rate increase can lead to oscillatory behavior and to an associated increase in loss rate. Increased losses lead to lower user goodputs. FAST-AT does somewhat better under heavy congestion because the α parameter can be lowered; this causes less aggressive rate increases. Still, under many conditions, FAST-AT can exhibit a similar increased loss rate to FAST.

10.1.1.3 Fairness. Comparing alternate congestion-control algorithms with respect to TCP fairness can be somewhat difficult because the alternate algorithms are designed to give better goodput than TCP for large file transfers on high bandwidth-delay paths. Thus, for example, all of the alternate algorithms can increase transmission rate more quickly than TCP given a low initial slow-start threshold and large file sizes. Further, all alternate algorithms take steps to provide loss/recovery improvements over the standard TCP congestion-avoidance procedures. On the other hand, most of the alternate

algorithms take steps to adopt TCP congestion-avoidance procedures when congestion is sufficiently high. Given these factors, one would expect all alternate congestion-control algorithms to provide better goodput than TCP under optimal conditions. In addition, most of the alternate algorithms are assured of performing no worse than TCP under suboptimal conditions. The usual measures of fairness do not apply in such circumstances because they would tend to measure how much of a goodput advantage a given alternate algorithm provides over TCP procedures. We measured fairness by ranking the average goodput achieved by TCP flows when they competed with each alternate congestion-control algorithm under the same conditions. We considered the average rank across four file sizes: Web objects, documents, software service packs and movies. In this way, we could tease out the relative TCP fairness of the alternate algorithms.

We found that CTCP and HTCP were most fair to TCP flows. We found FAST-AT third fairest to TCP flows under high initial slow-start threshold. Under low initial slow-start threshold, FAST-AT proved more unfair to TCP flows because of its quick increase in transmission rate upon entering congestion avoidance. Injecting more FAST-AT packets into the network induced more losses in TCP flows, which could not recover very quickly.

We found Scalable TCP, BIC and FAST to be most unfair to TCP flows. Established Scalable and BIC flows (large files) tended to maintain higher transmission rates after losses, while competing TCP flows cut transmission rates in half. By maintaining higher transmission rates and, thus, more buffer space, Scalable and BIC flows induced more losses in TCP flows. FAST could recover more quickly from losses than TCP flows and so FAST flows could occupy more buffers and induce more losses in TCP flows. In addition, like FAST-AT, FAST exhibited unfairness under low initial slow-start threshold because of its quick increase in transmission rate upon entering congestion avoidance.

HSTCP appeared moderately fair to TCP flows, especially under conditions of lower congestion and under a low initial slow-start threshold. HSTCP showed TCP unfairness, similar to Scalable TCP, under conditions of heavy congestion.

We believe that Scalable TCP, BIC and HSTCP could also be unfair to competing flows that are newly arriving. Given that some large flows operating under Scalable, BIC and HSTCP have established relatively high transmission rates and associated large buffer state and that newly arriving flows induce losses, the established flows will not reduce transmission rate very much and will maintain large buffer state. The newly arriving flows will be forced into congestion avoidance on the loss. Further, Scalable and HSTCP do not increase transmission rate very fast early in a flow's life; thus, the newly arriving flows will face difficulty increasing transmission rate.

10.1.1.4 Utility Bounds. We showed that alternate congestion-control protocols could provide increased utlity (goodput) for users; however, we also found that this increased utility would be maximized only under specific, bounded circumstances. First, the *rwnd* must not be constraining flow transmission rate. Second, a flow must be using a relatively low initial slow-start threshold. Third, a flow must be transmitting a large file. Fourth, a flow's packets must be transiting a relatively uncongested path (i.e., experiencing only sporadic losses from congestion or corruption) or else users must be willing to accept

marked unfairness (e.g., as seen with Scalable TCP) in trade for increased goodput. These bounds arise from some simple factors.

If a flow is restrained by receipt of a relatively small *rwnd*, then the ability of alternate congestion-control regimes to increase to a high *cwnd* cannot be used to transmit faster on a flow. Assuming *rwnd* does not constrain flow goodput, flows can increase goodput in concert with *cwnd* by using slow-start to discover the maximum transmission rate. Given a high initial slow-start threshold, then all flows can discover the maximum *cwnd* with the same quickness. In this case, TCP flows would reach maximum *cwnd* on average with the same pace as flows running alternate algorithms. Only when the initial slow-start threshold is low, forcing entry into congestion avoidance, could flows using alternate algorithms reach maximum *cwnd* more quickly than TCP. If flows are transferring large files, then the ability to reach maximum transmission rate quickly provides a substantial goodput advantage: the larger the file, the greater the advantage. Under small files the transmission could complete under initial slow-start and, thus, the advantage inherent in congestion-avoidance increase procedures for the alternate algorithms would not be realized. When flows transit heavily congested paths in the network, then most of the alternate congestion-control algorithms adopt TCP congestion-avoidance procedures, which negate any goodput advantage over TCP flows. Though FAST and FAST-AT do not adopt TCP congestion-avoidance procedures, we found that heavy congestion can cause oscillation in the transmission rate, which leads to higher loss rates, more retransmissions and lower goodput.

We are unable to determine how likely a particular flow is to operate under the bounded circumstances required for alternate congestion-control algorithms to provide improved goodput over TCP. Certainly it would be possible to engineer a network, or segments of a network, to provide specific users with high utility from alternate congestion-control algorithms. On the other hand, we suspect a rather low probability for such circumstances to arise generally in a network. Thus, we conclude that alternate congestion-control algorithms can provide improved user goodput; however, most users seem unlikely to benefit very often.

10.1.1.5 Safety. Given that on occasion some users could benefit from the increased goodputs available from alternate congestion-control algorithms, we need to consider whether widespread deployment of such algorithms could induce undesirable macroscopic characteristics into the network. In other words, are there significant costs that might offset the modest benefits associated with deploying alternate congestion-control algorithms? We can answer this question only in part because we simulated networks that used either a single congestion-control regime or a single alternate congestion-control algorithm mixed with TCP congestion-control procedures. There could be additional cautionary findings that arise from a heterogeneous mixture of alternate congestion-control algorithms. We postpone such findings to future work.

In our experiments, we simulated a wide range of conditions and we considered numerous scenarios comparing network behavior under specific alternate congestion-control algorithms, sometimes mixed with TCP procedures. For most algorithms under most conditions, we found little significant change in macroscopic network characteristics. One exception relates to FAST and FAST-AT. In spatiotemporal realms with high congestion, where there were insufficient buffers to support the flows transiting

specific routers, FAST and FAST-AT entered an oscillatory behavior where the flow *cwnd* increased and decreased rapidly with large amplitude. Under these conditions, the network showed increased loss and retransmission rates, a higher number of flows pending in the connecting state and a lower number of flows completed over time. Thus, FAST and FAST-AT should be deployed on a wide scale only with great care. There appears to be some possibility that FAST could cause significant degradation in network performance in selected areas and for selected users. We recommend the need for additional study of FAST and FAST-AT prior to widespread deployment and use on the Internet.

10.1.2 Characteristics of Individual Congestion-Control Algorithms

Below, we provide a brief summary of the characteristics found from our experiments for each alternate congestion-control algorithm. We discuss the algorithms in alphabetical order.

10.1.2.1 BIC. Clearly, among the seven algorithms we studied, BIC is the most complex to code and implement, requiring a potentially substantial amount of processing to adjust the *cwnd*. BIC uses standard TCP congestion-avoidance procedures when *cwnd* is below a low-window threshold (14 packets, here). Under congestion with losses spaced sufficiently in time, BIC reduces *cwnd* less quickly than standard TCP; thus, BIC can achieve higher goodputs under sporadic losses by maintaining a high transmission rate and associated buffer state. This can be somewhat unfair to newly arriving flows. On the other hand, when congestion becomes severe, with losses spaced closely in time, BIC reduces *cwnd* much more quickly than TCP. Under such circumstances, BIC can take substantial time (average 71.3 s in our experiments) to recover maximum goodput after congestion eases. When considering the rate of increase in transmission speed under low congestion after reaching initial slow-start threshold, BIC averaged about 18.8 s to reach maximum transfer speed on long-lived flows. This rate of increase ranked fifth (of six) overall, and was competitive with HTCP, Scalable TCP and HSTCP.

10.1.2.2 CTCP. The algorithm for CTCP requires periodic processing to adjust an auxiliary delay window (*dwnd*), which increases the processing cost beyond that found in standard TCP congestion control. Under congestion, CTCP reduces transmission rate by one-half and then recovers relatively quickly. The advantage of CTCP recovery procedures appears most obvious after a period of severe congestion on a path. Under easing congestion, *dwnd* can increase quite quickly. Since CTCP augments the *cwnd* with the *dwnd*, transmission rate can also increase quickly – returning to maximum rate in an average 2.9 s in our experiments. In fact, in some situations, the rate of increase in *dwnd* appears unbounded. CTCP implementations should probably require a bound on maximum *dwnd*. Under periods of heavier congestion, increase in *dwnd* is constrained. In addition, the CTCP algorithm appears quite fair to competing CTCP flows as well as TCP flows. CTCP had the highest default low-window threshold (41 packets, here) among the algorithms we studied. Further, CTCP averaged about 7.9 s to reach maximum transfer speed on long-lived flows under low congestion and low initial slow-start threshold. This rate of increase ranked second overall behind only FAST and FAST-AT, which tied for first.

10.1.2.3 FAST. The algorithm for FAST requires periodic processing to adjust the target *cwnd*. While each adjustment demands little computation, the default periodicity (20 ms, here) can require multiple adjustments within a single round-trip. FAST does not have a low-window threshold; thus, after initial slow-start, FAST flows never use standard TCP congestion-avoidance procedures. Under congestion, FAST reduces transmission rate by one-half and then recovers very quickly. The advantage of FAST recovery speed appears under both sporadic losses and when congestion eases following a period of severe congestion on a path. Under easing congestion, FAST recovered maximum transmission rate in an average of 6.6 s in our experiments. On the other hand, for flows transiting congested areas, with insufficient buffer space for all flows, FAST exhibits oscillatory behavior that increases losses and, thus, retransmissions, which reduces user goodput. Under severe conditions, FAST causes an increase in flows pending in the connecting state because SYN packets are loss with increased probability. In addition, FAST can significantly reduce the number of flows completed over time in a network. Among the algorithms we studied, FAST achieves maximum available transmission rate in the shortest time (3.7 s average) on long-lived flows under low congestion and low initial slow-start threshold. The ability of FAST to accelerate transmission rate led to superior goodputs (under low congestion and low initial slow-start threshold) for file sizes larger than Web objects, and the advantage of FAST increased with file size. The ability of FAST to quickly attain high transmission rates for large files tended to induce losses in competing flows. Since TCP flows could not recover quickly, FAST flows could attain much higher goodputs than competing TCP flows.

10.1.2.4 FAST-AT. The FAST-AT algorithm augments FAST with periodic procedures to monitor throughput and tune the α parameter used when adjusting the target *cwnd*. Without α tuning, FAST sets the α parameter to a fixed value. FAST-AT monitors throughput every round-trip time and tunes the α parameter periodically (every 200 s, here). As throughput improves past specified thresholds α is increased and as throughput declines past specified thresholds α is decreased. FAST-AT exhibits many of the same positive and negative properties as FAST. The main difference was that, under severe and sustained congestion, FAST-AT reduced the α parameter from a default setting of 200 to as low as 8. In such, circumstances FAST-AT recovers much more slowly than FAST. When throughput begins increasing, FAST-AT adjusts the α parameter only every 200 s and must make two adjustments (8 to 20 followed by 20 to 200) before reaching the maximum recovery rate. In our experiments, when recovering from sustained periods of heavy congestion, FAST-AT took longer (26 s average) to reach maximum transmission rate than all alternate algorithms except BIC. On the other hand, by recovering transmission rate more slowly under heavy congestion, FAST-AT proved more TCP friendly than FAST. This occurred because under such circumstances FAST-AT did not induce as many losses in competing TCP flows.

10.1.2.5 HSTCP. The HSTCP algorithm is a relatively straightforward; updating the *cwnd* no more frequently than standard TCP. The HSTCP *cwnd* updates involve somewhat costly logarithmic and exponentiation operations. HSTCP uses standard TCP congestion-avoidance procedures when the *cwnd* is below a low-window threshold (31

packets, here). HSTCP reduces *cwnd* less on a loss than standard TCP and provides more than linear increase in *cwnd* during congestion avoidance. Under both sporadic and heavy congestion, HSTCP retains a higher transmission rate (and associated buffers) than TCP. By maintaining more buffered packets, HSTCP can induce losses in competing flows. In such situations, newly arriving HSTCP flows can have difficulty increasing transmission rate, especially on paths with longer propagation delays. In addition, losses induced on competing TCP flows hurt goodput for TCP users because TCP recovers only linearly. When recovering from periods of sustained heavy congestion, HSTCP performed third best (10 s average) in our experiments; however, the short recovery time can be attributed mainly to the fact that, in comparable situations, HSTCP flows did not reduce transmission rate as much as most other congestion-control algorithms. Under low congestion and low initial slow-start threshold, HSTCP achieved maximum transmission rate more slowly (22.4 s average) than all other alternate congestion-control algorithms we studied.

10.1.2.6 HTCP. The HTCP algorithm requires a periodic (250 ms, here) process to monitor flow throughput. HTCP uses standard TCP congestion-avoidance procedures for a specified period (1 s, here) after a packet loss. Under congestion, HTCP behaves like standard TCP congestion avoidance. The heavier the congestion, the more time HTCP spends using TCP procedures. When recovering from periods of sustained heavy congestion, HTCP performed fourth best (10 s average) in our experiments. Under sporadic losses, HTCP can spend too much time using TCP's linear increase. In our experiments, this trait led HTCP to provide lower goodput than other alternate congestion-control algorithms on large files. On the other hand, by adopting TCP congestion-avoidance procedures following packet loss, HTCP is quite TCP friendly. Under low congestion and low initial slow-start threshold, HTCP achieved maximum transmission rate somewhat slowly (16.6 s average), comparable to BIC, HSTCP and Scalable TCP, but significantly slower than CTCP, FAST and FAST-AT.

10.1.2.7 Scalable TCP. The Scalable TCP algorithm is a small modification of standard TCP congestion-avoidance. Scalable TCP increases *cwnd* by a constant on each acknowledgment and decreases *cwnd* by 12.5% on each loss. In addition, Scalable adopts standard TCP congestion-avoidance procedures when *cwnd* is below a low-window threshold (16 packets, here). Under congestion, established Scalable TCP flows do not reduce transmission rate very quickly. By maintaining more buffered packets, Scalable can induce losses in competing flows. In such situations, newly arriving Scalable flows can have difficulty increasing transmission rate, especially on paths with longer propagation delays. In addition, losses induced on competing TCP flows hurt goodput for TCP users because TCP recovers only linearly. When recovering from periods of sustained heavy congestion, Scalable performed fifth best (22.5 s average) in our experiments; however, the recovery time can be attributed mainly to the fact that, in comparable situations, Scalable flows did not reduce transmission rate as much as most other congestion-control algorithms. Under low congestion and low initial slow-start threshold, Scalable TCP achieved maximum transmission rate somewhat slowly (17.8 s average). In fact, Scalable increased transmission rate very slowly for the first few

seconds of long-lived file transfers, which means that Scalable provides a steep increase in transmission rate only for large files.

10.1.3 Recommendations

Under some circumstances, users can benefit from adopting alternate congestion-control algorithms to transfer files on the Internet. For that reason, it makes sense to deploy such algorithms into computers attached to the Internet. Of course, the probability appears quite low that a specific user will see benefits on any particular file transfer. Among the alternate congestion-control algorithms we studied, CTCP appears to provide the best balance of properties. Under low congestion, CTCP can increase transfer rate relatively quickly when operating in the congestion-avoidance phase. Further, CTCP reduces transmission rate relatively quickly in the face of sustained congestion and recovers to the maximum transmission rate quite quickly when congestion eases. CTCP appears relatively friendly to flows using standard TCP congestion-control procedures. CTCP, along with most of the other alternate congestion-control algorithms we studied, is unlikely to induce large shifts in the macroscopic behavior of the Internet. FAST and FAST-AT have some appealing properties, especially with respect to achieving maximum transmission rate quickly on high-bandwidth, long-delay paths and recovering quickly from sporadic losses. Unfortunately, when transiting highly congested paths with insufficient buffers to support the flow volume, FAST and FAST-AT can enter an oscillatory regime that could significantly increase loss and retransmission rates. Flows transiting affected areas would take longer to connect and complete and would receive lower goodputs.

10.1.4 Future Work

We studied seven proposed replacement congestion-control mechanisms for the Internet. Despite the comprehensive nature of our study, more work remains to be done in at least four directions. First, we limited our study to a bounded set of alternate congestion-control algorithms for which we could find empirical data against which to validate our simulations. Researchers have proposed many congestion-control algorithms that were not included in our study; thus, one direction for future work is to consider the behavior of additional algorithms. Of particular interest is CUBIC, which has replaced BIC as the congestion-control algorithm enabled by default in Linux.

Second, we have not considered scenarios where multiple alternate congestion-control algorithms are mixed together in the same network. Increasing the heterogeneity of algorithms might reveal additional insights about the advantages and disadvantage of the various algorithms, as well as uncover undesirable macroscopic behaviors resulting from such mixtures. Where undesirable behaviors do not appear, then such a study would increase confidence in the safety of deploying alternate congestion-control regimes. Of course, conducting such a study would likely require substantial increase in demand for computation resources in order to simulate enough network evolution to accumulate sufficient samples to reveal statistically significant behavioral patterns.

Third, we have not validated our findings against live, controlled experiments configured in GENI or a similar test bed environment. Conducting such a validation would substantially increase confidence in the findings of our study. We intend to undertake such a validation as soon as we can gain access to sufficient resources to

support our experiments. In the meantime, we also plan to consider how we might attempt to validate our findings using test environments of smaller scale. One way to approach this may be to make predictions about behaviors we should see replicated even at smaller scale than the network sizes and speeds we simulated.

Fourth, our study revealed various strengths and weaknesses in the congestion-control algorithms we investigated. Future researchers could exploit our findings to propose algorithm improvements that compensate for identified weaknesses, while retaining strengths. Further, our general findings may also help other researchers to improve future designs for additional congestion-control algorithms.

10.2 Conclusions about Methods

SEE ON-LINE STUDY