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

The simulation and modeling studies reported here also enabled us to evaluate each of the modeling and analysis methods we used. Below, we first discuss the use of discrete-event simulation as a technique to model systems at large scale. Subsequently, we evaluate the specific methods we applied to solve each of the five hard problems that we identified in Sec. 2.4. We than provide some overall recommendations for those seeking to model and analyze large distributed systems. We close with suggestions for future work.

10.2.1 Discrete-Event Simulation

Recall that our discrete-event simulation model, MesoNetHS, was constructed because an existing cellular-automaton model proved unable to scale to simulate networks of the size and speed required for this study. The cellular-automaton model simply demanded far too many computation resources. What about discrete-event simulation? We demonstrated that MesoNetHS could feasibly simulate a network operating for one hour at contemporary router speeds while transporting hundreds of thousands of simultaneous flows with a mix of 100 Mbps and 1 Gbps sources and receivers sending flows with sizes ranging from tens of kilobytes to gigabytes. Of course, running such a simulation for an average parameter combination required about 17 ½ days of processing time. In the best case, such a scenario required just over 8 days and in the worst case just over 30 days. The speed of individual processors seems unlikely to improve much in the future. Instead, computer systems will be outfitted with an increasing number of processors that can be used in parallel. Increasing parallelization is a nice match for orthogonal fractional factorial experiment designs (see Sec. 10.2.2.3 below); however, each individual experiment run must still be completed within a time budget. We conclude that discrete-event simulation is unlikely to support network simulations much beyond the scale we used in our study. Even if one is willing to wait 60 or 90 days for a single simulation run to complete, the odds seem low that the underlying hardware, operating system, simulation environment and model could run so long without incurring some sort of failure. Researchers are investigating parallel simulation as a means to increase the scale of runs that can be executed; however, temporal relationships among elements in network simulations will probably restrict the degree of speedup that can be achieved. We conclude that increasing the scale of a simulated network will likely require a different paradigm, such as fluid-flow or hybrid simulations. We discuss such models further in Appendices A and B.

10.2.2 Scale-Reduction Techniques

We adopted five specific techniques to reduce the scale of parameter and response spaces in our experiments. Below, we evaluate each technique in turn.

10.2.2.1 Model Restriction and Parameter Clustering. Restricting model parameters to those germane to this specific study led to substantial reduction in intellectual effort associated with identifying and assigning values to both fixed and variable parameters. Further, reduction in the parameter space lowered the overall computational demand associated with individual experiment runs and with experiment campaigns. Clustering individual parameters into groups, each representing a key factor driving a simulated network, further reduced the intellectual effort needed to parameterize experiments and also to analyze responses to assess the influence of particular input factors. Of course, the reductions associated with restricting and clustering input parameters were insufficient alone to achieve computational tractability for the experiments in this study. Other reductions were required (see Sec. 10.2.2.2 and 10.2.2.3 below). Further, significant domain expertise was needed to identify reasonable parameter restrictions and groupings. This may impede studies where such expertise is unavailable. In some cases, substantial simulation executions and data analyses were required to identify parameter reduction decisions that were inappropriate. For example, we required about one week of simulation to identify a gradual trend toward an increasing number of active flows. Discovering this trend led us to introduce connection-establishment procedures into the simulation; illuminating the important role that such procedures play in controlling network-wide congestion.

10.2.2.2 Two-Level Experiment Designs. The largest reduction in computation demand for simulations in this study arose from the simple act of limiting parameter settings to only two levels. Of course, taking this step incurred several drawbacks. First, the experiment designer must select specific values for each level. This requires significant domain expertise. Second, the results obtained for each experiment are robust only over the range defined by the selected level settings. Third, drawing conclusions from a two-level experiment design entails an assumption that a model behaves linearly within the range defined by the selected settings. To mitigate these restrictions, the study adopted several experiments and varied level settings between experiments. While some may cringe at limiting parameter settings to two levels, we demonstrated in this study that significant insight can be gained even under such a severe restriction.

10.2.2.3 Orthogonal Fractional Factorial Experiment (OFF) Designs. OFF experiment designs enabled us to further reduce computation demand in cases where simulating all combinations of parameter settings proved too costly, even after limiting parameters to only two levels. In general, OFF designs allow an experiment designer to simultaneously vary parameter combinations in a balanced and orthogonal fashion to provide the maximum amount of information given a limit on the affordable number of experiment runs. Since each selected combination of parameters represents an independent simulation run, OFF experiment designs create an embarrassingly parallel suite of simulations that can be spread across all available processors. Recall, however, that each individual simulation run must still be computationally feasible (as discussed in Sec.

10.2.2.1 above). Another advantage to two-level OFF designs arises from an effective match with a ten-step graphical analysis technique developed at NIST (see Sec. 10.2.4.1 below). Pairing two-level OFF designs with the NIST analysis technique reveals substantial information about system behavior – within the restrictions of two-level designs (as discussed above in Sec. 10.2.2.2). Of course, OFF designs further reduce the potential parameter combinations examined in a particular study. In general, no study can cover all potential parameter combinations. The most typical approach adopted by network researchers entails fixing all parameter settings except one, which is varied across a range of levels. The results of this one-factor-at-a-time approach can produce nice x-y plots; however, the resulting conclusions are valid only under a specific combination of fixed parameters. OFF designs provide a principled technique to vary multiple parameter settings simultaneously, which yields more information about overall behavior of a system. Further, OFF designs can identify model errors more readily than one-factor-at-a-time experiments because OFF designs probe a model under a larger variety of conditions.

10.2.2.4 Correlation Analysis. Correlation analysis proved an effective technique to reduce the response space that we needed to examine. In one of our sensitivity analyses (see Sec. 4), for example, we showed how 22 potential responses could be reduced to only seven. Assuming availability of a domain expert, correlation results may also aid in model validation. For example, a domain expert should be able to verify whether or not the correlations make sense. A domain expert should also be able to attribute surprising correlations to modeling error or to new insights. We found that a given set of correlation results apply only to the specific range of parameter combinations used to generate the related responses. For example, the correlations identified in Appendix C differ in some ways from the correlations identified in Sec. 4. Examination by a domain expert revealed that both sets of correlations are valid; differences arose from variations in the range of parameters simulated. We conclude that correlation analysis should be applied separately to each suite of experiments where level settings differ.

10.2.2.5 Principal Components Analysis (PCA). We used PCA to complement correlation analysis[3]. PCA aims to identify orthogonal variations, so-called principal components, in response data and to assign weights to indicate the degree to which responses influence each identified principal component. For the models simulated in this study, we found that most variation in response data could be accounted for by the first three or four principal components in a given analysis. This implies, for example, that we might be able to analyze three or four responses instead of the 22 used in our sensitivity analysis. Further, a domain expert could compare the findings from a PCA against the findings from a correlation analysis to determine if the two analyses were consistent. This consistency check helps to further validate a model. On the other hand, working with PCA results can be somewhat difficult for a few reasons. First, principal components are abstract linear weighted combinations of responses; thus, there is no specific domain interpretation behind a given component. An analyst or expert must invest considerable effort to develop a domain interpretation of even the top two or three principal

[3] We also used PCA to identify sources of variation in data related to several experiments throughout our study. We evaluate PCA in these applications below in Sec. 10.2.4.3.

components. In some cases (e.g., Sec. 4), a clear and reasonable interpretation can be achieved. In other cases (e.g., Appendix C), interpretation becomes more difficult. Second, principal components can take on both positive and negative values, which present an analyst with difficulty assigning meaning. In fact, conducting a main-effects analysis of principal components required us to refer to main-effects analyses of responses in order to develop an interpretation. Third, PCA sometimes identified components that proved coarser than similar response groupings developed with correlation analysis. When aiming to reduce the response state space, we conclude that PCA provides a reasonable complement to correlation analysis; however, domain experts will often find correlation analysis more readily comprehensible than PCA.

10.2.3 Model-Validation Techniques

As we discussed in Sec. 2, network researchers typically do not know whether their models are valid. For this reason, we took two steps to increase confidence in the validity of MesoNetHS. We evaluate each step in turn.

10.2.3.1 Sensitivity Analysis. Sensitivity analysis provided a tractable means to investigate the response of MesoNetHS to various changes in model input parameters. Using sensitivity analysis we were able to find and fix errors in early model formulations and, ultimately, to develop confidence that the model was fit for its intended role in our study. Of course, to conduct such analysis we combined many of the individual methods evaluated here, including two-level OFF experiment design, correlation and principal-components analyses and ten-step graphical analysis. For this reason, our approach to sensitivity analysis inherited the strengths and weaknesses associated with the individual methods. In particular, the two-level design limited the range of our conclusions about model validity. To mitigate that, we conducted a supplementary sensitivity analysis (see Appendix C) that adopted different level settings for each input factor evaluated. We also used preliminary sensitivity analyses to identify factors that did not much influence model responses. We could then reduce the search space in subsequent analyses. We conclude that rigorous sensitivity analysis is feasible when using a reduced-scale model and two-level OFF design, combined with judicious choice of factors to vary. Results from sensitivity analyses guided us in designing specific experiments associated with our study. We recommend that campaigns of simulation (or analysis) experiments use only models that have been examined for sensitivity to changes in input parameters.

10.2.3.2 Key Empirical Comparisons. To increase confidence that we had correctly modeled specific congestion-control algorithms we relied on key comparisons between simulation results and empirical results. Such comparisons were facilitated by the existence of published empirical results measured under controlled circumstances in a small ("dumbbell") topology. We were easily able to model the small topology in MesoNetHS and to simulate the same scenarios and parameter settings used in the empirical studies. Comparing our simulation results with empirical results enabled us to identify errors in modeling several congestion-control algorithms. We were also able to correct our models and ensure that we obtained results consistent with empirical results. In this way, we gained confidence in our models of the various congestion-control algorithms prior to increasing the scale of topology we simulated. As an added benefit,

the empirical study identified default parameter settings adopted by congestion-control algorithms. We were able to adopt those settings for our large simulations. For a given study, empirical results may be unavailable, either because the problem under study is not yet implemented or because no one has published empirical results. Where feasible, we recommend that a small experimental configuration be used to generate empirical measurements in order to ground a mathematical model in reality. Preferably, the empirical measurements should be made from implementations developed independently from the models. The empirical measurements should capture key aspects of system behavior on a small scale. When empirical measurements cannot be made available, important aspects of a model may go unconfirmed. From our experiences, the resulting model can contain significant errors that lead to invalid behaviors. We recommend that significant studies endeavor to compare key model aspects with empirical measurements taken at small scale. Any reasonable expense required to obtain empirical measurements will be repaid by enhancing confidence in models used to study large-scale systems.

10.2.4 Data-Analysis Methods

Data analysis comprises the third axis of our method; modeling and experiment design comprise the other two axes. We applied and explored four data-analysis methods during our study. We evaluate each method below.

10.2.4.1 Ten-Step Graphical Analysis. In support of exploratory analysis, NIST designed a ten-step graphical method (explained in Sec. 4.1.3), where each step generates an individual plot designed to answer one specific question regarding a data set. We employed this ten-step method as a main part of our sensitivity analysis; we applied all ten steps to each response. The analysis method is designed to match well with a two-level OFF experiment design. Clearly, for our application, the main-effects plot (step 3) proved most insightful – revealing changes in system responses resulting from changes in input factors. The interaction effects matrix (step 4) also helped to identify that MesoNetHS was driven primarily by main effects, rather than by two-term interactions. Other plots proved useful for specific, limited purposes. For example, the ordered data plot identified specific combinations of factor settings that produced significant effects on the responses. The scatter plot summarized how the distribution in responses changed with respect to changes in input factors. Several other plots provided redundant information, which served to confirm related results or to identify analysis errors. For example, the Youden plot identified the most significant factors driving particular responses, which could also be ascertained from main effects plots, as well as a number of other plots. The |Effects| plot and the cumulative residual standard deviation plot helped to visualize whether a linear model could approximate a system's response to input factors. A derived contour plot suggested how specific changes in the two main factors influencing a response might drive the response in particular directions. For our purposes, the box plot did not provide significant new information. Overall, the ten-step graphical analysis proved quite useful in analyzing a model's sensitivity to changes in input parameter settings. We applied all ten steps to our initial sensitivity analysis. Subsequently, we used only main effects plots and interaction effects matrices, which provided the most important information for our supplementary sensitivity analysis. We recommend applying all ten steps of the graphical analysis during early stages of model

development and investigation. The various plots reveal a range of confirming and complementary information that could prove quite insightful. In later stages of analysis, we recommend limiting selected plots to only those necessary to address specific questions of interest.

10.2.4.2 Cluster Analysis. We employed cluster analyses to reveal overall patterns of similarities and differences in responses. We sought patterns in behavior among selected congestion-control algorithms under conditions, composed of combinations of input parameters. Because parameter combinations varied greatly, we clustered algorithms only with respect to individual conditions. To identify clustering patterns, we needed to characterize differences among conditions. Such characterization required external analyses. Given dendrograms for each condition, along with characterizations of each condition, we were able to identify patterns where selected algorithms clustered together. These clustering patterns provided general relationships among algorithms and congestion. On the other hand, the patterns did not identify specific relationships among responses that led to the patterns; for this we required more detailed analyses (see 10.2.4.4 below). Cluster analysis appears well suited to identify general patterns where such patterns exist. Further, using cluster analysis, we were able to identify close correspondence in the behaviors of two congestion-control algorithms, which enabled us to make informed decisions about more detailed analyses. On the whole, cluster analysis can provide a concise overview of patterns in data sets; however, one should not expect cluster analysis alone to provide insight about causality. At best, we found that cluster analysis can help to identify aspects of the data that should be given closer scrutiny.

10.2.4.3 Custom Multidimensional Visualizations. Using Dataplot, a statistical scripting language and supporting run-time environment developed at NIST, we could construct custom visualizations designed to explore specific relationships among multidimensional data sets. We developed several custom, multidimensional, visualizations for our study. The resultant representations provided substantial insight into overall system behavior. In fact, custom visualizations provided launching points for many causality analyses (see Sec. 10.2.5). Custom visualization can provide a domain expert with concise and precise information regarding the questions under study. Further, detailed custom visualizations can be subjected to custom summarizations that identify key patterns in experiment data. On the other hand, successful custom visualizations entail collaboration among a statistical visualization expert and a domain expert, who must iterate over the design and construction of each visualization until a useful result emerges. We found, however, that a few (four or five) well-crafted multidimensional visualizations could be reused to analyze data from most experiments in our study. We recommend custom multidimensional visualizations as a key tool for analysis of data sets for complex systems. Of course, we were fortunate that one study participant was expert in the design and programming of statistical visualizations. Custom visualizations would be difficult to create and apply without access to the necessary expertise.

10.2.4.4 Exploratory Multidimensional Interactive Visualization. Early in our study, we collaborated with visualization experts, who constructed DiVisa, a general purpose system for interactive exploration of multidimensional data. DiVisa enables an analyst to

view multiple, related, data simultaneously, while assigning custom visual attributes to represent various dimensions in the data. For example, visual attributes may include color, size and shape. Altogether, DiVisa allowed an analyst to assign up to eight different attributes to data. In using DiVisa, an analyst needs to remember how attributes are assigned. The resulting visualizations proved quite abstract and difficult to interpret. Late in development, and at the request of a domain expert, DiVisa was extended to support display of topological information associated with a given simulation. Since a topology is quite natural for a networking expert, DiVisa acquired increased utility for our study. In fact, given voluminous spatiotemporal information, such as queue sizes evolving over time in every router in a network topology, DiVisa could replay the dynamic behavior of a MesoNetHS simulation, which enabled us to detect unwanted behaviors in various simulations and to adjust model parameter settings as necessary to achieve desired effects. Unlike custom multidimensional visualizations, interactive visualization systems require a domain expert to explore system data and to develop revealing representations by assigning attributes to data dimensions. We found this quite difficult to do. Perhaps our experience would have been different had we collaborated with an expert in interactive multidimensional visualization. We do not recommend using abstract interactive systems for visualizing multidimensional data unless the resulting displays can be readily related to concepts comprehensible to a domain expert.

10.2.5 Causality-Analysis Methods

We chose data-analysis methods mainly based on an ability to reveal overall patterns in behavioral data derived from models of large systems. Once significant patterns were revealed, we typically needed to delve into more depth in order to determine root causes for the patterns. In this study, we adopted four main techniques

10.2.5.1 Principal Components Analysis (PCA). We were sometimes able to apply PCA to identify causality underlying variations in response data. For example, when searching for causes in goodput variation among many flow groups under generally low congestion, PCA was able to identify the main drivers as network speed, propagation delay and file size. PCA could also discern cases where congestion-control algorithms did contribute to variation in goodput. In general, after using PCA to find the two main principal components, an analyst creates a scatter plot of component one vs. component two, where each point represents a particular parameter combination. Visual inspection may then be used to discriminate clusters, or groupings, of points. Using supplementary analyses, an analyst can characterize common factor settings among points in each grouping. Using this technique, factors underlying variation in the data can become quite explicit. Further, this level of analysis requires little domain expertise and may be accomplished based solely on the factors and settings in a two-level OFF experiment design.

10.2.5.2 Detailed Measurements. Causality exploration sometimes requires detailed spatiotemporal data related to a specific question under investigation. At other times, an analyst may need to peruse spatiotemporal data to determine if a system is behaving as expected. We chose to collect detailed spatiotemporal data as an integral part of our simulation model, MesoNetHS. In fact, for pattern analysis we generated summary data from the detailed measurements. We found several advantages to this approach. First, we

could use the spatiotemporal behavior of our model to determine what data to summarize in order to avoid transient startup behavior. Second, we could subject our model to exploratory analyses (see Sec. 10.2.4.4 and Sec. 10.2.5.4). Third, should patterns from data analysis indicate need to further investigate detailed behavior, the data would already exist.[4] Fourth, should other researchers wish to investigate particular questions not addressed in this study, the data would be available for later use. Some drawbacks also arise from collecting such detailed data. First, collecting extensive spatiotemporal data can require substantial memory within a running simulation. We mitigated this by permitting the simulation to periodically dump the measurement buffer to disk and then reuse the space for additional measurements. Of course, this increases the failure surface of the simulation. In practice, we found that making incremental measurements worked effectively, even when writing results to a file server located on a network.[5] Second, collecting extensive spatiotemporal data requires availability of sufficient disk storage. The experiments in this study required approximately 250 GBytes of storage; so this was easily accommodated. For studies investigating behavior in large distributed systems, we recommend collecting detailed spatiotemporal data regarding every conceivable aspect that might be of interest. Summary data can be created from the details, as required for analysis.

10.2.5.3 Scientific Method. Given a pattern of interest revealed by data analyses, we used the scientific method to search for causality. In general, we would posit a hypothesis to explain an observed pattern. We would then suggest detailed behavioral data that should exist to confirm our hypothesis. We would investigate the detailed data and either confirm or refute the hypothesis. If the hypothesis were refuted, then we would develop an alternate hypothesis and repeat. Eventually, we would construct a confirmed hypothesis for a given pattern and that would establish a causal link. This approach often proved time consuming, especially when a domain expert had insufficient insight to formulate a good, initial hypothesis. On the other hand, the rigorous nature of the entire modeling and analysis method we used built increasing insight into system behavior as the study progressed. For this reason, it became easier over time to generate good hypotheses. We were able to establish causality for every pattern of interest to us. Of course, our data is available for use by other researchers who might reach different conclusions than we have about particular causal links. While we would prefer to suggest

[4] In practice, we made initial guesses about the detailed data we needed to collect. During our study, specific issues revealed the need to collect additional details, such as the temporal posture of the network with respect to the state (e.g., idle, connecting, active) and phase (e.g., initial slow-start, normal congestion avoidance, alternate congestion avoidance) of all flows. So, while one can arrange to collect substantial detailed data during model construction, the need might arise to add additional measurement data during a particular study.

[5] We did find it necessary to modify the simulation to detect network outages that prevented writing measurements and then to detect resumption of a network path so that the measurements could be written. During times of prolonged network outage the simulation halts while waiting to purge the measurement buffer. In some instances, when the file server crashed, the simulation could not write measurement results because the file handle was stale. Failure to recover from a stale file handle required a simulation to be restarted. Such instances were relatively rare. We were unable to use the SLX checkpoint and restart functions because they assume that a simulation would be reloaded into the same memory addresses. We could not guarantee that this would be the case.

a more direct process to establish causality, we had little recourse but to adopt the scientific method in cases where PCA could not provide sufficient insights.

10.2.5.4 Exploratory Analysis. While the scientific method provides an orderly approach to establish causality, we also sometimes adopted a more exploratory approach. In general, we would select some related aspects of system behavior and then analyze or interact with time varying data to discover trends. We used this technique, for example, to characterize the temporal evolution of flow states and phases arising from various levels of congestion. We also sometimes used exploratory analysis to develop hypotheses about causes underlying patterns arising from analysis of summary data. For example, we used exploration of temporal variations in congestion-window size on specific flows to create the hypothesis that frequent, high amplitude oscillations in the congestion-window were responsible for increased loss rates exhibited by FAST and FAST-AT under high spatiotemporal congestion. Of course, to engage in exploratory analysis, one needs to have sufficient data collected under various well-understood conditions.

10.2.6 Experiment-Selection Methods

Despite the powerful modeling and analysis techniques available to study behavior in large systems, significant spatiotemporal patterns can be missed completely if the selected experiments do not create the necessary conditions. While domain expertise plays a crucial roll in experiment selection, we also adopted two other techniques that helped ensure good coverage of key behaviors.

10.2.6.1 Rank Analysis. We exploited the sensitivity analysis of MesoNetHS to identify input factors exerting the largest influence on model response. This enabled us to concentrate our experiment campaign initially on varying those factors most likely to drive model behavior. We were able to keep other factors fixed. This shows how findings from sensitivity analysis can help to craft effective experiment designs.

10.2.6.2 Domain Expertise. In designing our initial experiment (described in Sec. 6), we relied mainly on domain expertise. A domain expert conceived a temporal scenario that started with typical Web browsing traffic, added heavy congestion in a spatiotemporally localized form and then removed the heavy congestion to allow for resumption of normal Web browsing. A domain expert also introduced three long-lived flows that could provide a detailed view of congestion-control behavior. This basic scenario proved well-suited to investigate many operational aspects of congestion-control algorithms. Insufficient domain expertise could create a significant impediment to designing insightful experiment scenarios.

10.2.6.3 Incremental Design. We used incremental design to help construct effective and efficient experiment campaigns. In incremental design, results of preceding experiments are used to select parameters and scenarios for subsequent experiments. For example, our first experiment showed that using a large initial slow-start threshold reduced differences among most congestion-control algorithms. The initial experiment also identified some distinctive behaviors arising from FAST. Given these factors, we were able to craft our second experiment (see Sec. 7) to examine any changes that resulted from using a low

initial slow-start threshold and from including FAST with α tuning. At the same time we were able to determine whether reducing the size and speed of a simulated network would reveal new information. We made these changes while retaining the fundamental scenario from the initial experiment. We used the findings from the second experiment to design a subsequent pair of experiments (discussed in Sec. 8) that examined the influence of initial slow-start threshold in an evolving network. At the same time, domain expertise injected much lower overall congestion and a richer variety of traffic sources into the experiments. Based on findings from these experiments, we decided to design an experiment (reported in Sec. 9) that focused on loss/recovery procedures within the congestion-control algorithms, while at the same time increasing the size and speed of the simulated network. We recommend interweaving domain expertise and incremental design to construct the most effective experiment campaign to fit within the allotted time.

10.2.7 Recommendations

Investigating the behavior of large distributed systems benefits from rigorous application of a coherent set of experiment design and analysis methods. We recommend that such investigations adopt two-level[6] OFF experiment designs, which can facilitate a wide range of analysis methods. We also recommend that system models (whether analytical or simulation) be subjected to sensitivity analyses in order to understand response to changing input parameters. We advocate the use of incremental design when constructing an experiment campaign. We suggest that discrete-event simulations can capture more detailed aspects of system behavior than would typically be feasible with more abstract models. We found that various scale-reduction methods can lead to tractable simulations for systems of significant size; however, there appears to be a limit. We recommend validating key model behaviors against empirical measurements, where feasible. We also identified several useful data-analysis methods that can reveal overall behavior in large systems. Some methods, such as cluster analysis, reveal the presence of behavioral patterns without providing much insight into underlying causes. Other methods, such as the NIST-developed ten-step graphical analysis, give better insights. We found that custom multidimensional visualizations can be quite informative; however, creating such visualizations requires significant iteration between a domain expert and an expert in statistical visualization. Causality analysis remains largely beyond the grasp of automated analysis methods. Investigating causality required iterative application of the scientific method: a domain expert developed a hypothesis regarding a macroscopic pattern of behavior and then used evidence from detailed spatiotemporal data to confirm or refute the hypothesis. For this reason, we recommend capturing data in as much spatiotemporal detail as a model will permit. Finally, we found that effective use of software for interactively exploring multidimensional data requires visualizations that relate to concrete concepts within the domain under investigation.

[6] Two-level designs allow a system to be investigated under a wide range of conditions with reasonable cost. Once overall behavior of a model is understood, later experiments can focus on fewer factors with more levels, as needed to investigate specific aspects of behavior. The quality of these more focused experiments will be improved when placed within the context of a well-understood model.

10.2.8 Future Work

We suggest three areas for future work on modeling and analysis methods for large distributed systems. First, we recommend investigating methods that enable abstract models to yield improved accuracy. For example, some researchers have developed a hybrid model combining continuous-time logic with discrete events to achieve efficient simulation of system behaviors (see Appendix B). Similarly, we are working to improve the accuracy of fluid-flow models of congestion-control algorithms (see Appendix A). Such hybrid or fluid-flow models could be augmented with features necessary to support the experiments adopted in the current study and then the experiments could be repeated. Perhaps one of these abstract models could reveal the same findings at reduced cost. Second, we suggest investigating approaches to automate design of custom visualizations for multidimensional data. Currently, successful application of custom visualization requires substantial collaboration between a domain expert and an expert in statistical visualization. Perhaps the knowledge of a statistical-visualization expert can be packaged in an automated form that enables a domain expert to create effective visualizations? Third, we encourage research into automated support for causality analysis. In this study, establishing causality required iteration of the scientific method by a domain expert. This approach was error-prone, time-consuming and difficult.

DRAFT