Chapter 7 – Comparing Congestion Control Regimes in a Scaled-Down Network

7 Comparing Congestion Control Regimes in a Scaled-Down Network

In this section, we repeat the previous experiment (from Chapter 6) making a few parameter changes and including an additional congestion control regime: FAST with α-tuning enabled. Our parameter changes include reducing network size (number of sources and receivers) and speed by about an order of magnitude and reducing the initial slow-start threshold to a relatively low value. As shown in Table 7-1, we retain the algorithm identifiers from Chapter 6, adding FAST with α-tuning enabled (FAST-AT) as algorithm number eight. We also retain the topology (Fig. 6-1), scenario (Fig. 6-2), path classes (Table 6-2), and fixed parameters for the network (Table 6-4), for simulation control (Table 6-10), for user traffic (Table 6-11) and for long-lived flows (Table 6-12). In addition, we measure the same responses defined in Chapter 6 (recall Tables 6-15 through 6-25). We collect data in the same fashion as described in Sec. 6.2.2. We also adopt the same fundamental approach to data analysis (as described in Sec. 6.3).

Table 7-1. Congestion Control Mechanisms Compared

Identifier	Label	Name of Congestion Avoidance Algorithm
1	BIC	Binary Increase Congestion Control
2	CTCP	Compound Transmission Control Protocol
3	FAST	Fast Active-Queue Management Scalable Transmission Control Protocol
4	HSTCP	High-Speed Transmission Control Protocol
5	HTCP	Hamilton Transmission Control Protocol
6	Scalable	Scalable Transmission Control Protocol
7	TCP	Transmission Control Protocol (Reno)
8	FAST-AT	FAST with α-tuning Enabled

We expect the scaled-down network simulation to require an order of magnitude fewer resources, while confirming the main findings from simulating a large, fast network. Scaling down network size and speed by a similar factor should generate conditions with relative congestion aligned to those described in Chapter 6 (recall Figs. 6-5 through 6-8). We also expect FAST-AT to exhibit similar response to congested conditions as FAST. This expectation arises from the fact that FAST and FAST-AT showed similar oscillatory behavior when simulated in a single-path topology with insufficient buffers. We also expect that lowering the initial slow-start threshold from $2^{31}/2$ to 100 packets will have only a limited effect on our results. This expectation arises from the fact that Web objects retain an average size of no more than 100 packets, which should enable the transfer of Web objects to complete under initial slow start during uncongested conditions. In uncongested conditions, document transfers, with average size of up to 10^3 packets, might be affected by the lower initial slow-start threshold, but such flows make up only about 1 % of all transfers. During congested conditions initial slow

start tends to end before the congestion window reaches the initial slow-start threshold. We do expect a lower initial slow-start threshold to have a negative influence on long-lived flows that use standard TCP congestion avoidance. Long-lived flows have an infinite size, so congestion avoidance procedures are activated before such flows reach a maximum achievable transfer rate. Since standard TCP congestion avoidance procedures lead to a linear increase in the congestion window, long-lived TCP flows transiting high-delay paths should take quite some time to achieve maximum rate.

We organize what follows into six sections. Sec. 7.1 describes the experiment design, concentrating on changes from the previous experiment where we simulated a large, fast network. Sec. 7.1 also explains how the revised experiment design influences the domain view of the simulated network. Sec. 7.2 compares resource requirements for simulating a large, fast network against resource requirements for simulating a scaled-down network. Sec. 7.3 explains the nature of (and rationale for) a tactical change in the data analysis approach we used to investigate the scaled-down network. Sec. 7.4 presents selected results from simulating a scaled-down network, while Sec. 7.5 discusses key findings from the results. We conclude in Sec. 7.6.

7.1 Experiment Design

We adopt the same 2^{6-1} orthogonal fractional factorial design template (see Table 6-13) used in Chapter 6. As discussed below, we change only one robustness factor and two fixed parameters and then instantiate the design template to create 32 simulated conditions.

7.1.1 Changes in Robustness Factors and Fixed Factors

Table 7-2 specifies the robustness factors and values we used for this experiment. Recall that robustness factors define the range of parameter combinations over which experiment conclusions will hold. We highlight (in red) our changes (from Table 6-3) to robustness factor x1 (network speed). These changes result in a network that operates at only 15 % of the speed we simulated in Chapter 6.

Table 7-2. Robustness Factors Adopted for Comparing Congestion Control Mechanisms

Identifier	Definition	PLUS (+1) Value	Minus (-1) Value
x1	Network Speed	1200 packets/ms	600 packets/ms
x2	Think Time	5000 ms	2500 ms
x3	Source Distribution	Uniform (.33/.33/.33)	Skewed (.1/.6/.3)
x4	Propagation Delay	2	1
x5	File Size	100 packets	50 packets
x6	Buffer Sizing Algorithm	RTTxCapacity	RTTxCapacity/SQRT(*N*)

We make only two other parameter changes from our previous experiment. The changes, shown below in red in Table 7-3, affect fixed parameters related to sources and receivers. We reduce the base number of sources under an access router from 1000 to 100 and we reduce the initial slow-start threshold from $2^{31}/2$ to 100 packets.

7.1.2 Orthogonal Fractional Factorial Design of Robustness Conditions

We inject the robustness factors from Table 7-2 into the design template from Table 6-13 to yield 32 instantiated robustness conditions, as shown in Table 7-4. Changes in the robustness conditions (from Table 6-14) are emphasized in red.

Table 7-3. Fixed Parameters Related to Sources and Receivers

Parameter	Definition	Value
Bsources	Basic unit for sources per access router	100
ΔU	Avg. sources per access router = Bsources x ΔU	2
P(Nr)	Probability receiver under normal access router	0.6
P(Nrf)	Probability receiver under fast access router	0.2
P(Nrd)	Probability receiver under directly connected access router	0.2
sst_{INT}	Initial slow-start threshold (packets)	100

Table 7-4. Instantiated Robustness Conditions

Factor->	X1	X2	X3	X4	X5	X6
Condition	--	--	--	--	--	--
1	600	2500	.1/.6/.3	1	50	RTTxCapacity/SQRT(N)
2	1200	2500	.1/.6/.3	1	50	RTTxCapacity
3	600	5000	.1/.6/.3	1	50	RTTxCapacity
4	1200	5000	.1/.6/.3	1	50	RTTxCapacity/SQRT(N)
5	600	2500	.3/.3/.3	1	50	RTTxCapacity
6	1200	2500	.3/.3/.3	1	50	RTTxCapacity/SQRT(N)
7	600	5000	.3/.3/.3	1	50	RTTxCapacity/SQRT(N)
8	1200	5000	.3/.3/.3	1	50	RTTxCapacity
9	600	2500	.1/.6/.3	2	50	RTTxCapacity
10	1200	2500	.1/.6/.3	2	50	RTTxCapacity/SQRT(N)
11	600	5000	.1/.6/.3	2	50	RTTxCapacity/SQRT(N)
12	1200	5000	.1/.6/.3	2	50	RTTxCapacity
13	600	2500	.3/.3/.3	2	50	RTTxCapacity/SQRT(N)
14	1200	2500	.3/.3/.3	2	50	RTTxCapacity
15	600	5000	.3/.3/.3	2	50	RTTxCapacity
16	1200	5000	.3/.3/.3	2	50	RTTxCapacity/SQRT(N)
17	600	2500	.1/.6/.3	1	100	RTTxCapacity
18	1200	2500	.1/.6/.3	1	100	RTTxCapacity/SQRT(N)
19	600	5000	.1/.6/.3	1	100	RTTxCapacity/SQRT(N)
20	1200	5000	.1/.6/.3	1	100	RTTxCapacity
21	600	2500	.3/.3/.3	1	100	RTTxCapacity/SQRT(N)
22	1200	2500	.3/.3/.3	1	100	RTTxCapacity
23	600	5000	.3/.3/.3	1	100	RTTxCapacity
24	1200	5000	.3/.3/.3	1	100	RTTxCapacity/SQRT(N)
25	600	2500	.1/.6/.3	2	100	RTTxCapacity/SQRT(N)
26	1200	2500	.1/.6/.3	2	100	RTTxCapacity
27	600	5000	.1/.6/.3	2	100	RTTxCapacity
28	1200	5000	.1/.6/.3	2	100	RTTxCapacity/SQRT(N)
29	600	2500	.3/.3/.3	2	100	RTTxCapacity
30	1200	2500	.3/.3/.3	2	100	RTTxCapacity/SQRT(N)
31	600	5000	.3/.3/.3	2	100	RTTxCapacity/SQRT(N)
32	1200	5000	.3/.3/.3	2	100	RTTxCapacity

7.1.3 Domain View of Robustness Conditions

Changes in simulated speed and size influence the domain view of our simulated network. Table 7-5 shows the simulated router speeds for this experiment, which are reduced by about an order of magnitude over the values given in Table 6-5. Changes in router speeds are highlighted in red in Table 7-5. Reduction in **Bsources** (base number of sources) leads to an order of magnitude fewer sources, as shown in Table 7-6, which highlights in red changes from Table 6-9.

Table 7-5. Domain View of Router Speeds

Router	PLUS (+1)	Minus (-1)
Backbone	28.8 Gbps	14.4 Gbps
POP	3.6 Gbps	1.8 Gbps
Normal Access	360 Mbps	180 Mbps
Fast Access	720 Mbps	360 Mbps
Directly Connected Access	3.6 Gbps	1.8 Gbps

Table 7-6. Number of Simulated Sources

PLUS (+1)	Minus (-1)
27.8×10^3	17.46×10^3

We use the same topology as the previous experiment and we simulate the same propagation delays (shown in Table 6-6). Recall, though, that buffer sizing is influenced by up to three factors: network speed, propagation delay and number of sources. Since we have changed two of these factors, simulated buffer sizes also change, as shown in Table 7-7, which highlights in red changes in buffer sizes from the previous experiment (see Table 6-7).

Table 7-7. Buffer Sizes Simulated

	PLUS (+1)			Minus (-1)		
Router	Min	Avg	Max	Min	Avg	Max
Backbone	48.830×10^3	109.866×10^3	195.317×10^3	547	1.236×10^3	2.208×10^3
POP	6.104×10^3	13.734×10^3	24.415×10^3	105	240	431
Access	971	2.184×10^3	6.104×10^3	44	99	105

Overall, the order of magnitude reduction in network speed and number of sources should scale the network model so that simulated conditions exhibit about the same relative congestion levels as the previous experiment. Fig. 7-1 plots the retransmission rates (y axis) for each of the 32 simulated conditions. The x axis is plotted in order of increasing retransmission rate. Comparing this plot with Fig. 6-5, we see that

the order of the conditions shifts around slightly. Congestion levels in the scaled-down network are reduced somewhat, as can be seen by the fact that the most congested condition (21) has a retransmission rate of around 30 % as compared with 50 % for the larger network. Uncongested conditions in the scaled-down network have retransmission rates two orders of magnitude lower than exhibited by the larger network. Selected conditions with moderate congestion (4, 10, 16, 28 and 11) exhibit increased retransmission rates in the scaled-down network, as compared with the larger network. In fact, condition 11 can now be considered congested – implying that the scaled-down network has 17 congested conditions, labeled **C** in Fig. 7-1, as compared with 16 congested conditions in the previous experiment (see Fig. 6-5). As one can see, condition 11 exhibits a substantially higher retransmission rate than condition 28 but a substantially lower rate than condition 22. We arbitrarily divide the remaining conditions into three categories reflecting no (**N**), limited (**L**) and moderate (**M**) congestion.

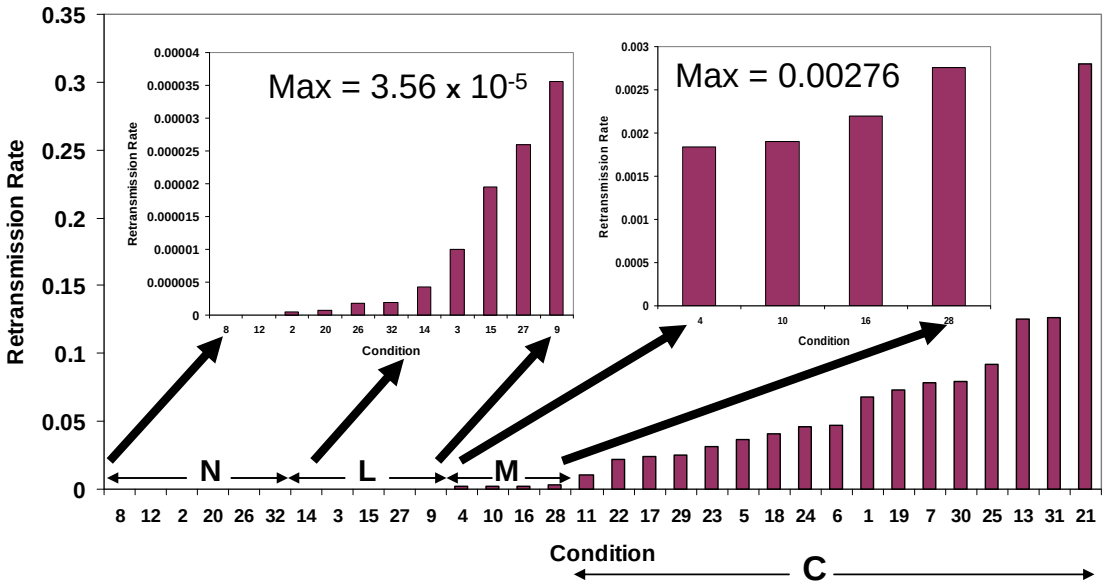

Figure 7-1. Retransmission Rate (proportion of retransmitted packets) vs. Simulated Conditions Ordered from Least to Most Congested

The range of retransmission rates in Fig. 7-1 spans about six orders of magnitude, as was the case in Fig. 6-5. On the other hand, the two least congested conditions (8 and 12) in Fig. 7-1 show no retransmissions. In the large, fast network (Fig. 6-5) condition 12 had 6 retransmissions in 10^9 packets and condition 8 had 6 retransmissions in 10^7 packets. This difference can be attributed to the fact that the larger network typically had an order of magnitude more active flows and a higher slow-start threshold, both of which increase the likelihood of lost packets.

As with the previous experiment, we select one uncongested and one congested condition to examine more closely. For the large, fast network we examined conditions 4 (Fig. 6-6) and 5 (Fig. 6-7) under standard TCP congestion control. For the scaled-down network we also examine congested condition 5, but we select uncongested condition 3 because condition 4 has moved from the category of little congestion (in Fig. 6-5) to moderate congestion (in Fig. 7-1).

Fig. 7-2 plots several time series that show the change in flow states for uncongested condition 3 under standard TCP congestion control. The x axis displays time (in 200 ms intervals since the beginning of the simulation) over the three time periods (final 15 simulated minutes) measured for the scenario. The y axis indicates the number of active (red curve) and connecting (yellow curve) flows. Additional curves decompose active flows by congestion control states: initial slow start (green curve) and normal congestion control (brown curve). Fig. 7-2 resembles Fig. 6-6, which plots the change in flow states for uncongested condition 4 in the large, fast network. The network is sufficiently uncongested that most transfers during TP1 (3000-4500) complete in initial slow start. Things change during TP2 (4500-6000) as jumbo transfers induce congestion in directly connected access routers. Congestion leads to losses, which increases the number of flows operating under normal congestion control procedures. As jumbo transfers diminish during TP3 (6000-7500), congestion decreases so that, by time t = 6600, most active flows again complete transfers without packet loss. Plots for other uncongested conditions show similar patterns.

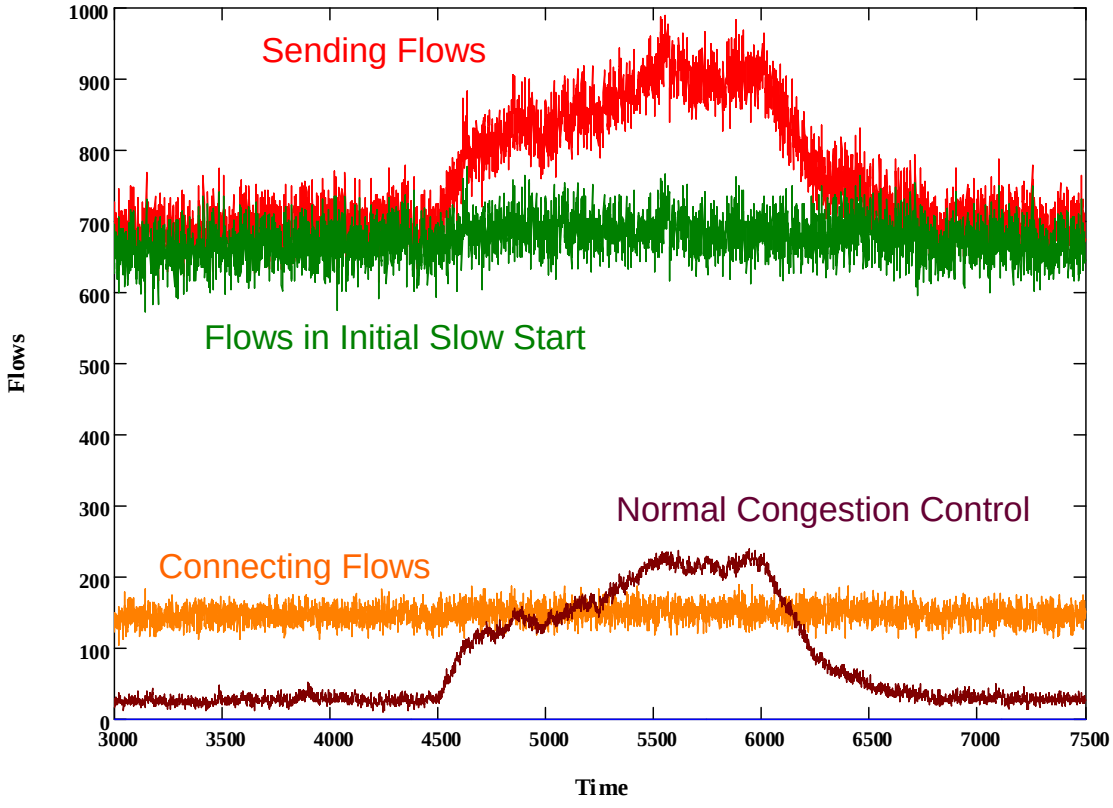

Figure 7-2. Change in Flow States over Three Time Periods under Condition 3 for Standard TCP – x axis gives time in 200 ms increments from the beginning of the simulation, covering the final 15 minutes of the simulated scenario, and y axis gives the number of flows

Fig. 7-3 illustrates the change in flow states for condition 5, a representative congested condition. The number of active flows (red curve) shows an order of magnitude increase over uncongested condition 3. Comparing Fig. 7-3 with Fig. 6-7 reveals some similarities and some differences. Both plots show that network congestion is sufficiently high that introducing jumbo transfers in TP2 (4500-6000) makes little

difference in the overall distribution of flow states. In the scaled-down network, about 60 % of active flows operate under normal congestion control (brown curve) and 40 % operate in initial slow start (green curve). On the other hand, Fig. 6-7 shows that under the large, fast network about 85 % of active flows operate in normal congestion control. This difference occurs over the range of all congested conditions until the three most congested conditions (13, 31 and 21). Under these highest levels of congestion, the relative proportion of active flows using normal congestion control appears similar for both the large and scaled-down network.

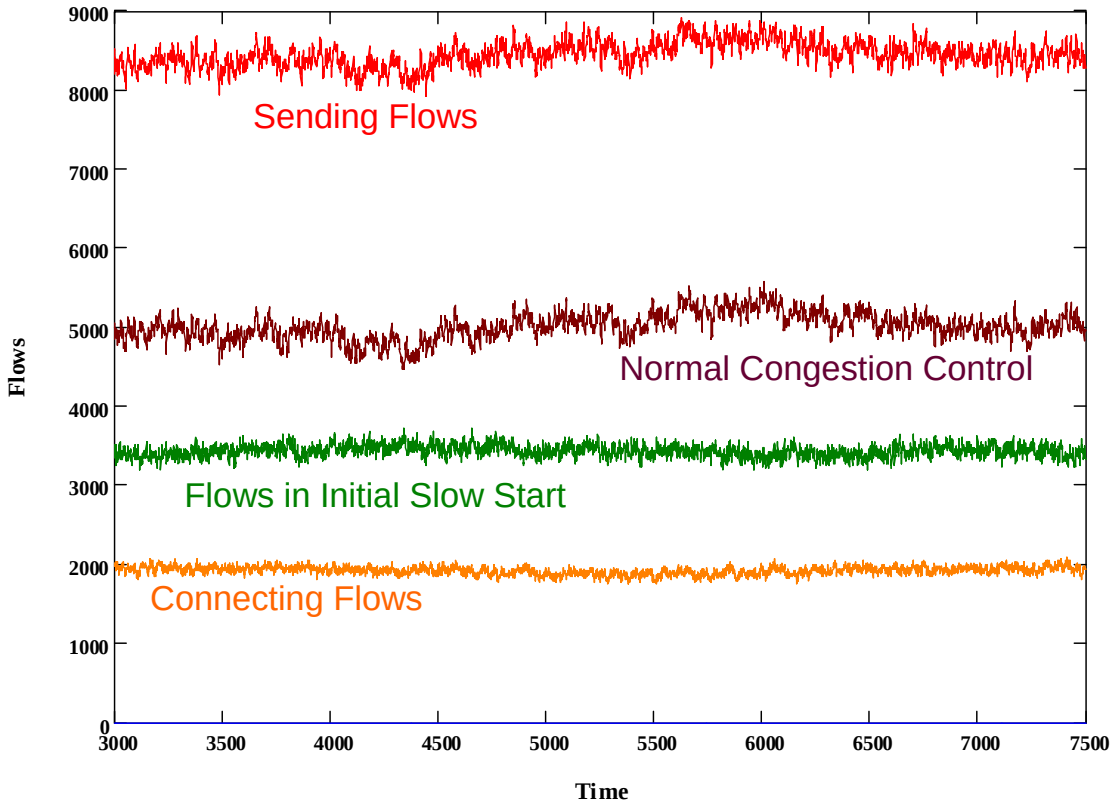

Figure 7-3. Change in Flow States over Three Time Periods under Condition 5 for Standard TCP – x axis gives time in 200 ms increments from the beginning of the simulation, covering the final 15 minutes of the simulated scenario, and y axis gives the number of flows

To further understand similarities and differences in conditions created by the scaled-down network versus the large, fast network, we can plot a cluster analysis (similar to Fig. 6-8) for each of the 32 conditions and eight congestion control algorithms across all response variables and then label each condition with the congestion class identified in Fig. 7-1. We show the annotated cluster analysis as Fig. 7-4, which encompasses the first time period (TP1). Comparing Fig. 7-4 and Fig. 6-8 we find that 27 of the 32 conditions retain the same classification in both figures. Three conditions (9, 26 and 32) remain classified as uncongested but fall into the next lower congestion category as we scale down the network simulation. One condition (4) remains classified as uncongested but with a movement to the next higher congestion category (from **L** to **M**). Only condition 11 becomes congested from uncongested (**M**).

The foregoing analysis demonstrates that scaling down the network model leads to congestion conditions that show reasonable alignment with the large, fast network. Overall, conditions, especially uncongested conditions, tend to be less congested under the scaled-down network. This lessening of congestion can be attributed to a lower initial slow-start threshold and a decrease in the number of active flows throughout the network.

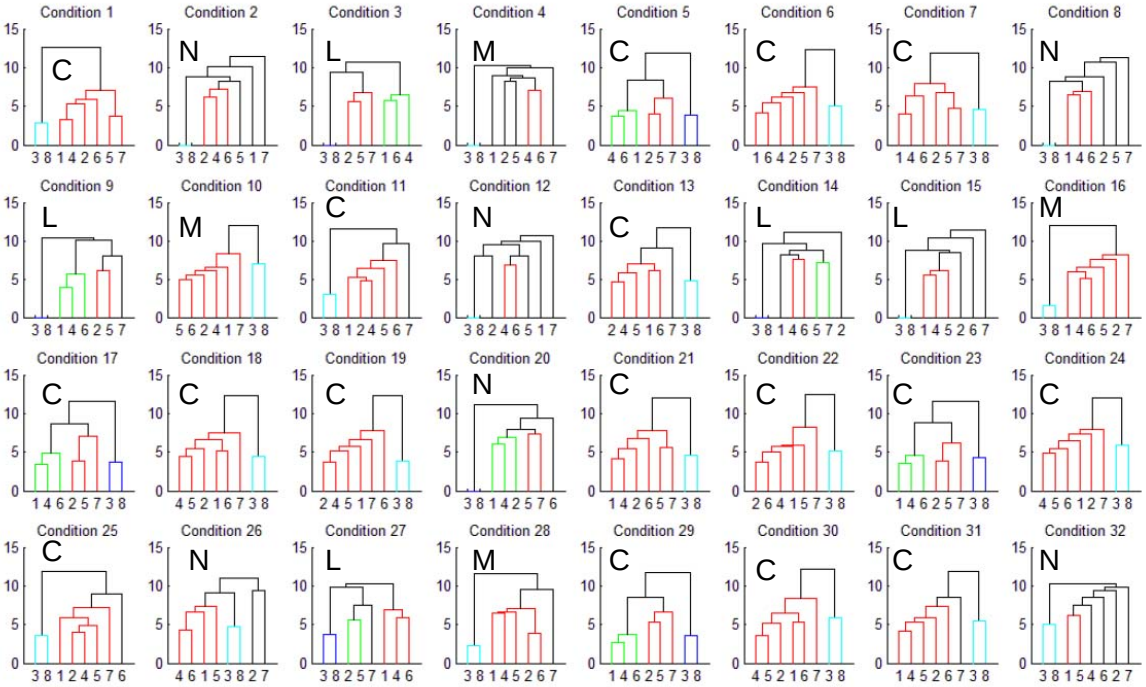

Figure 7-4. Cluster Analysis for Time Period One – x axis on each sub-plot gives algorithm identifier from Table 7-1 and y axis shows distance in the response space between pairs of algorithms and clusters of algorithms as indicated - each sub-plot also labeled with Congestion Level

7.2 Experiment Execution and Data Collection

Table 7-8 compares processing and memory requirements for simulating the FAST congestion control algorithm in the scaled-down network against resource requirements for simulating FAST in a large, fast network (Chapter 6). All simulations compared in Table 7-8 were executed on identical compute servers (ws11 through ws14, described in Table 6-26). As expected, scaling down the network simulation by an order of magnitude leads to a similar reduction in the processing time and memory usage. Table 7-9 shows that this reduction in resource usage arises because the scaled-down network simulates 10 times fewer flows and packets per run. We collect data as described in Sec. 6.2.2, so reducing the scale of the simulation does not reduce the amount of data collected.

7.3 Data Analysis Approach

While we use the same fundamental data analysis approach described in Sec. 6.3, we adopt a tactical change in order to enhance clarity. Careful review of Fig. 7-4 reveals that algorithms 3 (FAST) and 8 (FAST with α-tuning enabled) exhibit similar performance under all conditions. In fact, the cluster analysis groups the two algorithms together into the same exclusive cluster for each of the 32 conditions. This indicates that the two

algorithms respond similarly to similar conditions. This should come as no surprise because algorithms 3 and 8 share the same underlying FAST procedures, differing only with respect to treatment of the α parameter.

Table 7-8. Comparing Resource Requirements for Simulating the FAST Congestion Control Algorithm in a Large, Fast Network and a Scaled-Down Network

	Large, Fast Network	Scaled-Down Network
CPU hours (32 Runs)	2241.6	197.3
Avg. CPU hours (per run)	70.1	6.2
Min. CPU hours (one run)	34.6	2.52
Max. CPU hours (one run)	124.1	12.3
Avg. Memory Usage (Mbytes)	1250	139

Table 7-9. Comparing Number of Simulated Flows and Packets for a Large, Fast Network and a Scaled-Down Network under All Congestion Control Algorithms

	Large, Fast Network Chapter 6		Scaled-Down Network Chapter 7	
Statistic	Flows Completed	Data Packets Sent	Flows Completed	Data Packets Sent
Avg. Per Condition	74.033×10^6	6.912×10^9	8.329×10^6	897.379×10^6
Min. Per Condition	40.966×10^6	3.146×10^9	4.329×10^6	380.349×10^6
Max. Per Condition	154.915×10^6	11.917×10^9	16.730×10^6	1.749×10^9
Total All Runs	16.583×10^9	1.548×10^{12}	2.132×10^9	229.729×10^9

Similarity in responses for algorithms 3 and 8 lead to paired, extreme response values under many conditions. As a result, responses for the two algorithms are not necessarily distinguished as significant outliers by a Grubbs' test. For example, examine Fig. 7-5, which gives a detailed analysis of the retransmission rate among all algorithms and conditions during the first time period (TP1). The figure shows that algorithms 3 and 8 have extreme, high retransmission rates under congested conditions. Note, however, that in none of these conditions does a response satisfy our selected cutoff (> 2.08 deviations from the residual mean) for the Grubbs' test. The existence of two, similar, extreme values for two related congestion control algorithms inflates the mean, which causes the Grubbs' value to reach only 1.6. This prevents automated highlighting (green for high and red for low) of either algorithm as an outlier. Without highlighting, the detailed analyses become more difficult to interpret, especially because similar response values for algorithms 3 and 8 can lead to an overstrike of the two numbers on the plots, as shown in Fig. 7-5.

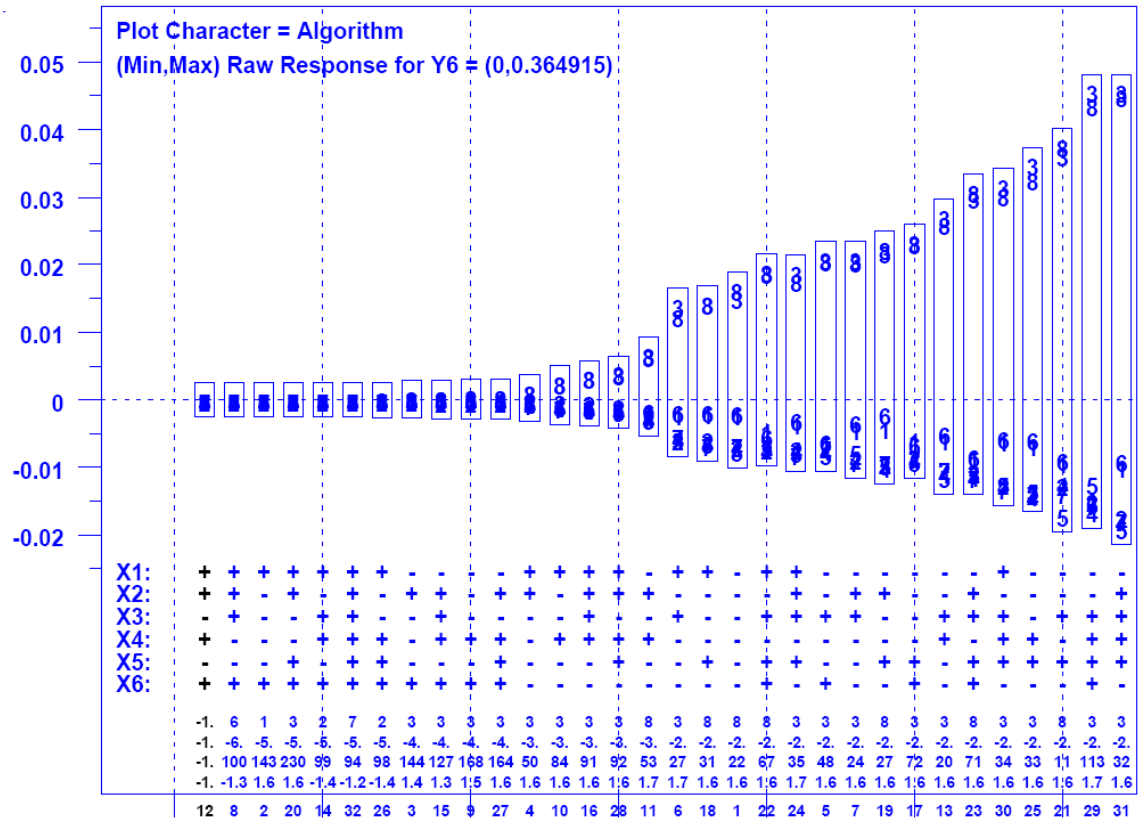

Figure 7-5. Detailed Analysis of Retransmission Rate (proportion of packets retransmitted) in Time Period One for All Algorithms – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

To enhance clarity, we adopt an approach that enables algorithm 8 to be highlighted as an outlier wherever both algorithms exhibit similar extreme responses. We omit responses for algorithm 3 from the detailed analyses. Omitting responses for algorithm 3 reduces the values of the mean residuals and thus enables responses for algorithm 8 to exceed our chosen 2.08 cutoff under the Grubbs' test for outliers. For example, Fig. 7-6 shows the result of using our tactic to reconsider retransmission rate under all conditions for TP1. Note that numerous responses for algorithm 8 are now highlighted in green, which signifies a statistically significant increase in retransmission rate over the other algorithms (excluding algorithm 3).

To justify this, we point out that the cluster analyses grouped together algorithm 3 and algorithm 8 as a pair for each of the 32 conditions. We chose to omit algorithm 3 rather than algorithm 8 because results for algorithm 3 were analyzed and discussed previously in Sec. 6.4. By focusing in this section on algorithm 8, we will be better positioned to identify significant similarities in the behavior of the two algorithms. Should we wish to seek significant differences, we could conduct the detailed analyses excluding responses for algorithm 8 instead of algorithm 3 and compare the two analyses.

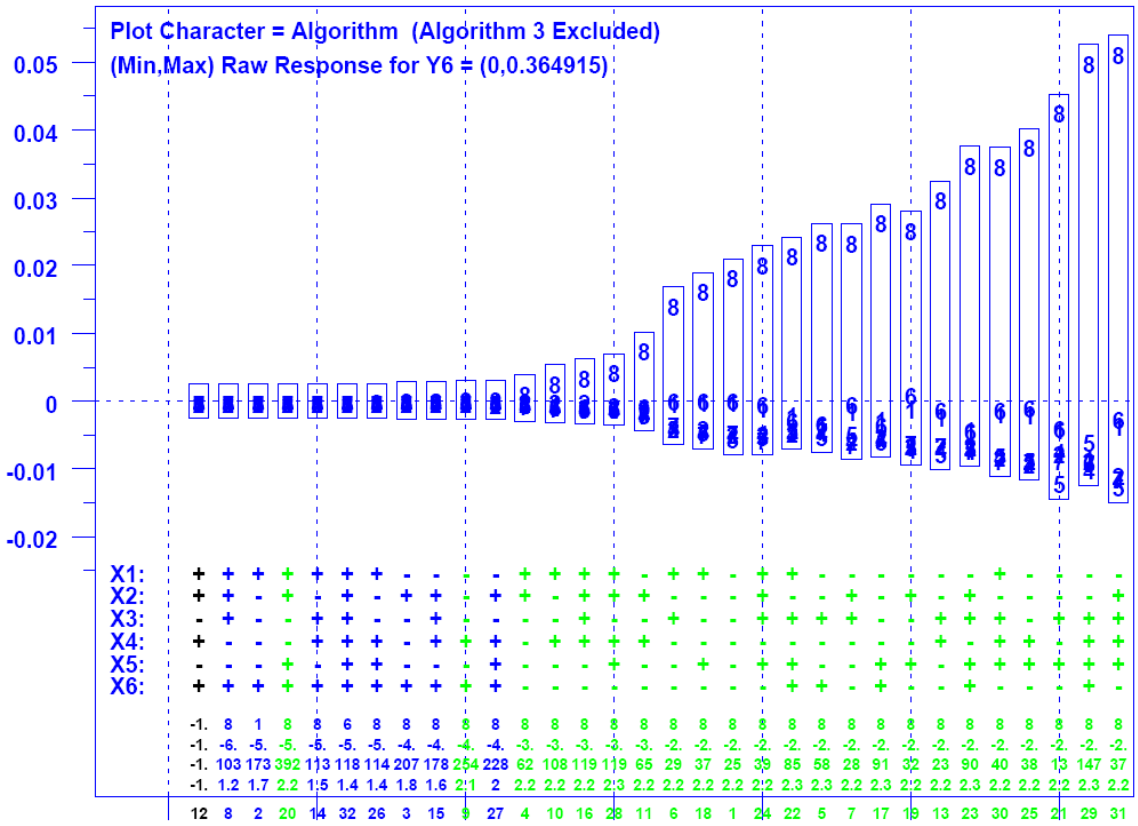

Figure 7-6. Detailed Analysis of Retransmission Rate (proportion of packets retransmitted) in Time Period One when Excluding Responses for Algorithm 3 – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

7.4 Results

In this section, we report the key results from our analysis of summarized response data (described in Sec. 6.2.2). We provide brief commentaries to explain the results presented. We give results in four segments: one for each of the three time periods and one for response data aggregated over the entire 25-minute scenario. We follow a similar plan for each segment: (1) present results from cluster analysis, (2) present results from condition-response summaries, (3) present detailed analysis of significant responses and (4) give a summary of the results for the segment. We defer drawing inferences from the results until Sec. 7.5, where we give our findings.

7.4.1 Time Period One (TP1)

TP1 comprises a five-minute period (200 ms measurement intervals 3000-4500) where three long-lived flows commence within an overall background of normal Web traffic, which includes downloading Web pages and documents. In this experiment, each long-lived flow exits initial slow-start once the congestion window passes 100 packets, so maximum transfer rate can only be achieved after entering congestion avoidance. For this reason, TP1 should reveal any differences (among the eight congestion control algorithms) in the time needed to achieve maximum transfer rate. In the previous experiment (Chapter 6), no such differences arose because the initial slow-start threshold

was set arbitrarily high (at $2^{32}/2$ packets) and all long-lived flows achieved maximum transfer rate within initial slow-start.

7.4.1.1 Cluster Analysis for TP1. Fig. 7-7 presents a cluster analysis comparing all eight algorithms (recall Table 7-1) under each of the 32 simulated conditions. We annotate the individual dendrograms with the algorithm identifier of a congestion control algorithm that stands out. Where more than one algorithm (usually 3 and 8) stands out we include all relevant algorithms separated by slashes, e.g., 3/8.

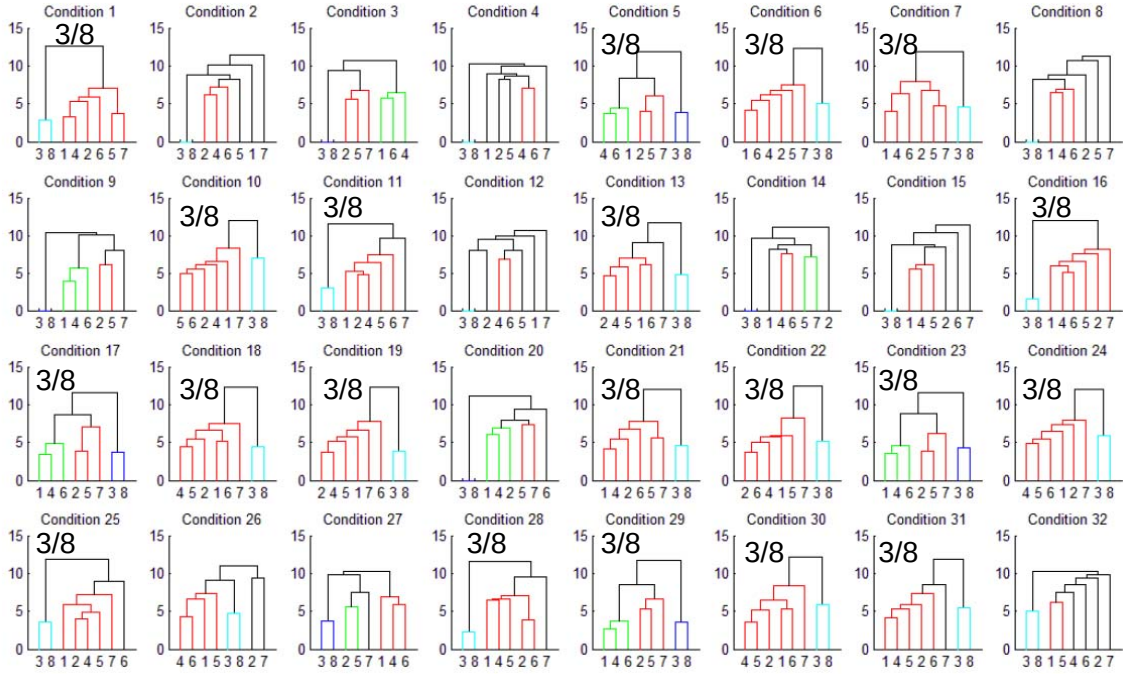

Figure 7-7. Clustering for Time Period One – x axis on each sub-plot gives algorithm identifier from Table 7-1 and y axis shows distance in the response space between pairs of algorithms and clusters of algorithms as indicated – each sub-plot Annotated to Identify Distinctive Algorithms 3/8

7.4.1.2 Condition-Response Summary for TP1. Fig. 7-8 gives the condition-response summary for TP1 – but with the data for algorithm 3 excluded in order to highlight specific responses for which algorithm 8 may be distinguished. Fig. 7-9 gives the same summary after applying a filter showing only statistically significant outliers for which the relative effect exceeds 10 %.

7.4.1.3 Analysis of Significant Responses for TP1. Based on Figs. 7-8 and 7-9, we selected responses for more detailed analysis. In Figs. 7-10 to 7-15, we report analyses for congestion window increase rate (y2), flow completion rate (y5), retransmission rate (y6), goodput (y29) and flows completed (y31) on typical (NN) paths and average number of connecting flows (y42). We selected y5 and y31 based on Fig. 7-8 even though they did not pass the 10 % filter applied to generate Fig. 7-9. When accumulated over time, the absolute magnitude of each effect within a measurement interval appears large enough to affect overall system performance.

In Figs. 7-16 to 7-18, we provide additional, detailed analyses selected using the condition-response summaries shown in Figs. 7-8 and 7-9. Fig. 7-16 reveals that algorithm 7 (TCP Reno) lags significantly in the average rate at which packets exit the network. Fig. 7-17 shows a similar lag by algorithm 7 in average goodput provided on the moderate-distance long-lived flow (L2). We omit similar analyses for the short- and long-distance long-lived flows. Finally, Fig. 7-18 explores average buffer utilization in directly connected access router K0a. We selected this analysis to show a tendency for differences in buffer utilization among the various congestion control algorithms.

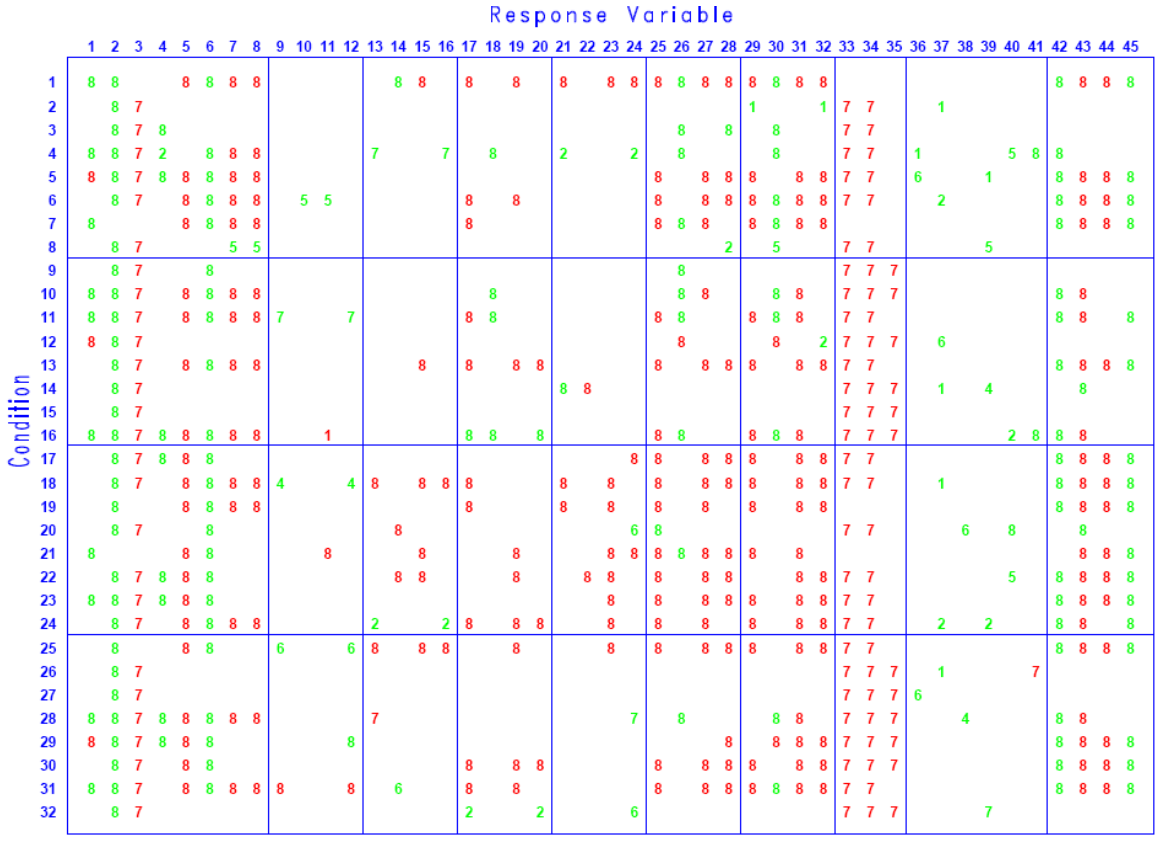

Figure 7-8. Condition-Response Summary for Time Period One – plot displays for each Factor Combination (row) vs. Response Variable (column) the Identifier of the Algorithm (from Table 7-1) manifesting a Statistically Significant Outlier (green high and red low)

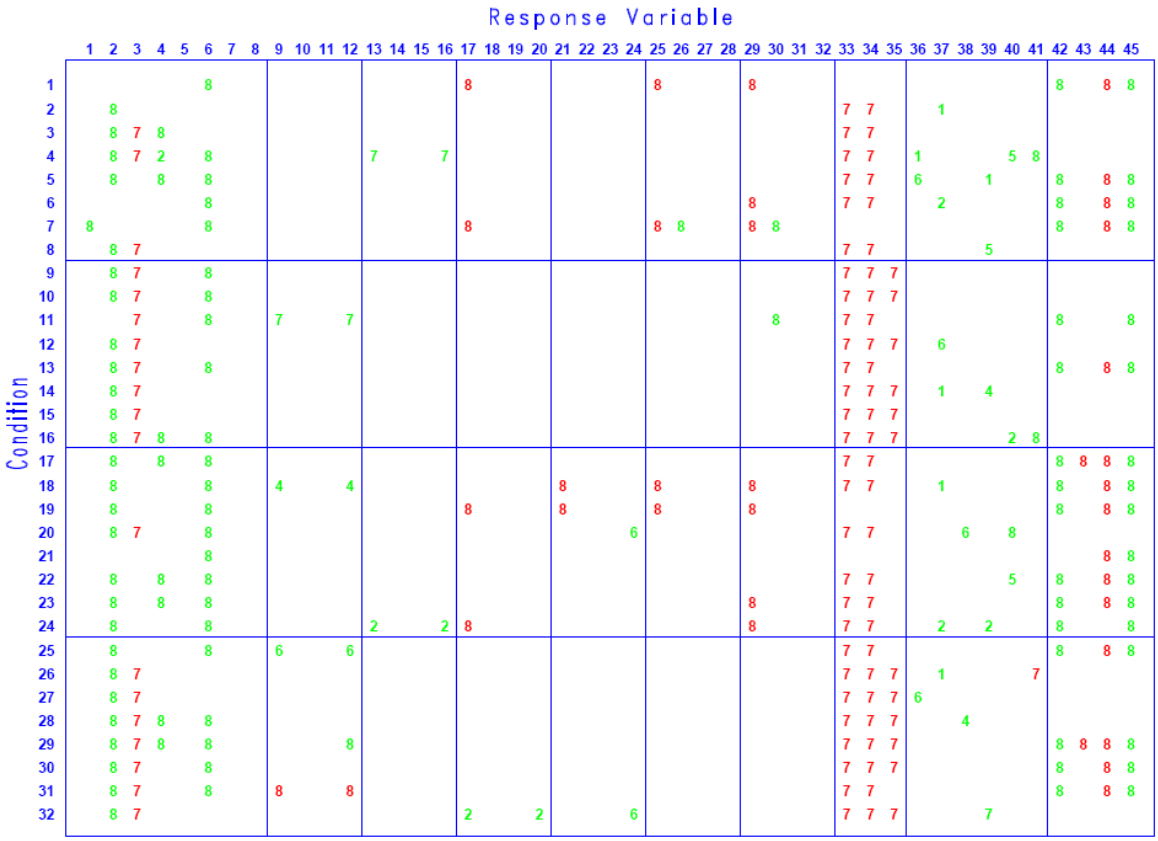

Figure 7-9. Condition-Response Summary for Time Period One – 10% Filter Applied – plot displays for each Factor Combination (row) vs. Response Variable (column) the Identifier of the Algorithm (from Table 7-1) manifesting a Statistically Significant Outlier (green high and red low)

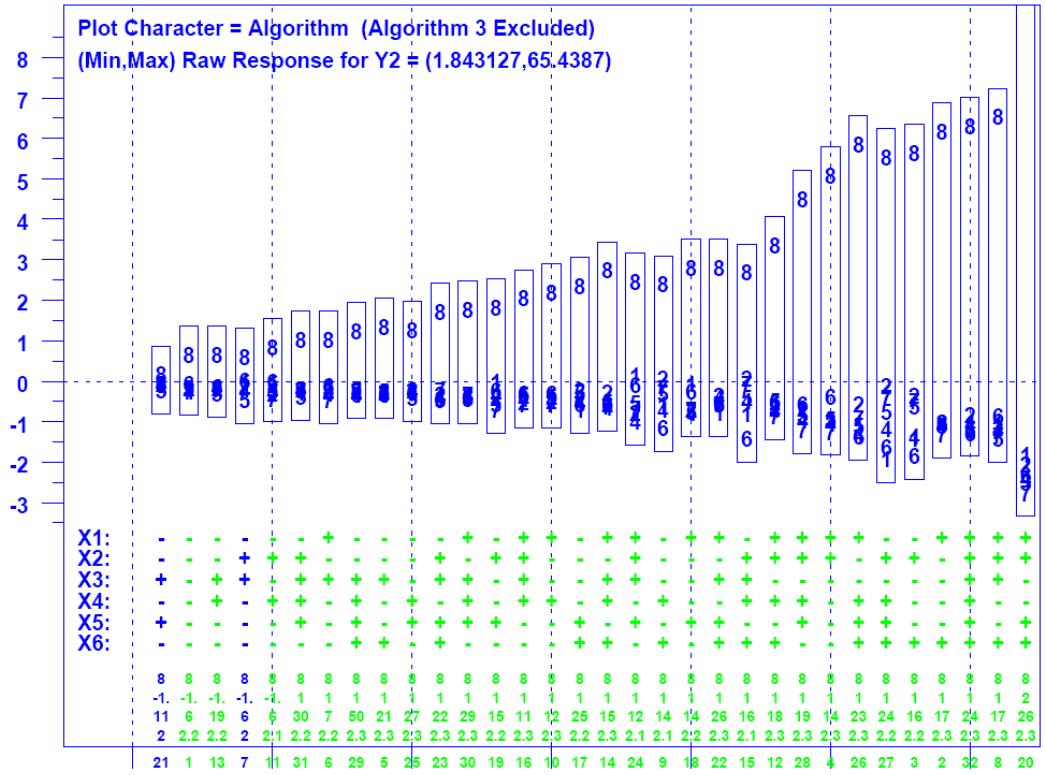

Figure 7-10. Detailed Analysis for Congestion Window Increase Rate Per Flow (increases per 200 ms) in Time Period One – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

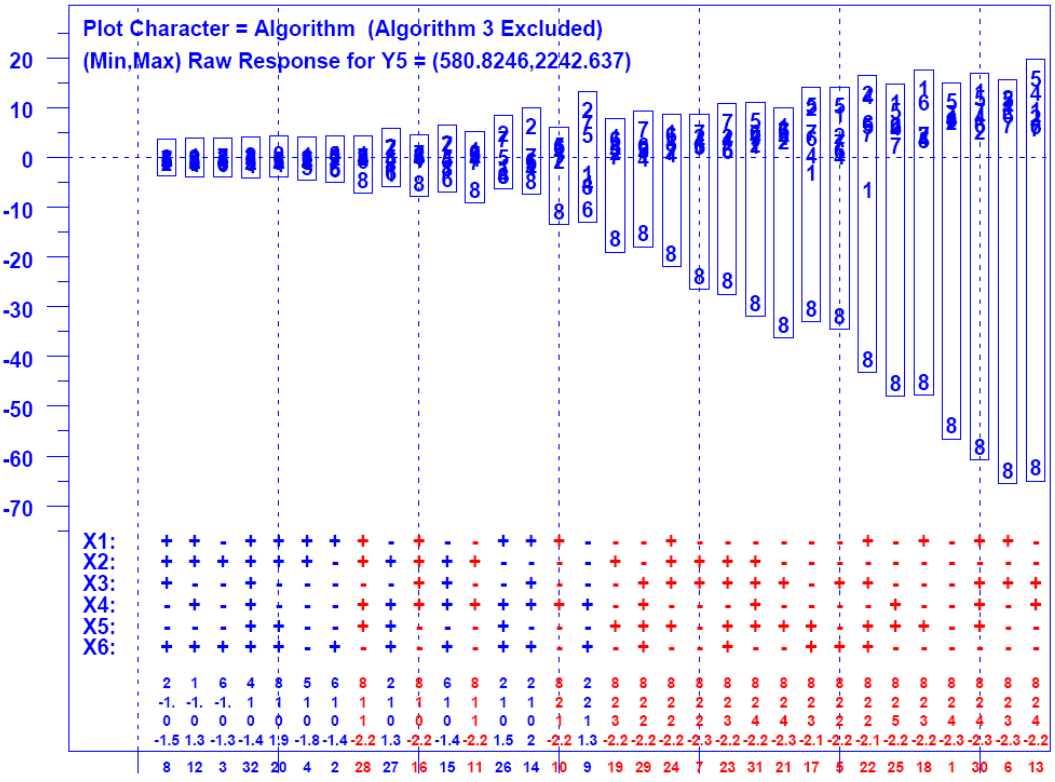

Figure 7-11. Detailed Analysis for Flow Completion Rate (flows per 200 ms) in Time Period One – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

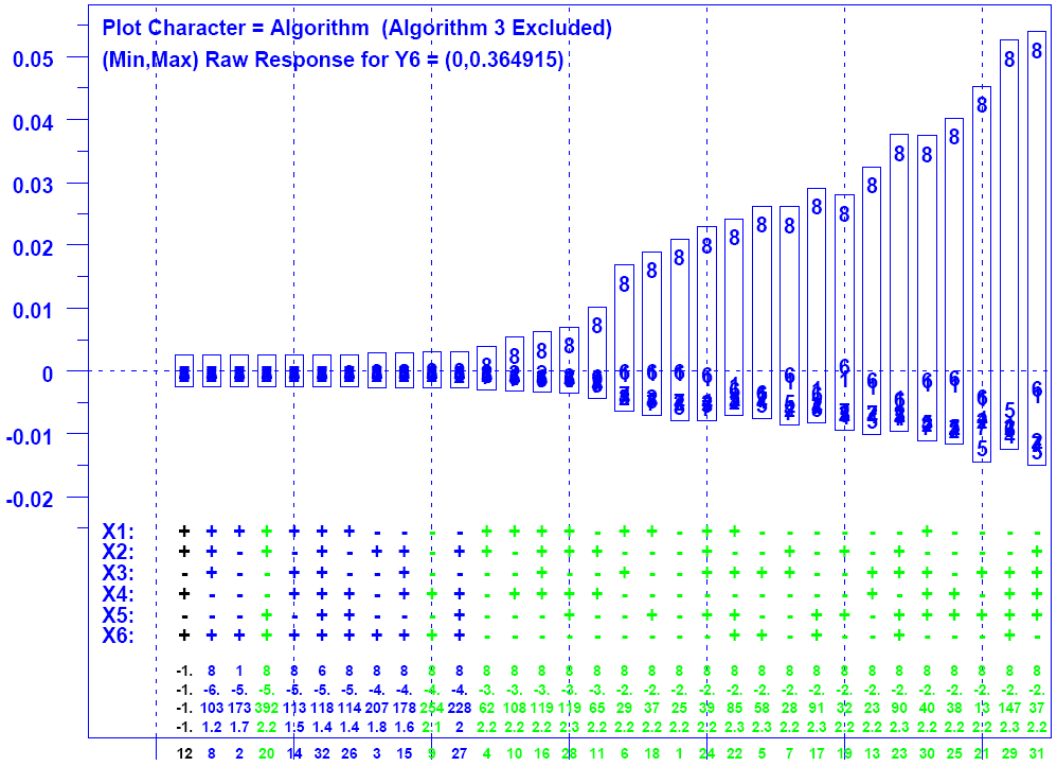

Figure 7-12. Detailed Analysis for Retransmission Rate (proportion of packets retransmitted) in Time Period One – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

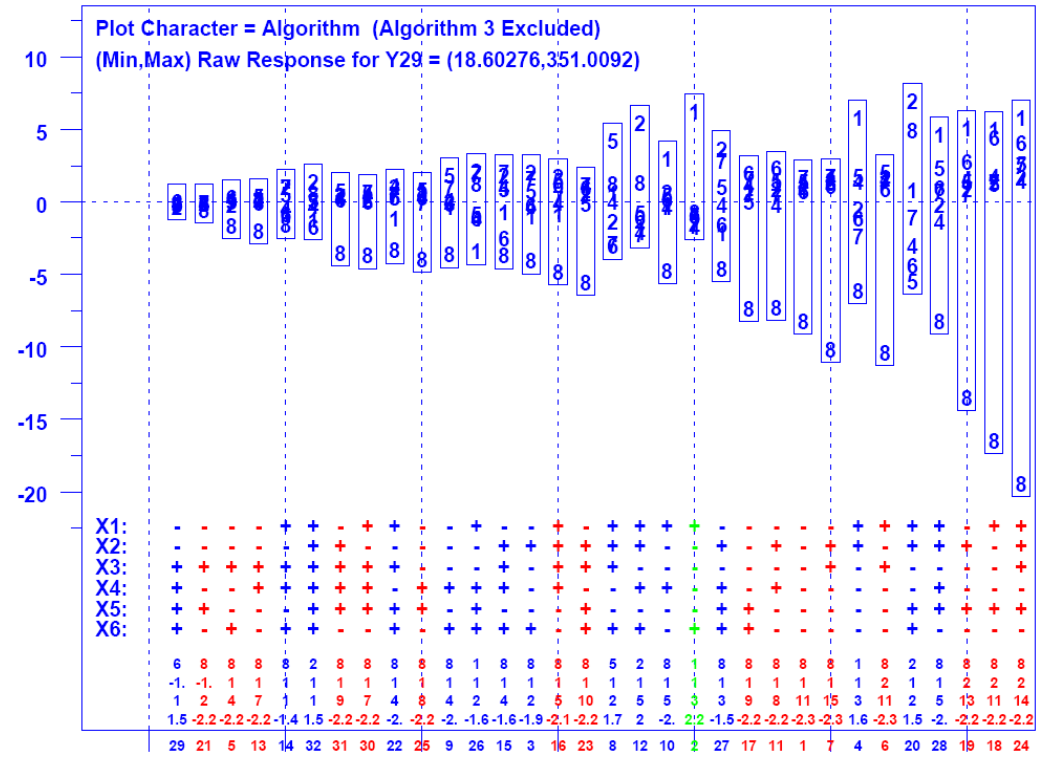

Figure 7-13. Detailed Analysis for Average Goodput (packets per second) on NN Flows in Time Period One – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

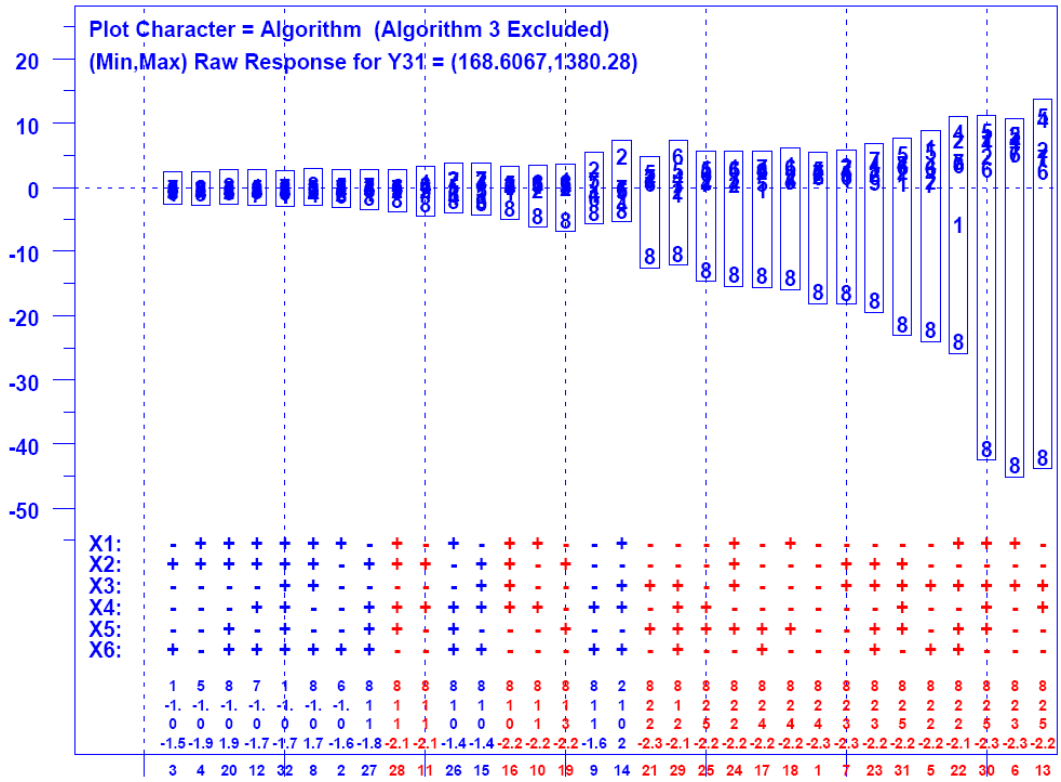

Figure 7-14. Detailed Analysis for NN Flow Completion Rate (flows per 200 ms) in Time Period One – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

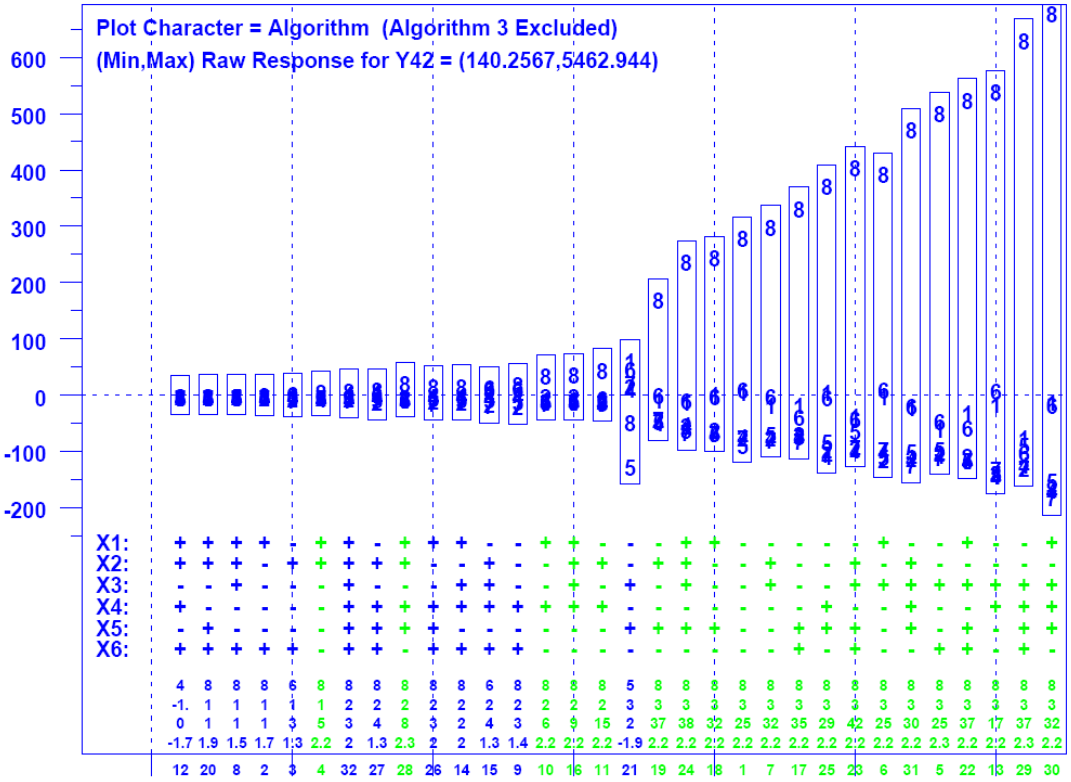

Figure 7-15. Detailed Analysis for Number of Connecting Flows in Time Period One – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

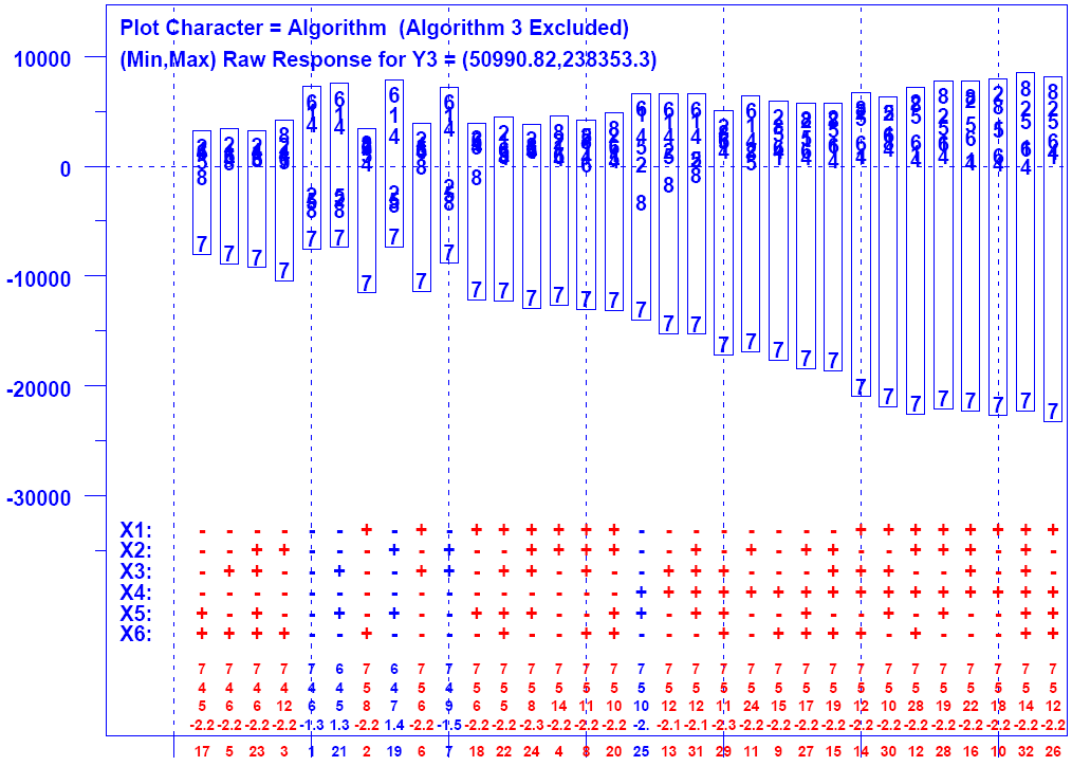

Figure 7-16. Detailed Analysis for Average Packets Output Per 200 ms Interval in Time Period One – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

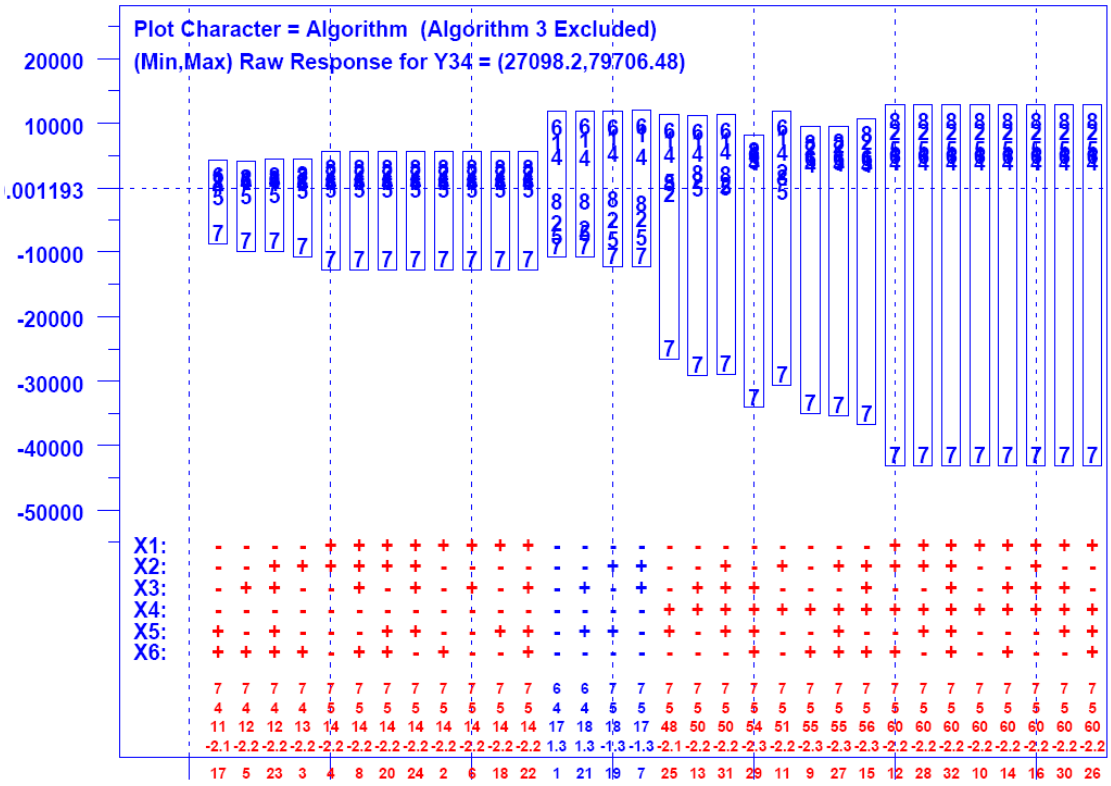

Figure 7-17. Detailed Analysis for Average Goodput (packets per second) on Long-lived Flow L2 in Time Period One – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

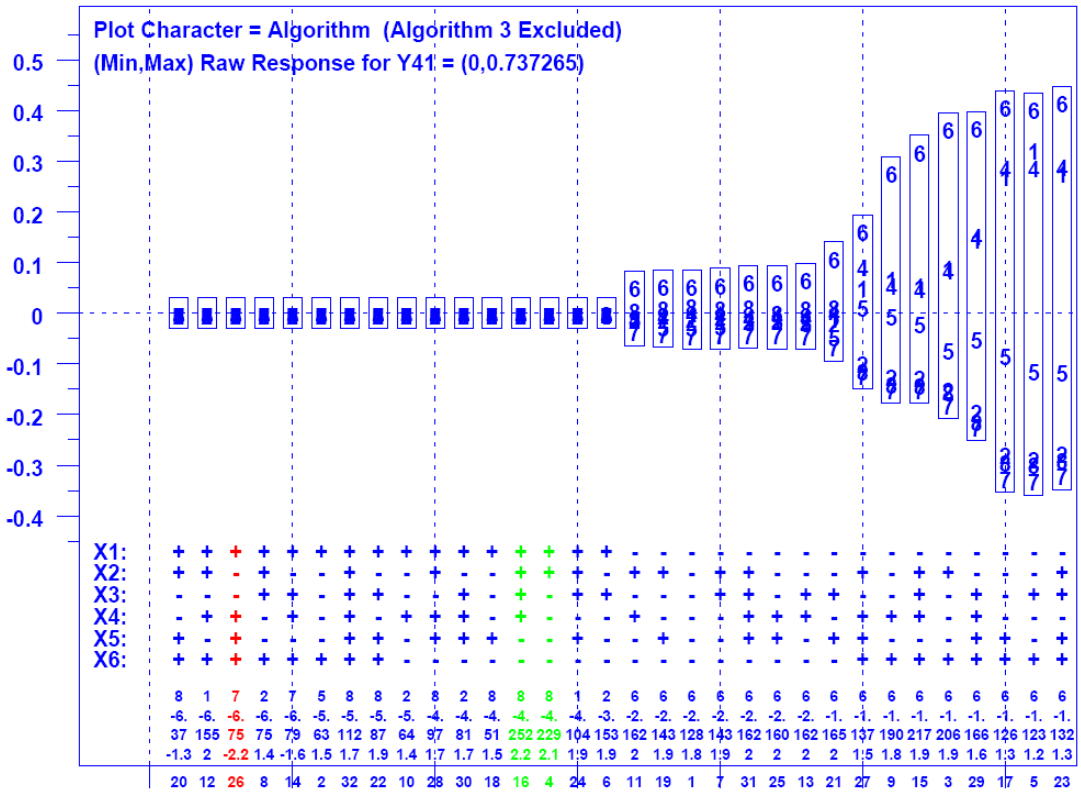

Figure 7-18. Detailed Analysis for Average Buffer Utilization (percentage of buffers occupied) at Router K0a in Time Period One – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

7.4.1.4 Summary of Results for TP1. Given normal Web traffic, FAST with α-tuning enabled (FAST-AT, i.e., algorithm 8) – exhibits distinctive behavior, which appears to grow more distinctive with increasing congestion. When faced with congestion, FAST-AT exacerbates the situation (as shown by the higher rate of increase in congestion window), which leads to more packet losses and then to a higher rate of retransmissions. Increased losses under FAST-AT also appear to increase the difficulty for establishing flows because more SYN and SYN+ACK packets are lost. As a result, on average more flows are pending in the connecting state. Increased retransmissions also cause flows to send more packets (y4) in order to ensure all data is successfully received. This means that flows take longer to finish, as shown by the lower completion rate for flows in general and for NN flows in particular. This mirrors the findings for FAST with α-tuning disabled, as reported previously (Sec. 6.4.1).

Two additional results for TP1 should be mentioned. First, as expected, when the initial slow-start threshold is set small, standard TCP Reno (algorithm 7) provides substantially lower (compared with the other algorithms) average goodput on long-lived flows. This result occurs during the period when each long-lived flow attempts to achieve maximum transmission rate using congestion avoidance increase procedures. Since most packets exiting the network during TP1 are associated with long-lived flows, the lower goodput provided by TCP Reno also leads to a lower aggregate rate of packet output. Finally, our results show that buffer utilization in routers supporting long-lived flows appears highest for Scalable TCP (algorithm 6), followed by BIC (algorithm 1) and HSTCP (algorithm 4). HTCP (algorithm 5) tends to follow next, while the other

algorithms (CTCP, FAST-AT and TCP Reno) appear to utilize the smallest proportion of buffers.

7.4.2 Time Period Two (TP2)

During the five minutes (200 ms intervals 4500 to 6000) of TP2, DD flows (explained in Sec. 6.1) become jumbo file transfers, which lead to increased congestion within directly connected access routers, and also to increased packet load on the network backbone. Jumbo file transfers compete with the three long-lived flows transiting the same directly connected access routers. The remaining flow classes continue to generate normal Web traffic and occasional document downloads during TP2.

7.4.2.1 Cluster Analysis for TP2. Fig. 7-19 shows an annotated set of 32 dendrograms for TP2. Since the level of congestion increases throughout the network during this period and algorithms 3 and 8 appear sensitive to congestion, one might expect the behavior of those algorithms to become more distinctive when compared with TP1. Note that algorithms 3 and 8 now standout under 28 of the conditions – compared with only 20 conditions in TP1. Fig. 7-19 also shows algorithm 6 (Scalable TCP) standing out in a couple of conditions.

7.4.2.2 Condition-Response Summary for TP2. Fig. 7-20 gives the condition-response summary for TP2. Fig. 7-21 gives the same summary after applying a filter showing only statistically significant outliers for which the relative effect exceeds 30 %. Remember that these figures omit data for algorithm 3. Algorithm 8 stands out in both figures and algorithms 4 (HSTCP) and 6 (Scalable TCP) tend to stand out with respect to buffer utilization (y36 through y41).

7.4.2.3 Analysis of Significant Responses for TP2. Guided by Figs. 7-20 and 7-21, we selected several responses for detailed analyses, which are reported in Figs. 7-22 through 7-30. We provide analyses for congestion window increase rate (y2), average packets output per measurement interval (y3), flow completion rate (y5), retransmission rate (y6), average goodput (y25) and completions (y27) for FN flows, average goodput for the moderate-distance (L2), long-lived flow (y34) and average number of connecting flows (y42). Taken together, these analyses highlight the key similarities and differences in behavior between TP1 and TP2. Fig. 7-30 reports the average buffer utilization for directly connected router I0a.

7.4.2.4 Summary of Results for TP2. FAST-AT exhibits the same distinctive behaviors seen during TP1. Increased congestion in TP2 enhances these effects, most of which now show up as relative differences of 30 % or more. Under congestion associated with jumbo file transfers, FAST-AT outputs the fewest packets per measurement interval under most conditions. This represents a change from TP1, when TCP Reno output the fewest packets. During TP1, the packet output rate is dominated by long-lived flows, which are attempting to achieve maximum transfer rate. Here, TCP Reno lags. During TP2, packet output rate is dominated by jumbo flows and long-lived flows, which share bandwidth at directly connected routers. Here, FAST-AT lags. Congestion also causes FAST-AT to provide lower goodput on other flows, such as FN and NN flows. FAST-AT also

completes fewer FN and NN flows per interval. Note that Scalable TCP provides highest goodput on long-lived flows and continues its tendency to use the most buffers in directly connected routers.

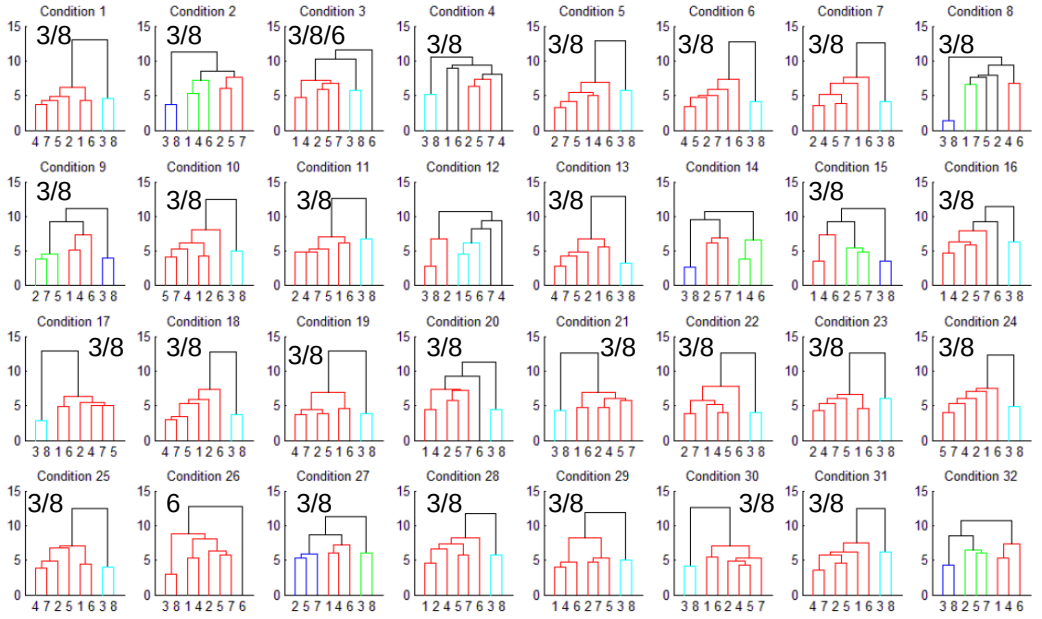

Figure 7-19. Cluster Analysis for Time Period Two – x axis on each sub-plot gives algorithm identifier from Table 7-1 and y axis shows distance in response space between pairs of algorithms and clusters as indicated – each sub-plot Annotated to Identify Distinctive Algorithms 3/8

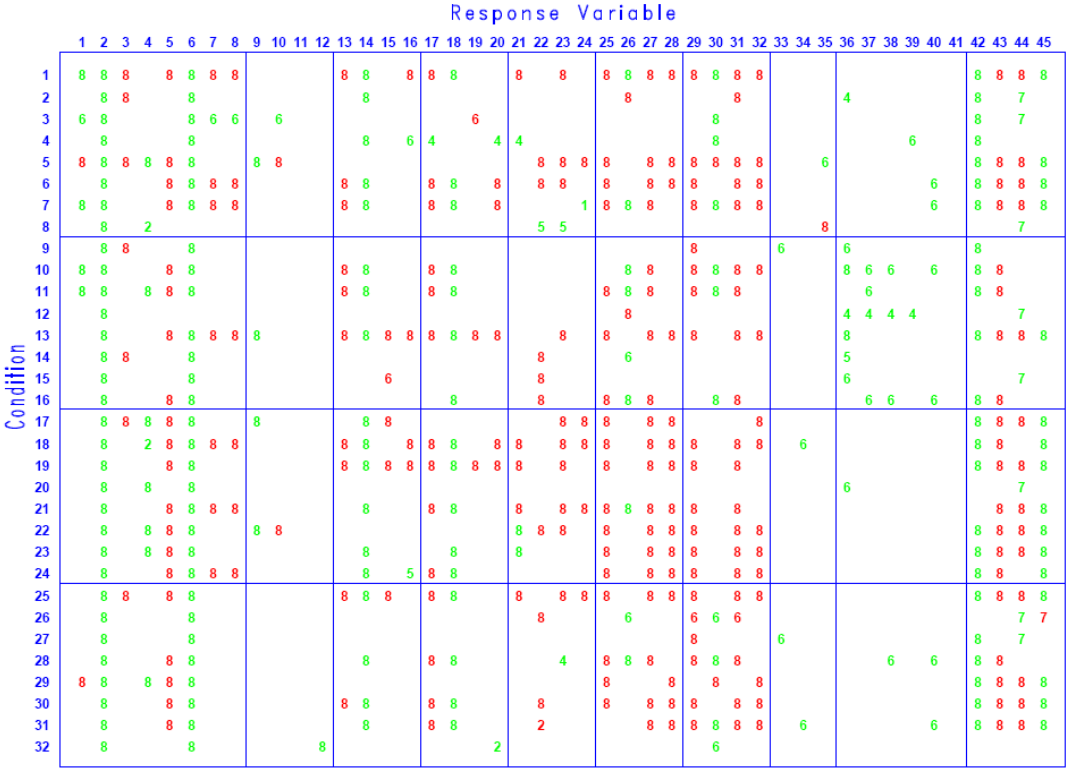

Figure 7-20. Condition-Response Summary for Time Period Two – plot displays for each Factor Combination (row) vs. Response Variable (column) the Identifier of the Algorithm (from Table 7-1) manifesting a Statistically Significant Outlier (green high and red low)

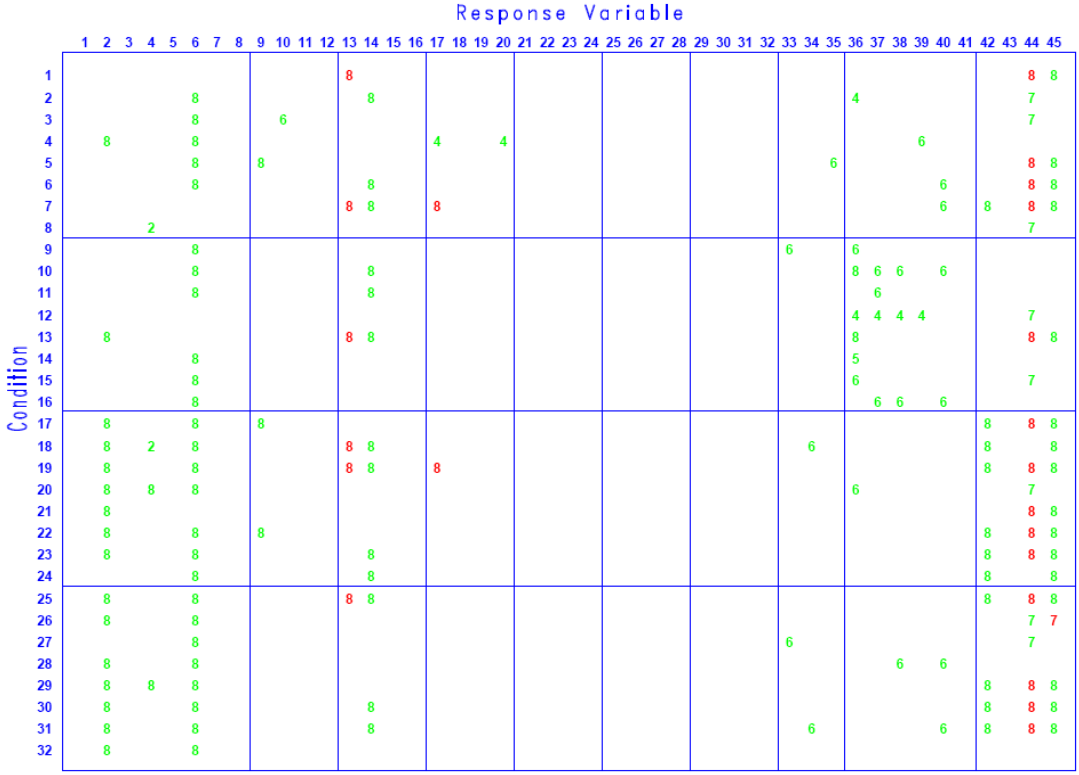

Figure 7-21. Condition-Response Summary for Time Period Two – 30% Filter Applied – plot displays for each Factor Combination (row) vs. Response Variable (column) the Identifier of the Algorithm (from Table 7-1) manifesting a Statistically Significant Outlier (green high and red low)

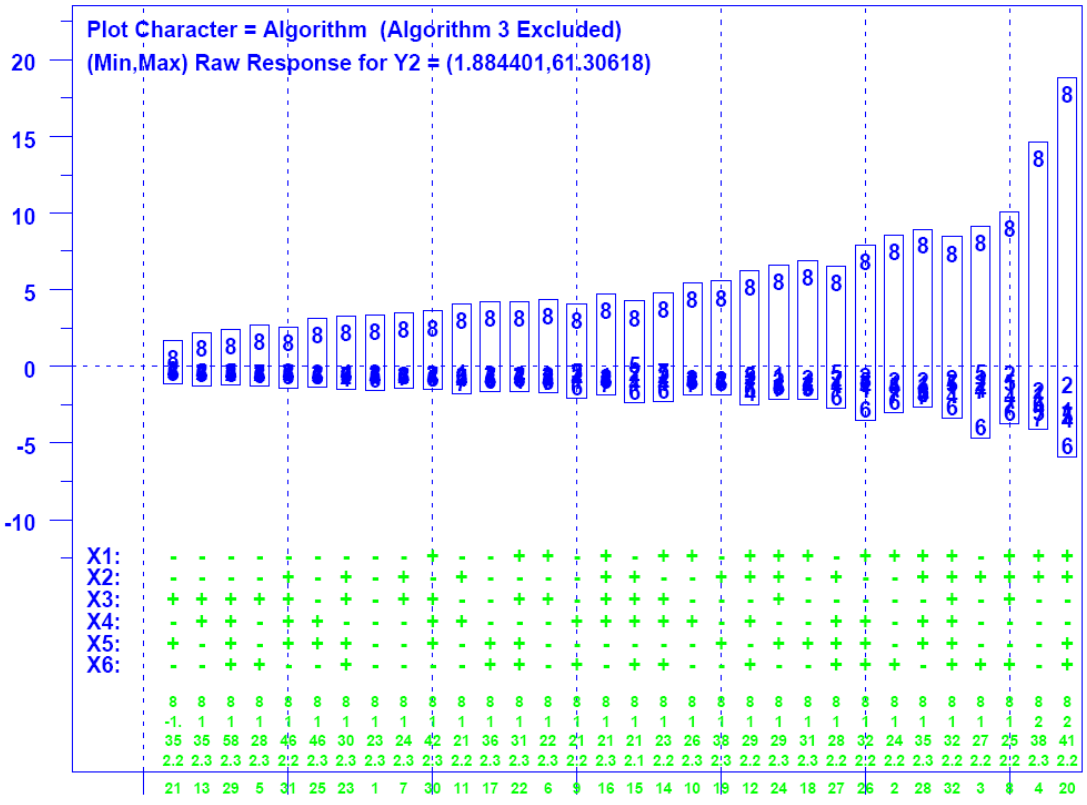

Figure 7-22. Detailed Analysis for Congestion Window Increase Rate (increases per 200 ms) in Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

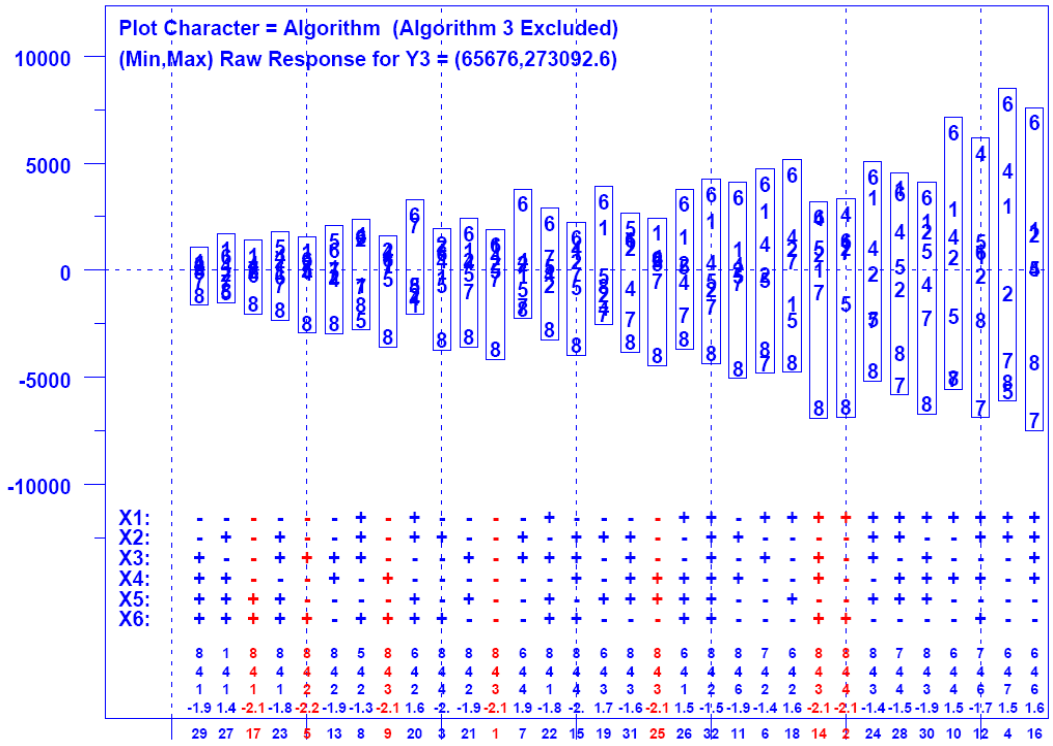

Figure 7-23. Detailed Analysis for Packet Output Rate (packets per 200 ms) in Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

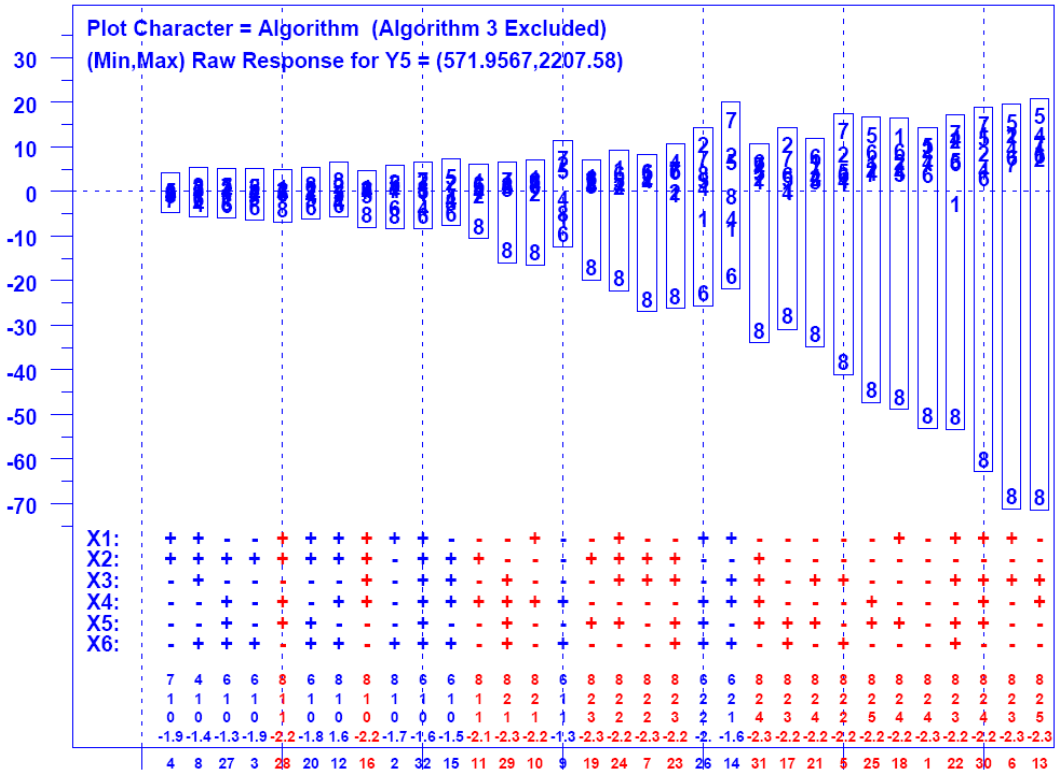

Figure 7-24. Detailed Analysis for Flow Completion Rate (flows per 200 ms) in Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

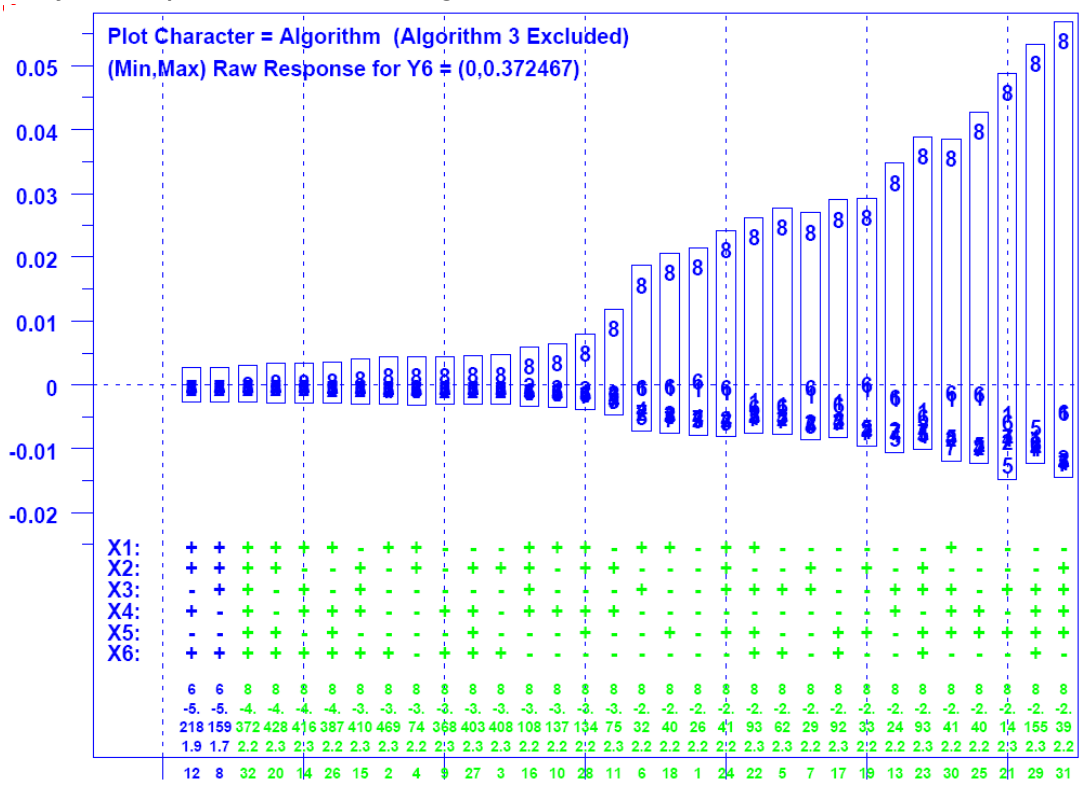

Figure 7-25. Detailed Analysis for Retransmission Rate (proportion of packets retransmitted) in Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

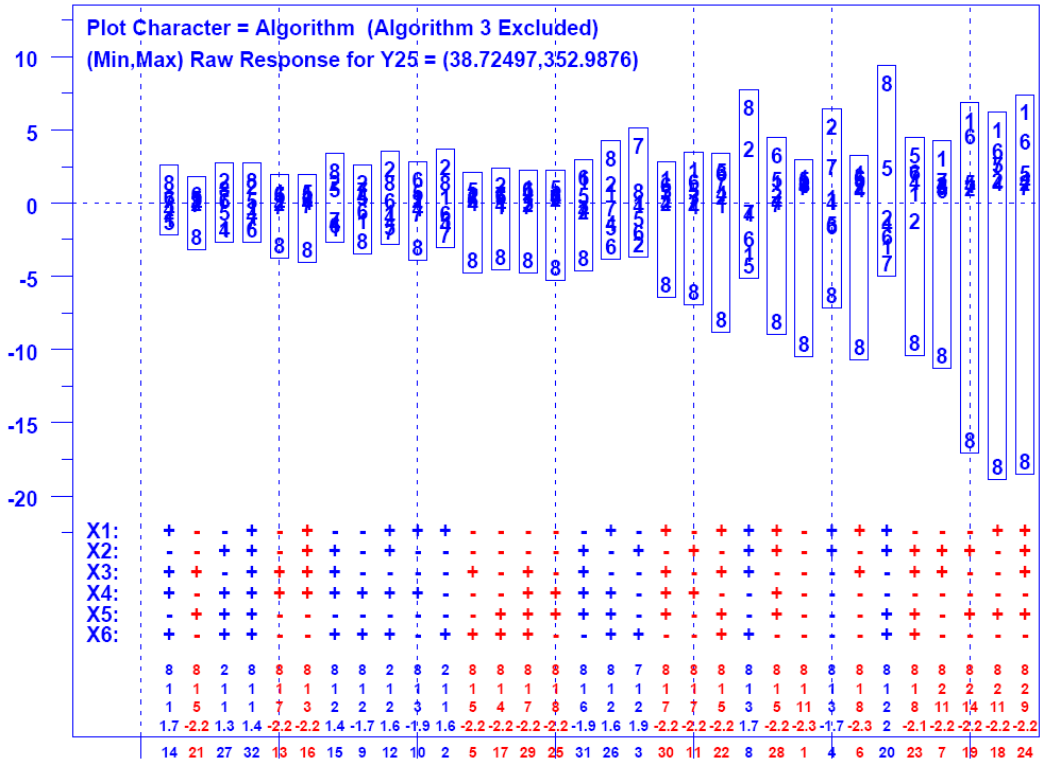

Figure 7-26. Detailed Analysis for Average Goodput (packets per second) on FN Flows in Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

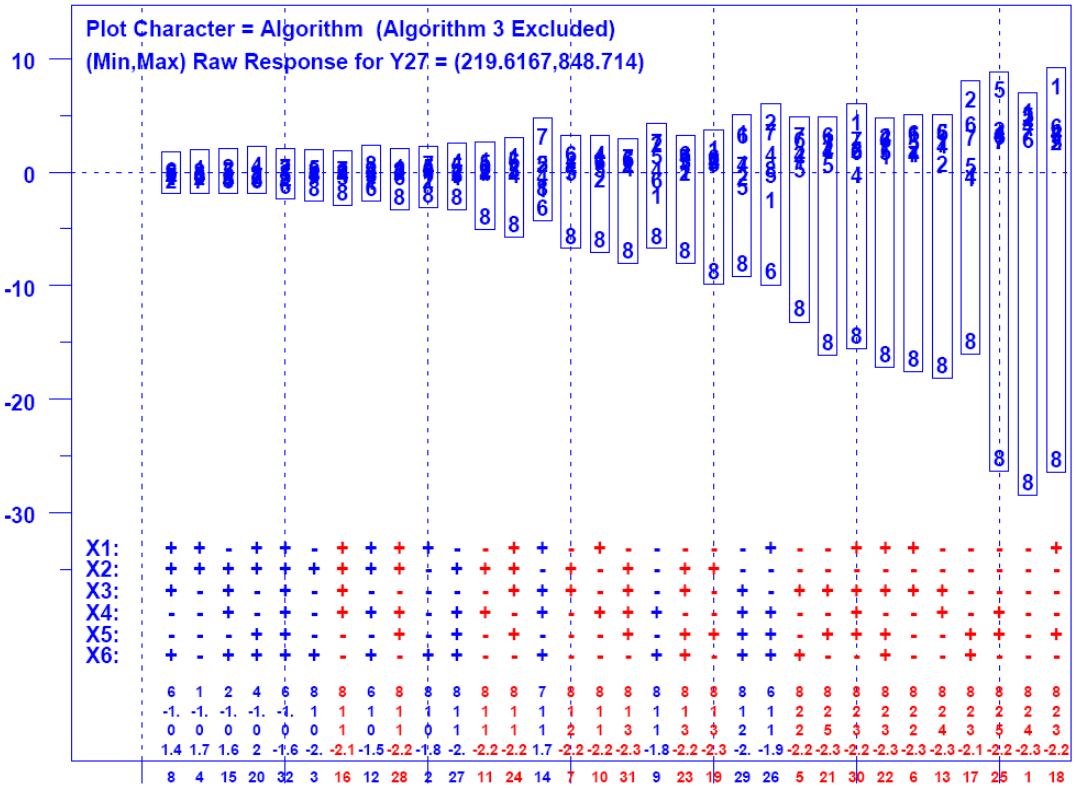

Figure 7-27. Detailed Analysis for Average FN Flow Completion Rate (flows per 200 ms) during Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

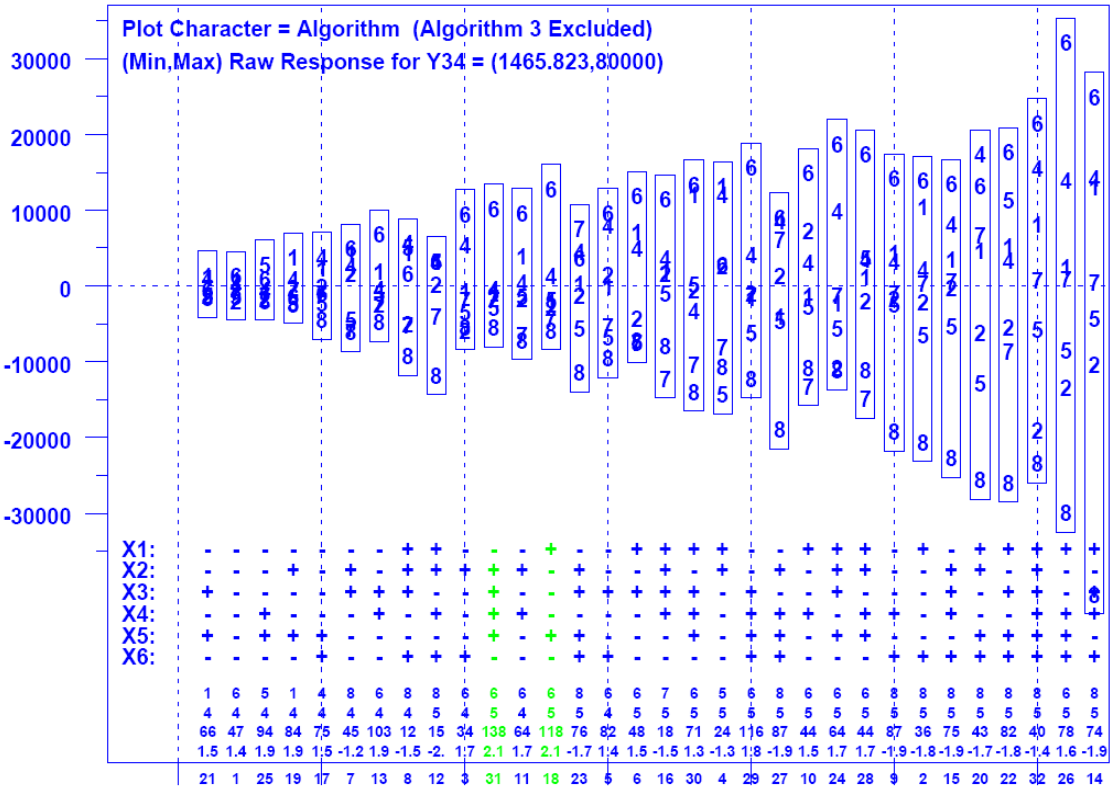

Figure 7-28. Detailed Analysis for Average Goodput (packets per second) on Long-lived Flow L2 in Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

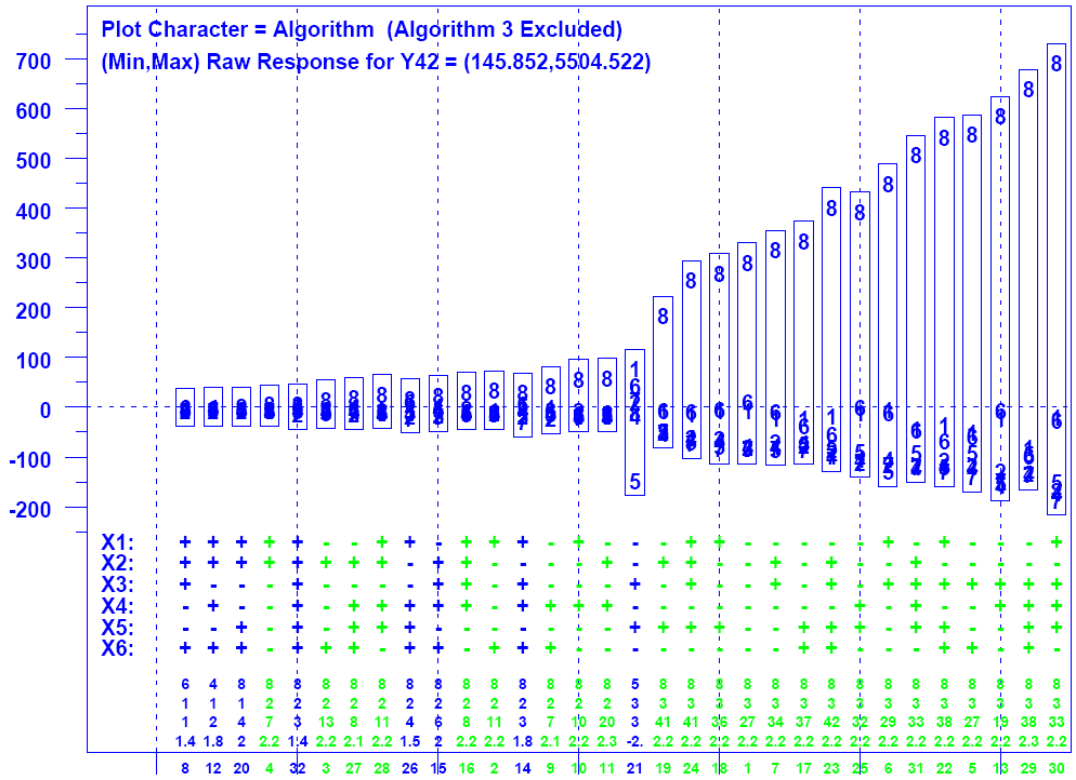

Figure 7-29. Detailed Analysis for Average Number of Connecting Flows during Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

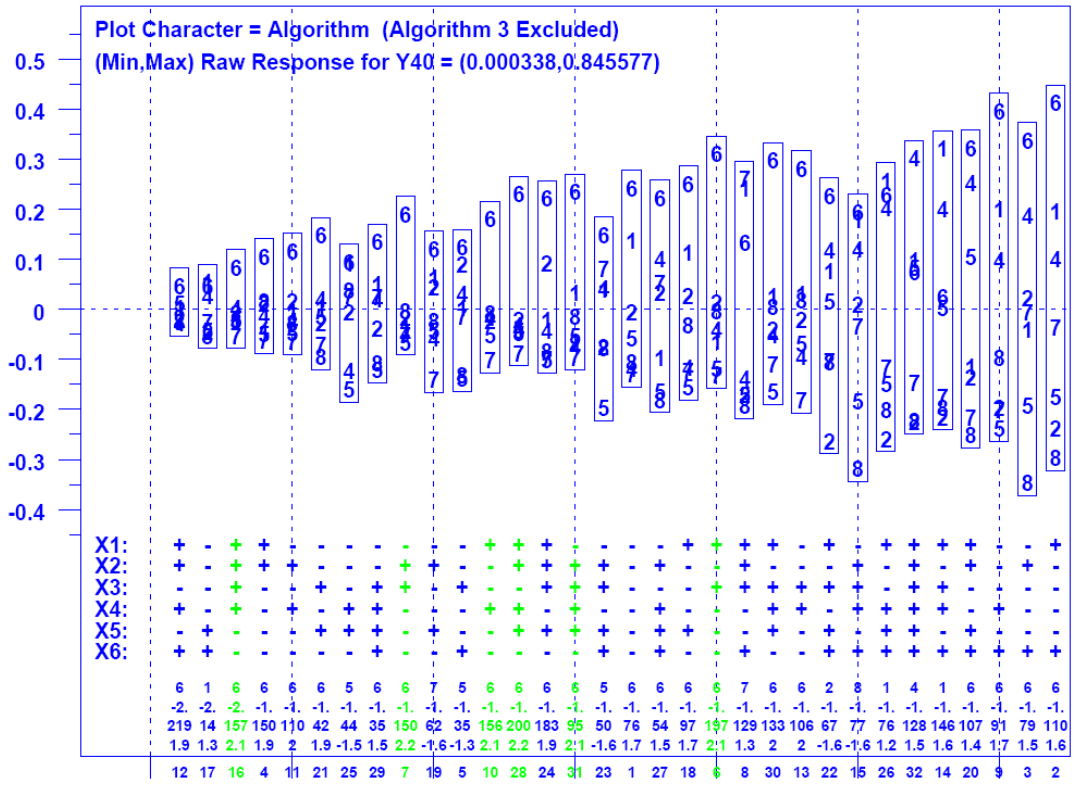

Figure 7-30. Detailed Analysis for Buffer Utilization (percentage of buffers occupied) at Router I0a during Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

7.4.3 Time Period Three (TP3)

During TP3 (200 ms intervals 6000 to 7500) no new jumbo file transfers are initiated on DD flows. What remains is for residual jumbo transfers to complete as the network transitions back toward normal Web traffic. The degree to which normal conditions can be restored depends upon how many jumbo transfers were created during TP2 and on how well a congestion control algorithm can recover from periods of intense congestion.

7.4.3.1 Cluster Analysis for TP3. Fig. 7-31 shows an annotated set of 32 dendrograms for TP3. Since the level of congestion stays relatively high, as residual jumbo file transfers drain from the network, algorithms 3 and 8 remain distinctive. The cluster analysis also finds algorithms 6 (Scalable TCP) and 7 (TCP Reno) to be somewhat distinctive under some conditions.

7.4.3.2 Condition-Response Summary for TP3. Fig. 7-32 gives the condition-response summary for TP3. Fig. 7-33 shows the same summary after applying a filter passing only statistically significant outliers for which the relative effect exceeds 30 %. These condition-response summaries confirm the findings from the cluster analysis: algorithms 3 and 8 stand out, along with algorithms 6 and 7 for selected responses and conditions. Figs. 7-32 and 7-33 also identify that algorithm 2 (CTCP) obtains a larger average congestion window size under selected conditions.

7.4.3.3 Analysis of Significant Responses for TP3. Guided by Figs. 7-32 and 7-33, we selected eight responses for detailed analysis, as reported in Figs. 7-34 to 7-41. Specifically, we show detailed analyses for rate of congestion window increases (y2), average packet output rate (y3), average congestion window size (y4), flow completion rate (y5), retransmission rate (y6), average goodput on the long-distance (L1), long-lived flow (y33), average buffer utilization on router C0a (y37) and average number of flows attempting to connect (y42).

7.4.3.4 Summary of Results for TP3. FAST-AT (algorithm 8) exhibits most of the same distinctive behaviors seen during TP1 and TP2. The results for TP3 also show that FAST-AT and TCP Reno (algorithm 7) both lag in recovering from the congestion of TP2. This shows up, for example, when examining detailed analyses for average packet output rate and for average goodput on the long- and moderate-distance, long-lived flows. The effect is more muted for the short-distance, long-lived flow. Scalable TCP (algorithm 6), BIC (algorithm 1) and HSTCP (algorithm 4) continue to use a higher proportion of buffers in the directly connected routers. In a previous experiment (Sec. 6.4.3), we found that CTCP exhibits a large congestion window size during easing congestion. This effect also appears in the current experiment but is somewhat attenuated.

7.4.4 Aggregated Responses (Totals)

Here we present analyses for the 28 responses collected over the entire 25-minute scenario. Recall that most of these responses are aggregated counts. Selected responses augment those counts with average values, specifically SYN rate on connected flows and goodput on completed flows. The analysis of totals examines the effects of behavioral differences when viewed over a longer period.

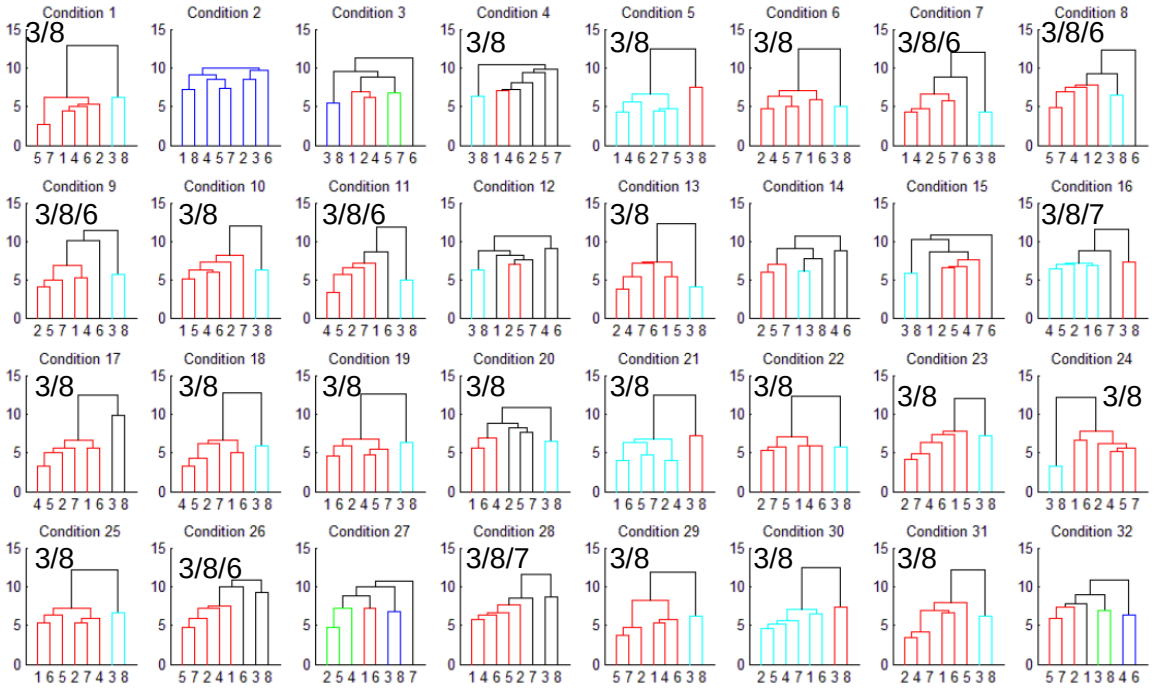

Figure 7-31. Cluster Analysis for Time Period Three – x axis on each sub-plot gives algorithm identifier from Table 7-1 and y axis shows distance in response space between pairs of algorithms and clusters as indicated – each sub-plot Annotated to Identify Distinctive Algorithms 3/8

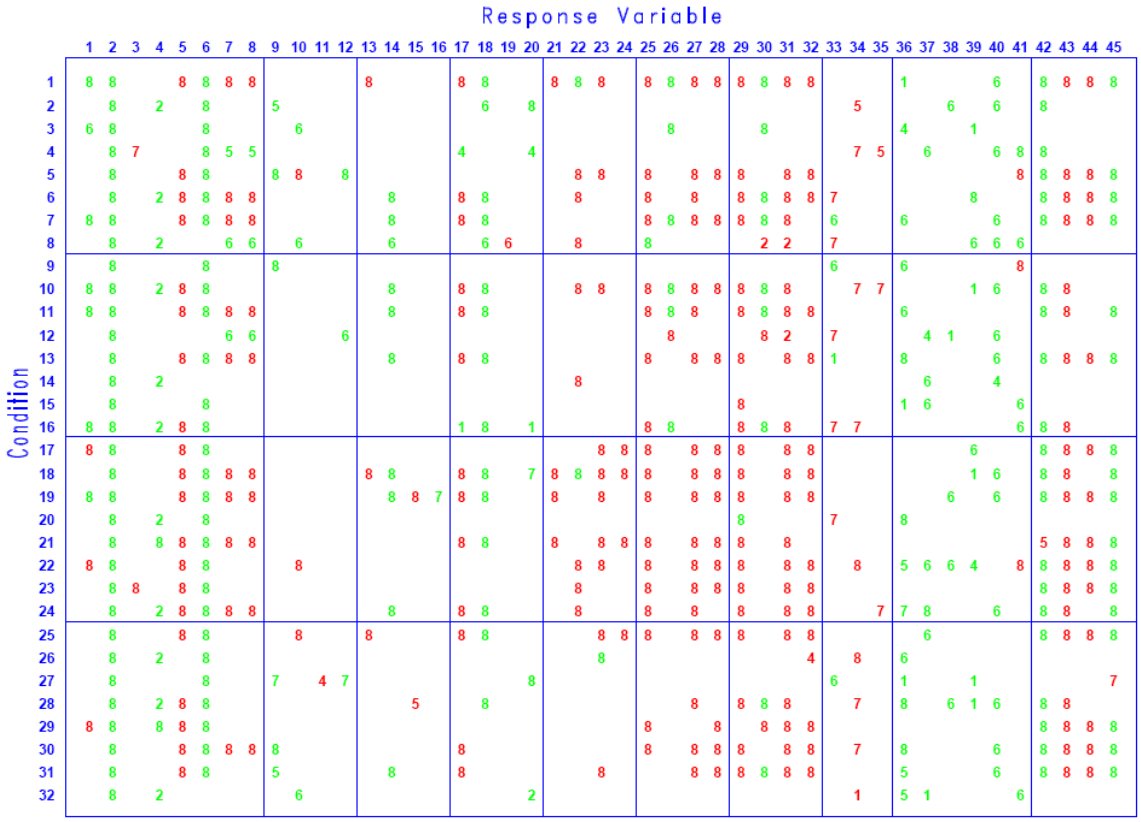

Figure 7-32. Condition-Response Summary for Time Period Three – plot displays for each Factor Combination (row) vs. Response Variable (column) the Identifier of the Algorithm (from Table 7-1) manifesting a Statistically Significant Outlier (green high and red low)

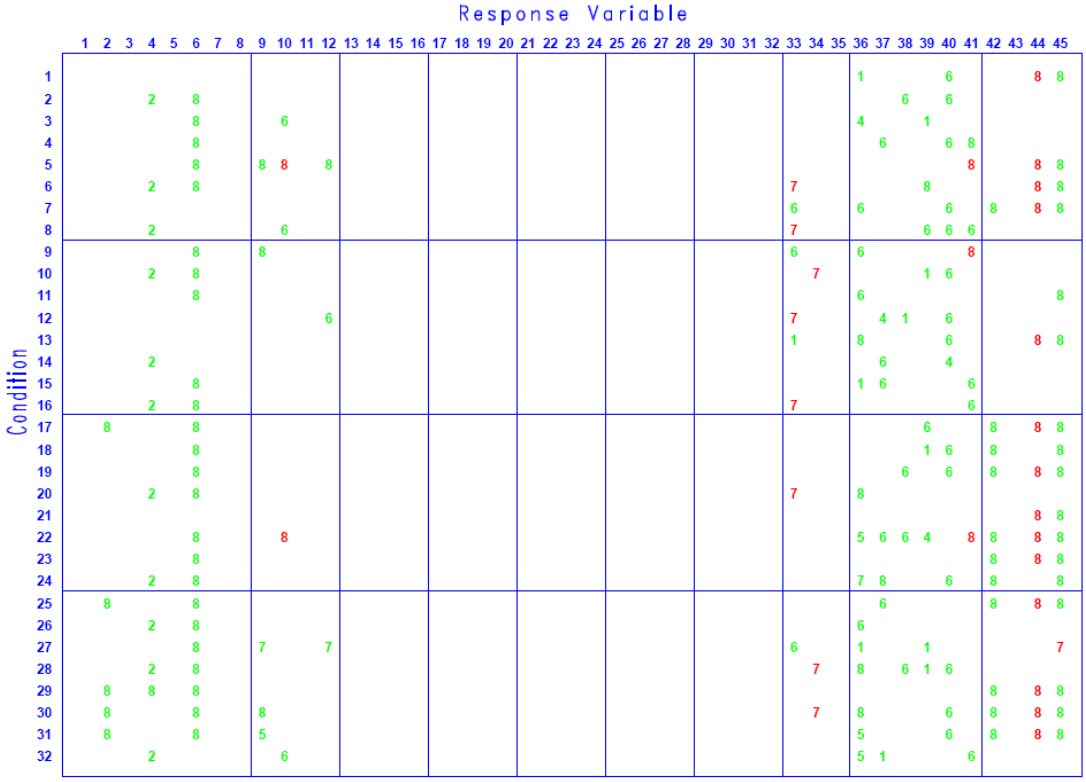

Figure 7-33. Condition-Response Summary for Time Period Three – 30% Filter Applied – plot displays for each Factor Combination (row) vs. Response Variable (column) the Identifier of the Algorithm (from Table 7-1) manifesting a Statistically Significant Outlier (green high and red low)

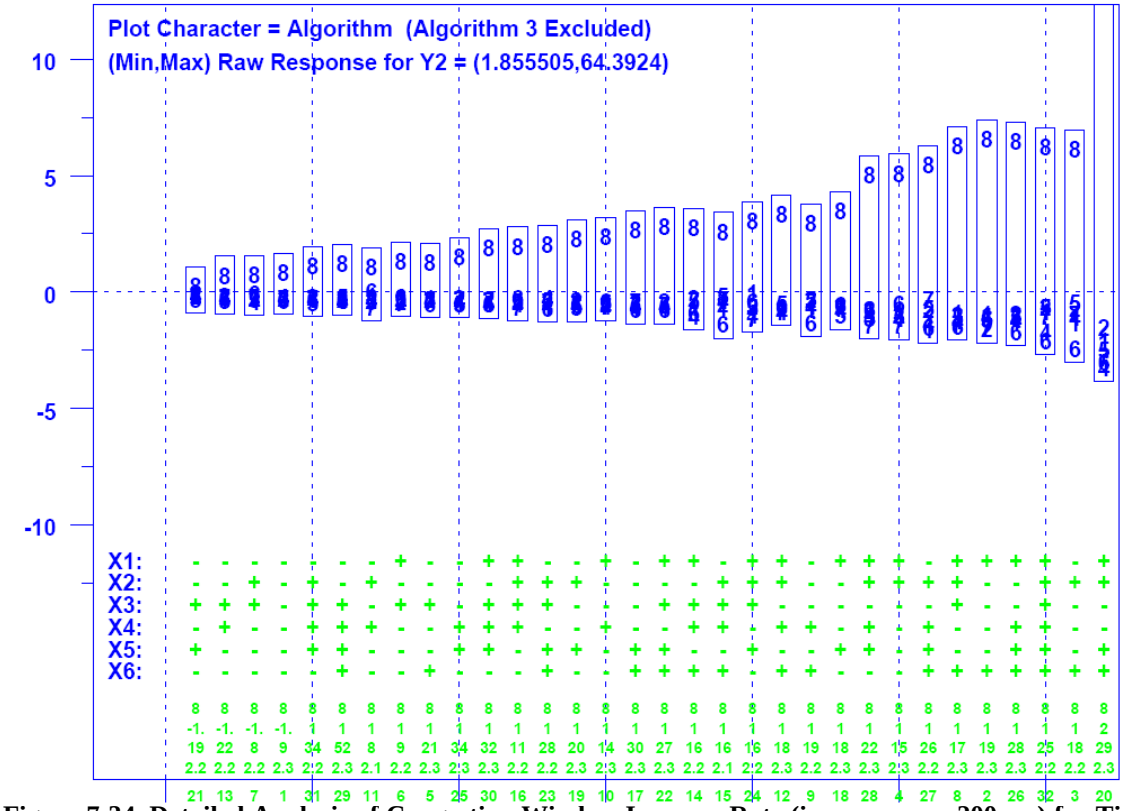

Figure 7-34. Detailed Analysis of Congestion Window Increase Rate (increases per 200 ms) for Time Period Three – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

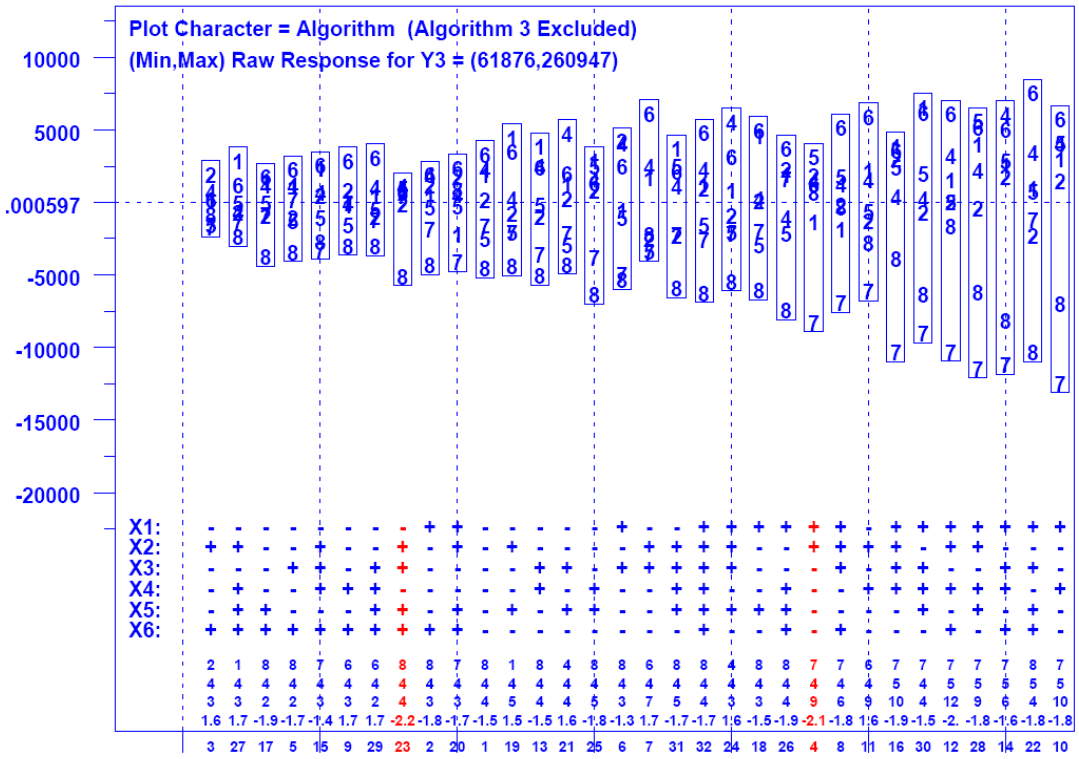

Figure 7-35. Detailed Analysis of Packet Output Rate (packets per 200 ms) for Time Period Three – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

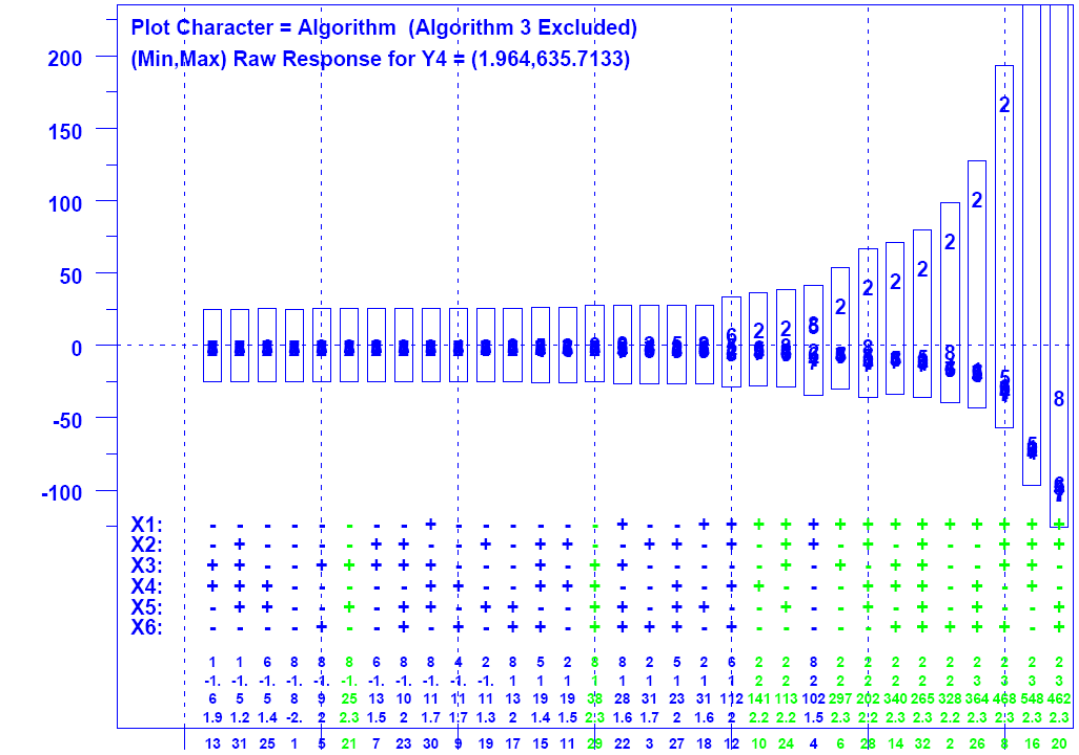

Figure 7-36. Detailed Analysis of Congestion Window Size (packets) for Time Period Three – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

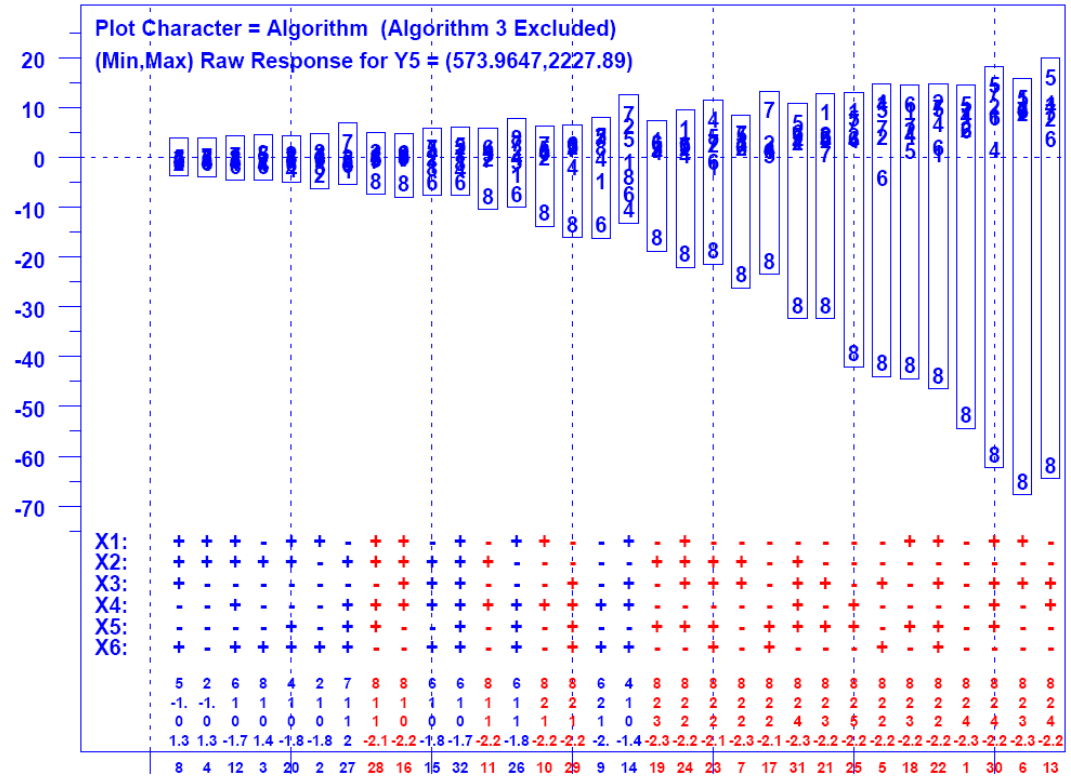

Figure 7-37. Detailed Analysis of Flow Completion Rate (flows per 200 ms) for Time Period Three – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

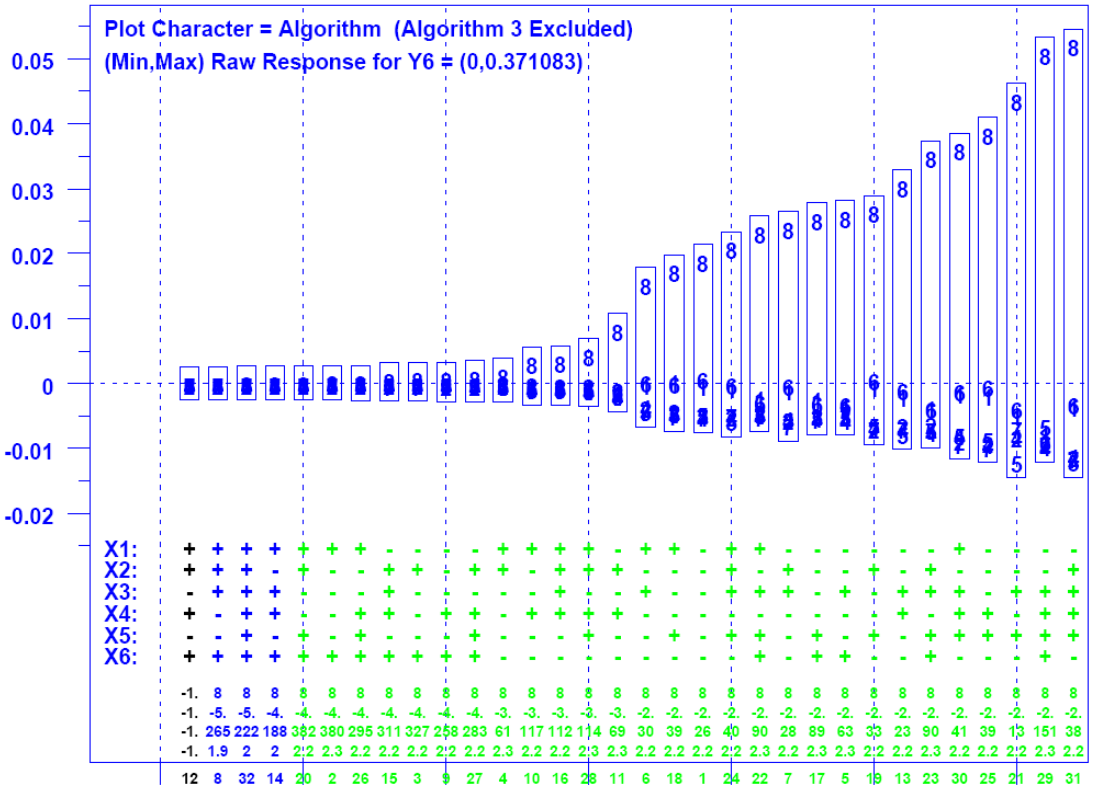

Figure 7-38. Detailed Analysis of Retransmission Rate (proportion of packets retransmitted) for Time Period Three – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

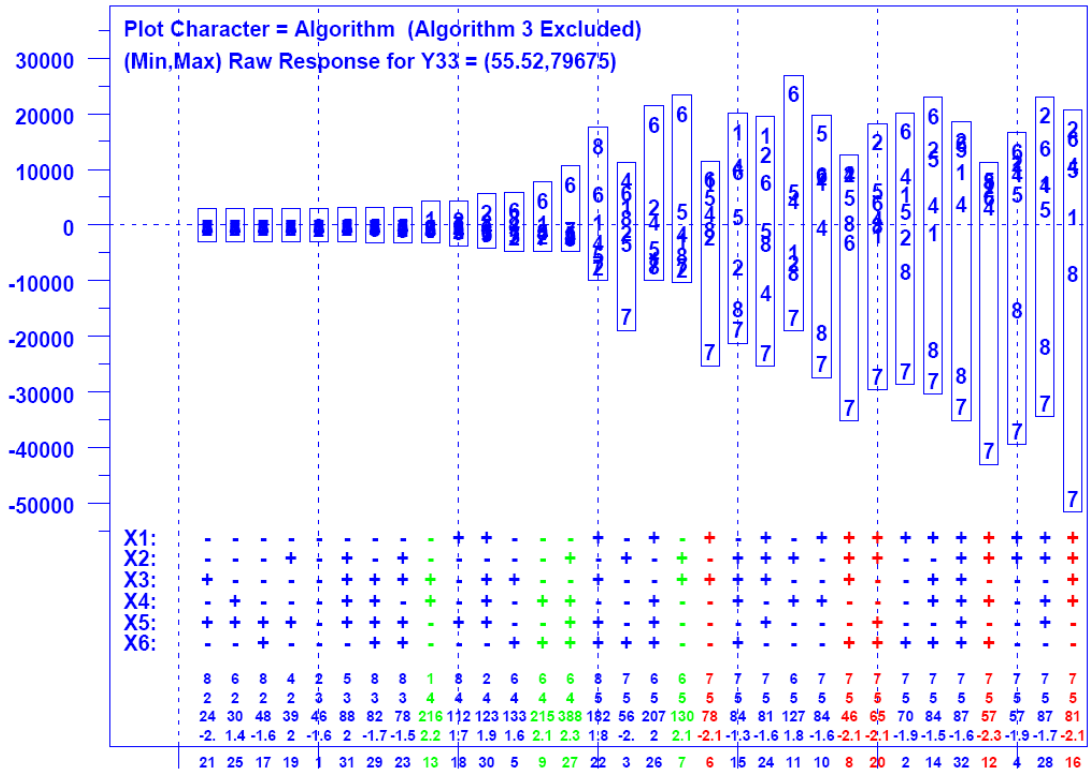

Figure 7-39. Detailed Analysis of Average Goodput (packets per second) on Long-lived Flow L1 in Time Period Three – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

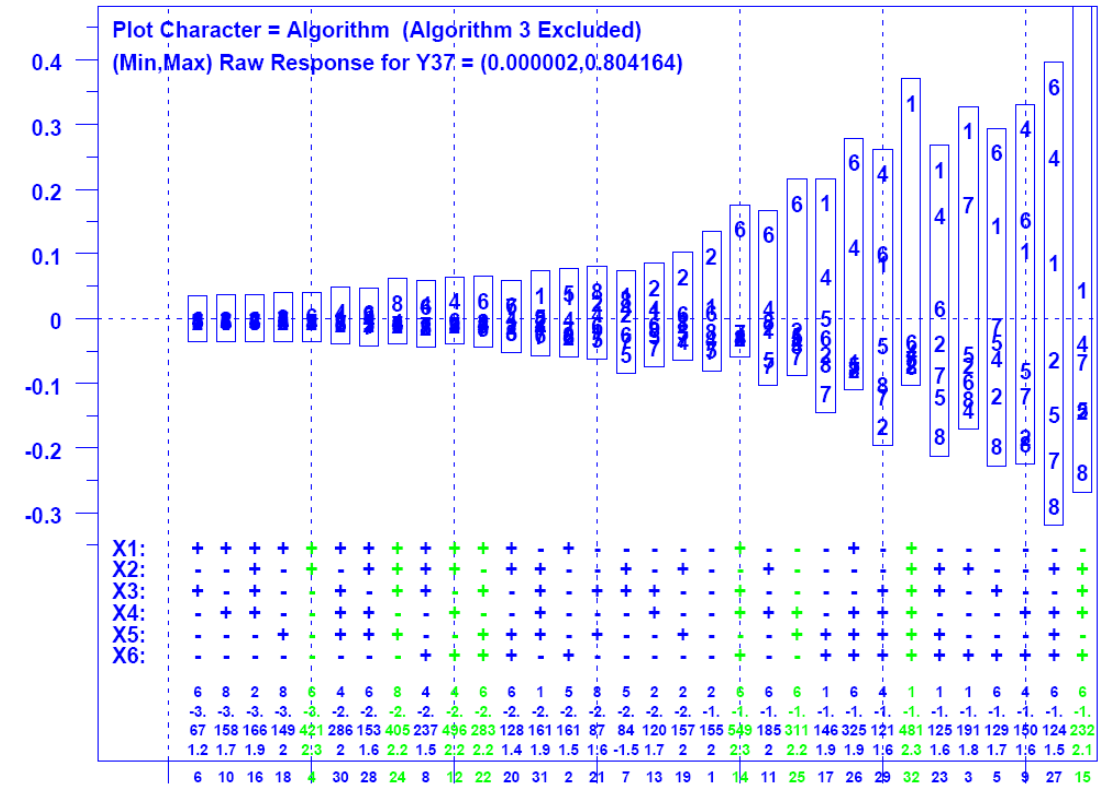

Figure 7-40. Detailed Analysis of Buffer Utilization (percentage of buffers occupied) in Router C0a during Time Period Three – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

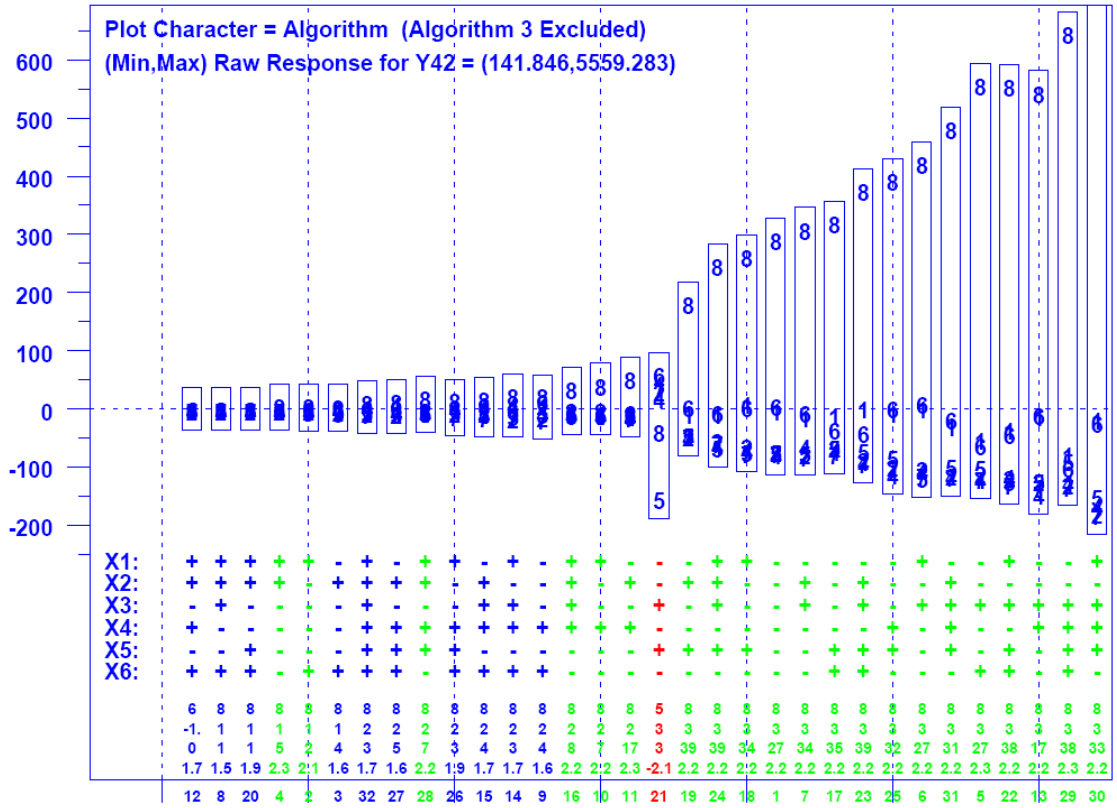

Figure 7-41. Detailed Analysis of Average Number of Connecting Flows during Time Period Three – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

7.4.4.1 Cluster Analysis for Totals. Fig. 7-42 shows a set of 32 annotated dendrograms with clustering based on the 28 aggregate responses. Similar to the cluster analyses for the three time periods, algorithms 3 and 8 appear distinctive in many (23) conditions. Also, as was true for TP2 and TP3, algorithm 6 stands out under condition 26. In addition, we note that cluster analyses for each of the time periods and the totals tend to suggest an affinity among some of the algorithms. For example, under selected conditions, algorithms 1, 4 and 6 (BIC, HSTCP and Scalable TCP) tend to cluster together and algorithms 2, 5 and sometimes 7 (CTCP, HTCP and TCP Reno) tend to cluster together.

7.4.4.2 Condition-Response Summary for Totals. Fig. 7-43, which gives the condition-response summary for the aggregate responses, proves quite revealing. First, standard TCP Reno (algorithm 7) pushes the fewest packets through the network. This occurs because TCP Reno increases its transmission rate slowly in congestion avoidance after leaving initial slow start. Second, FAST-AT (algorithm 8) provides the highest average goodput for DD, DF and FF flows (explained in Sec. 6.1), which transit very fast and fast paths, respectively. On the other hand, under most conditions, FAST-AT provides lowest average goodput for DN, FN and NN flows, which transit typical paths. Under uncongested conditions (e.g., 2, 12, 20 and 32) FAST-AT tends to provide highest throughput for DN, FN and NN flows. FAST-AT also sends more SYN packets, which

means flows have a more difficult time making connections. As a result FAST-AT connects and completes fewer flows.

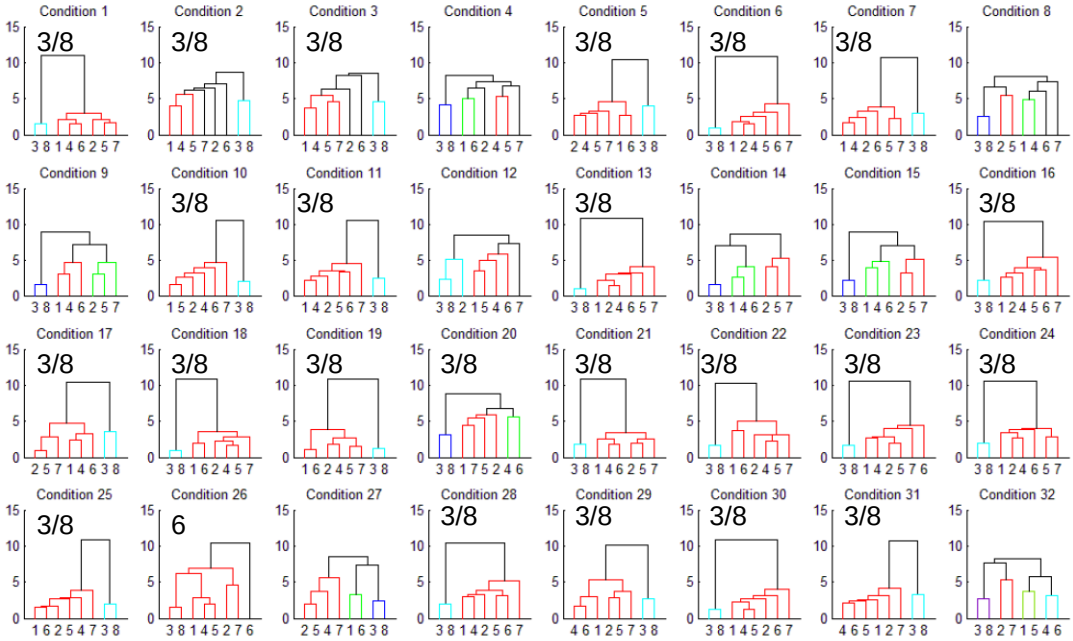

Figure 7-42. Cluster Analysis for Totals – x axis on each sub-plot gives algorithm identifier from Table 7-1 and y axis shows distance in response space between pairs of algorithms and clusters as indicated – each sub-plot Annotated to Identify Distinctive Algorithms 3/8

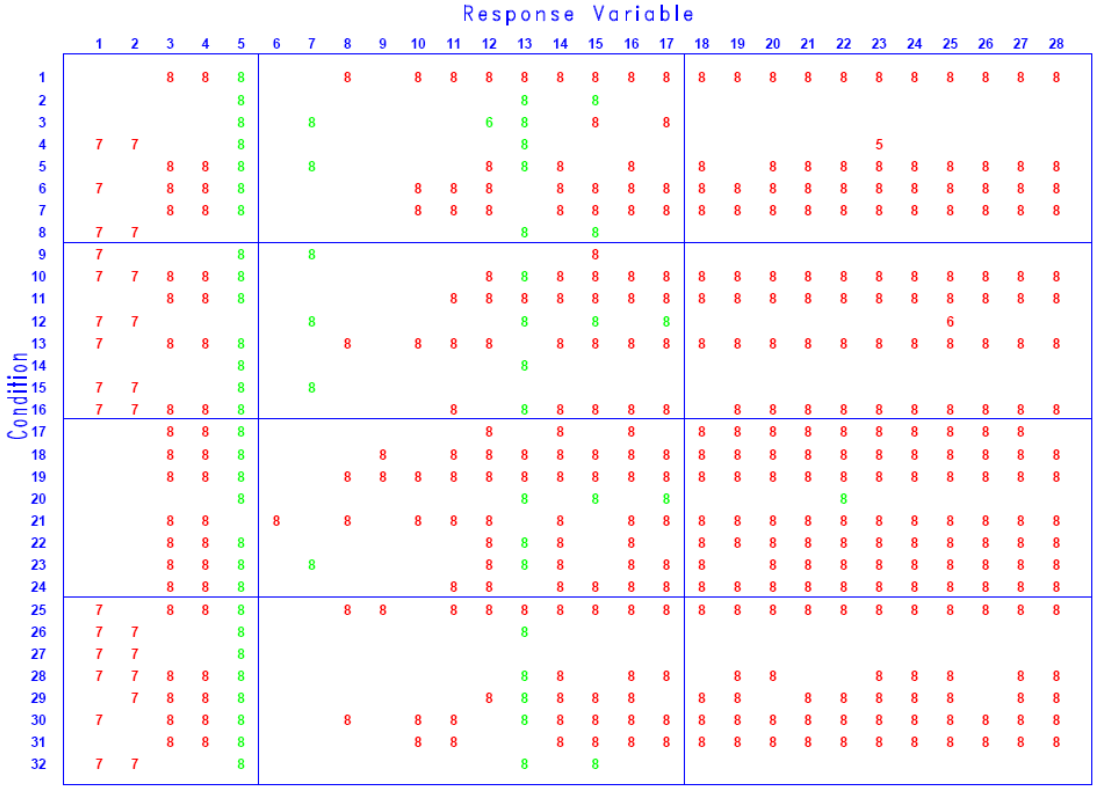

Figure 7-43. Condition-Response Summary for Totals – plot displays for each Factor Combination (row) vs. Response Variable (column) the Identifier of the Algorithm (from Table 7-1) manifesting a Statistically Significant Outlier (green high and red low)

7.4.4.3 Analysis of Significant Responses for Totals. Based on Fig. 7-43 we selected five responses for detailed analysis. Specifically, Figs. 7-44 to 7-48 provide results for aggregate packets input (T.y1) and output (T.y2), aggregate number of flows completed (T.y4), average number of SYNs sent per flow (T.y5) and average goodput on completed DD flows (T.y7).

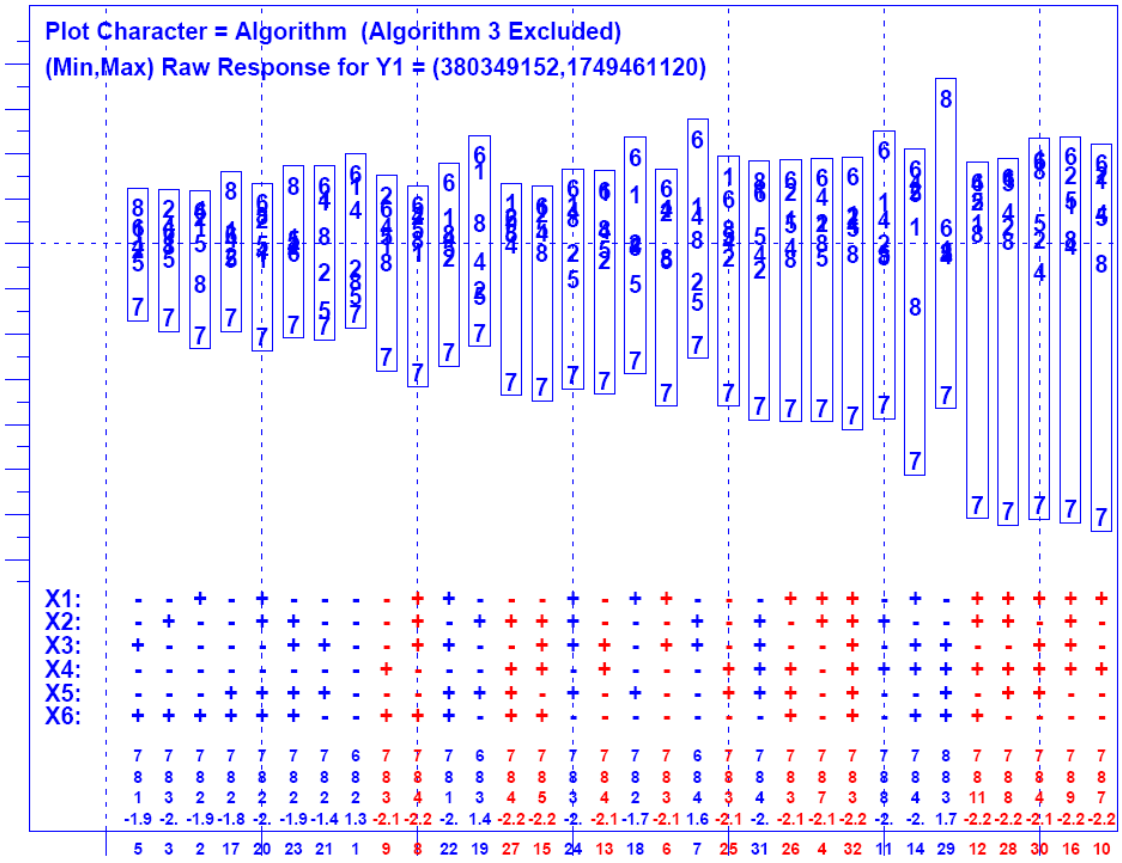

Figure 7-44. Detailed Analysis for Number of Packets Input during 25-minute Scenario – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

7.4.4.4 Summary of Results for Totals. Under most conditions investigated in this experiment, FAST-AT connects and completes on the order of 10^5 to 10^6 fewer flows during the 25-minute scenario. For flows that complete over fast and very fast paths, FAST-AT provides higher average goodput than the other algorithms. In some cases, this could be attributed to the fact that FAST-AT has fewer active flows sharing such paths because it is harder for flows to connect. In other cases, this can be attributed to the ability of FAST-AT to quickly achieve maximum transfer rate. TCP Reno injects fewer packets into the network (and thus outputs fewer packets). This can be attributed to the slow rate of increase in the congestion window during TCP Reno congestion avoidance. The results also show another a change over the previous experiment, where FAST input more packets into the network. In this experiment we excluded FAST data from the analyses, which then revealed that FAST-AT does not input as many packets per second into the network as FAST. This appears attributable to the ability of FAST-AT to lower the α parameter under congestion, and to increase α only slowly as congestion eases.

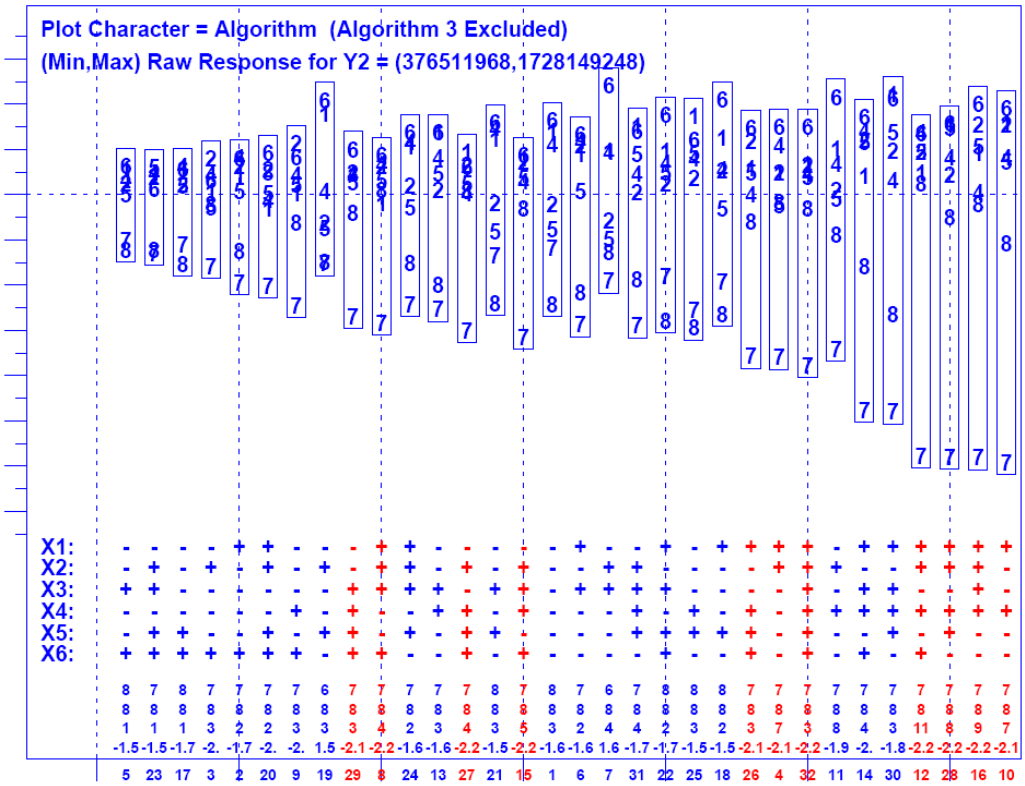

Figure 7-45. Detailed Analysis for Number of Packets Output during 25-minute Scenario – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

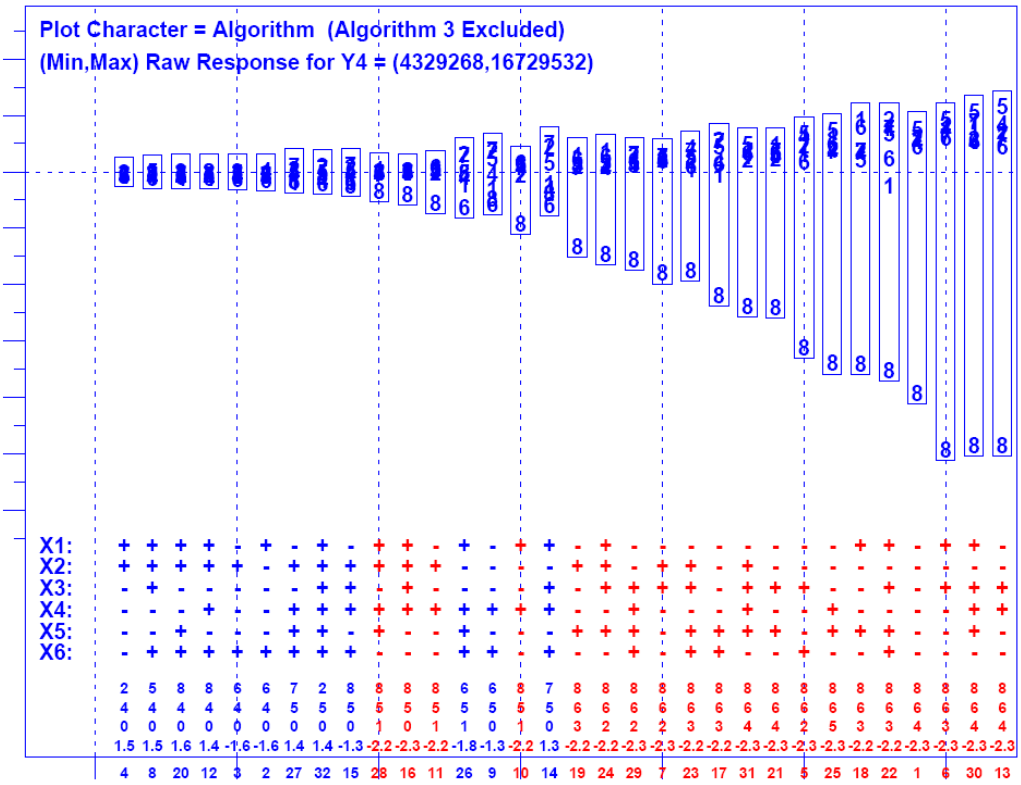

Figure 7-46. Detailed Analysis for Number of Flows Completed over 25-minute Scenario – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

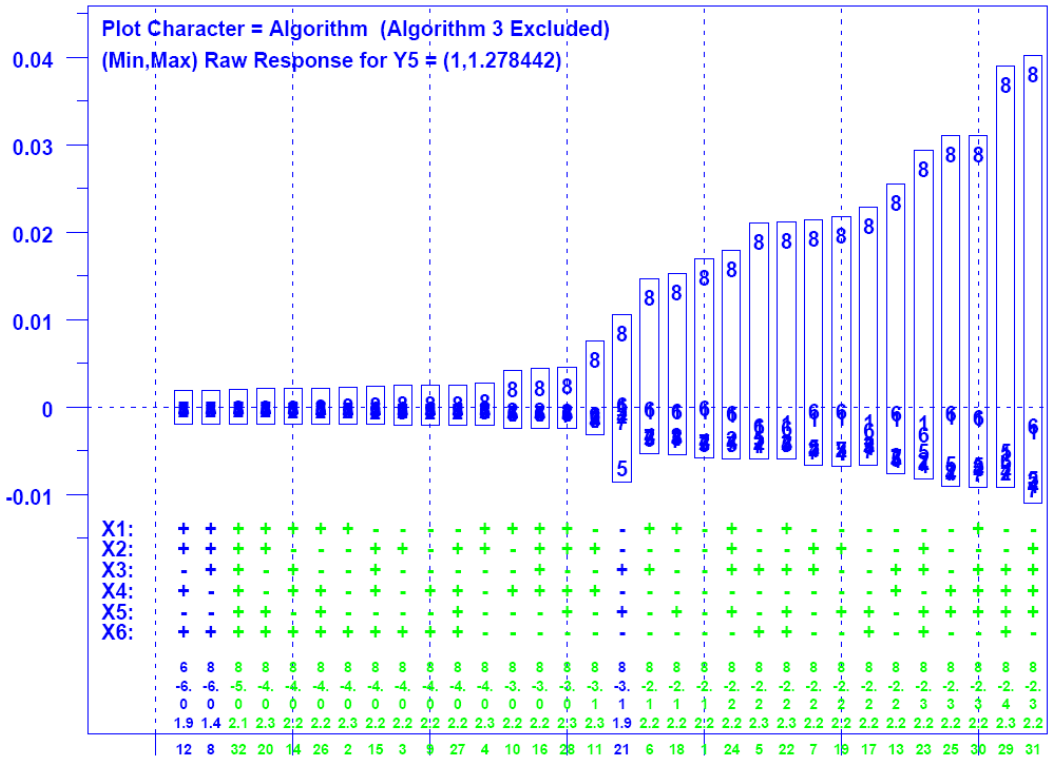

Figure 7-47. Detailed Analysis for Average SYN Rate (number of SYNs sent per flow) for Connecting Flows over 25-minute Scenario – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

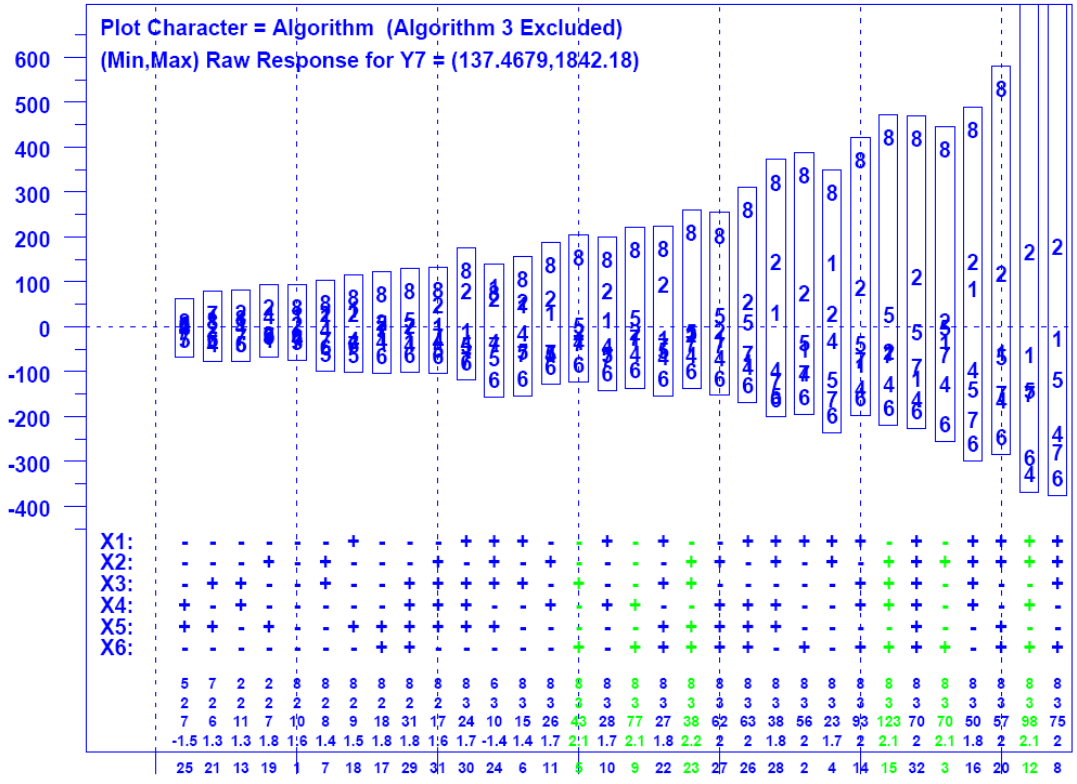

Figure 7-48. Detailed Analysis for Average Goodput (packets per second) on Completed DD Flows over 25-minute Scenario – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

7.5 Findings

We draw four main findings from our experimental results. First, all congestion control algorithms (except FAST and FAST-AT, algorithms 3 and 8, respectively) provided comparable network-wide behavior and user experience with respect to Web downloads and small file transfers over most path classes. This confirms similar findings from our previous experiment (discussed in Chapter 6) that adopted a large initial slow-start threshold. Modest differences (usually $\leq$ 20 %) do appear with respect to throughputs on DD flows. We identify and discuss these differences below in Sec. 7.5.1. Second, as discussed below in Sec. 7.5.2, when deployed network wide, alternate congestion control algorithms 3 (FAST) and 8 (FAST-AT) can produce macroscopic changes in network behavior at congested places in the topology and during congested periods. This confirms findings from our previous experiment. Third, for long-lived flows, TCP provides significantly lower throughput during TP1. While other congestion control algorithms perform better than TCP during TP1, we did detect differences among the algorithms in time taken to achieve maximum transfer rate for long-distance, long-lived flows. Differences also occur during TP3, when attempting to recover from the excessive congestion in TP2. During TP2, the congestion control algorithms provide comparable throughput across all long-lived flows. Below in Sec. 7.5.3 we discuss findings relating to long-lived flows. Fourth, under certain conditions, CTCP (algorithm 2) can drive congestion window size to substantially higher values than the other congestion control algorithms we simulated. As with our previous experiment, this behavior arises during TP3. However, as discussed below in Sec. 7.5.4, in this experiment the CTCP congestion window reaches a much lower maximum size than in our previous experiment. Finally, in Sec. 7.5.5, we identify some tendencies that were apparent but that did not achieve statistical significance.

7.5.1 Finding #1

Setting aside FAST and FAST-AT, for the experiment scenario and conditions examined in this chapter, the alternate congestion control algorithms exhibited indistinguishable macroscopic behavior and modest differences in user experience. This suggests that there was no overall advantage to be gained in switching the entire network to a particular alternate congestion avoidance scheme, nor was there any overall disadvantage in switching. (Remember we are excluding FAST and FAST-AT from this finding.) Selected users could experience somewhat higher throughputs when using alternate congestion control algorithms to complete file transfers with a size exceeding the initial slow-start threshold, but no widespread improvement in user experience should be expected.

The reasons underlying this finding are similar to the reasons identified in Sec. 6.5.1. Slow-start procedures are unaffected by alternate congestion control mechanisms, which define replacements only for the TCP congestion avoidance phase. No matter what congestion control mechanism is used, a flow commences operating in initial slow-start and switches to congestion avoidance only after a packet loss or once the initial slow-start threshold is reached. Aside from FAST, FAST-AT and TCP Reno, the alternate congestion avoidance procedures specify an activation threshold. Below that threshold, a flow adopts standard TCP congestion avoidance procedures; above that threshold the flow adopts alternate congestion avoidance procedures.

Recall from Fig. 7-1 that we simulated 32 conditions covering a range of congestion patterns, which could be classified approximately into 15 uncongested and 17 congested conditions. Condition 8 created the least congestion, while condition 21 created the most congestion. Of course, even uncongested conditions include localized congestion arising from the onset of jumbo file transfers in TP2, as well as from hot spots appearing from time-to-time at particular access routers. For example, Fig. 7-2 plots the distribution of flow states under an uncongested condition (condition 3) for algorithm 7 (TCP Reno). Note that most of the 700 or so active flows (red) in TP1 (t=3000 to t=4500) operate in initial slow start (green). This means that these active flows complete their file transfers without entering congestion avoidance. Only 20 or so (brown) flows enter congestion avoidance. Since condition 3 is uncongested, most of these 20 flows have likely entered congestion avoidance because they are larger than 100 packets, which is the initial slow-start threshold. This means that fewer than 3 % of active flows achieve any advantage from using alternate congestion avoidance procedures. This pattern also holds in TP3, after about t=6500, when the congestion (induced in TP2) abates. During the congested period (TP2) DD flows build up, as jumbo file transfers commence, and these flows contend for bandwidth, which leads to packet losses and thus to more flows operating under normal congestion control. Under such conditions, alternate congestion control algorithms do not offer any advantage. A similar story appears for congested conditions that extend over all three time periods.

Fig. 7-3 plots the distribution of flow states for TCP Reno (algorithm 7) under a congested condition (condition 5). About 60 % of the nearly 8500 active flows (red) operate in normal congestion control mode (brown), while the remaining 40 % operate in initial slow start (green). Under such congested conditions, alternate congestion avoidance procedures are not much activated. Most flows in a heavily congested network, or in heavily congested portions of a network, will be sharing paths with many other flows. For this reason, one should expect most flows to be operating within normal congestion control mode; these flows cannot achieve a large enough congestion window size (or avoid losses for long enough) to activate alternate congestion avoidance procedures. On the other hand, flows transiting very fast (DD) paths may be able to benefit from alternate congestion control procedures during periods with little congestion.

Table 7-10 reports the average, minimum and maximum per flow goodputs on DD flows (excluding long-lived flows), averaged over all conditions for each time period, for each of the eight congestion control algorithms. The table shows only modest differences in average goodput among most of the algorithms for TP1, though the average goodput lags somewhat for Scalable TCP. Note also that TCP Reno provides relatively high average goodput. Also notice that the average maximum goodput does not differ much in TP1 for most congestion control algorithms, though HSTCP seems to stand out somewhat on this metric. The average minimum goodputs are lower for BIC, HSTCP and Scalable TCP, which suggests that some flows are treated unfairly under these algorithms. On the other hand, TCP Reno provides the highest average minimum throughput, which suggests that these flows receive fairly equal treatment. During TP1 and TP2, the average, minimum and maximum goodputs provided by FAST and FAST-AT appear to be quite a bit higher than for the other algorithms, but this is due to the fact that fewer flows are actively transmitting under FAST and FAST-AT. As discussed previously, FAST flows have a more difficult time making connections. (This trait also

holds for FAST-AT.) The average goodputs among all the algorithms are relatively close in TP3. In general, for the conditions simulated, the alternate congestion control algorithms do not appear to provide a significant advantage over TCP Reno on DD flows with typical Web traffic. We consider the case of long-lived DD flows below in Sec. 7.5.3.

Table 7-10. Goodputs (pps) on DD Flows Averaged over all 32 Conditions for Each Time Period

		BIC	CTCP	FAST	HSTCP	HTCP	Scalable	TCP	FAST-AT
TP1	Average	174.89	189.85	207.05	172.80	183.82	164.79	194.20	201.12
	Minimum	49.48	89.62	89.25	49.97	70.90	39.05	100.63	89.25
	Maximum	378.29	345.40	377.81	431.02	374.62	363.71	357.84	356.34
TP2	Average	1545.97	1651.73	2132.79	1280.32	1433.19	1163.48	1327.92	2197.17
	Minimum	470.14	498.14	712.72	420.35	523.24	280.87	500.09	667.58
	Maximum	6173.99	5250.65	7380.92	4992.77	4781.59	4868.55	4406.78	7455.79
TP3	Average	2105.58	2077.44	2205.36	2309.45	2222.60	1998.66	2262.30	2407.78
	Minimum	322.88	254.83	372.24	262.99	289.84	317.60	291.07	400.14
	Maximum	4827.73	5627.25	5456.05	4726.75	5774.65	4684.17	5724.38	5963.79

7.5.2 Finding #2

In our previous experiment, we found (Sec. 6.5.2) that the FAST congestion control algorithm, when deployed throughout the network, produced macroscopic behavior changes at congested places in the topology and during congested periods. Further, we found that these changes could present Web-browsing users with lower average goodputs and longer connection times. The influence of these effects increased with increasing congestion. These findings suggested that deploying FAST on a wide scale could incur significant risk. Our current experiment reveals that FAST-AT (algorithm 8) shares these same undesirable traits, and for the same reasons.

FAST-AT exhibits rapid oscillations in congestion window size when a path has insufficient buffers to contain the packets that the algorithm attempts to maintain queued at a bottleneck. When a large number of flows simultaneously transit a network router, the overall effect can be to flood the router with many packets. When the number of flows is sufficient to overrun the available buffers in the router, FAST-AT flows exhibit an oscillatory behavior that can create additional congestion that causes flows to remain in oscillation for an extended time. For example, Fig. 7-49 shows the change in the congestion window for long-lived FAST-AT flow L2 during 500 measurement intervals (100 s) within TP2 under condition 21. For comparison, Fig. 7-50 gives the behavior of standard TCP Reno under the same circumstances. (The reader may wish to compare these figures with Figs. 6-60 through 6-66 from the previous experiment.)

The rapid oscillatory behavior of FAST-AT results in numerous packet losses, which leads to a large rate of congestion window increases (as shown in Figs. 7-10, 7-22 and 7-34) and to a higher retransmission rate (as shown in Figs. 7-12, 7-25 and 7-38). The higher loss rate also causes a higher SYN rate (as shown in Fig. 7-47), which leads to a larger number of flows pending in a connecting state (as shown in Figs. 7-15, 7-29 and

7-41) because flows take longer to connect. Flows also take longer to complete because a larger number of packets must be retransmitted. This effect can be seen in Figs. 7-11, 7-14, 7-24, 7-27 and 7-37, which show that FAST-AT flows have a significantly lower completion rate. The net effect of a lower completion rate appears in Fig. 7-46, which shows that FAST-AT completes many fewer flows than other algorithms, excluding FAST, over a 25-minute period.

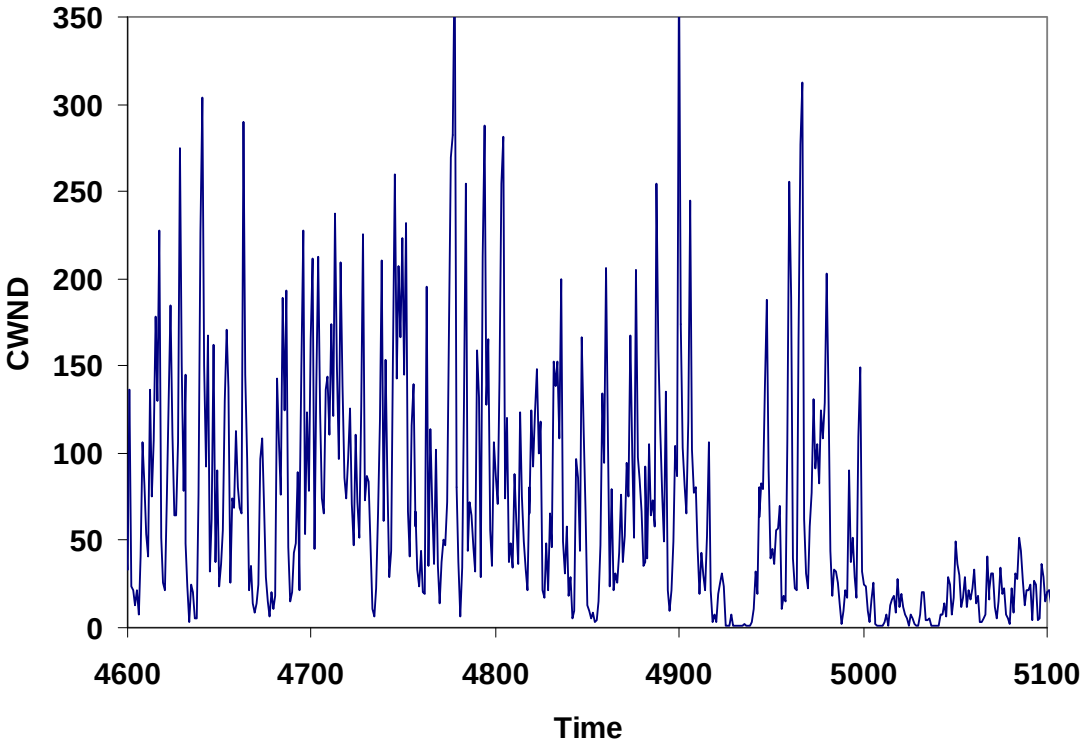

Figure 7-49. Change in Congestion Window (packets) under FAST-AT for Long-Lived Flow L2 during 500 Measurement Intervals (200 ms each) within TP2 under Condition 21

In summary, a large network with many simultaneously active flows can induce congestion at various times and locations within the topology. When congestion is sufficient to induce losses, flows using the FAST-AT algorithm can enter a rapid oscillatory behavior that exacerbates congestion. As a result, the network can exhibit a higher overall loss rate with consequent increase in retransmissions. Flows can take longer to connect and complete. The number of flows completed in such a network can be significantly reduced over long time spans. Should FAST-AT be deployed throughout a network, typical Web-browsing users could experience lower average goodput on flows transiting through congested areas. These findings also apply to FAST, which showed similar behavior in the previous experiment.

7.5.3 Finding #3

The design of most alternate congestion control algorithms is motivated by a desire to improve on standard TCP congestion avoidance procedures by providing higher goodput when transmitting large files over long-delay, high-speed network paths. To investigate the degree to which alternate congestion control algorithms achieve this aim, we included

three long-lived flows in several of our experiments. The long-lived flows transmit (infinitely large) files over very fast (DD) paths between sources and receivers capable of operating at 80 x 10^3 packets per second. In the previous experiment, we found little difference in goodput among congestion control algorithms (including TCP Reno) because all long-lived flows achieved maximum transfer rate (in TP1) while operating in initial-slow start and because all long-lived flows were influenced by heavy congestion in TP2. In the current experiment, we expected TCP Reno to perform less well on long-lived flows in TP1 because the initial slow-start threshold is low (100 packets). As in the previous experiment, we also expected TCP Reno to perform less well in TP3 during recovery from the heavy congestion in TP2. Further, we expected any differences in performance among the alternate congestion control algorithms to become apparent.

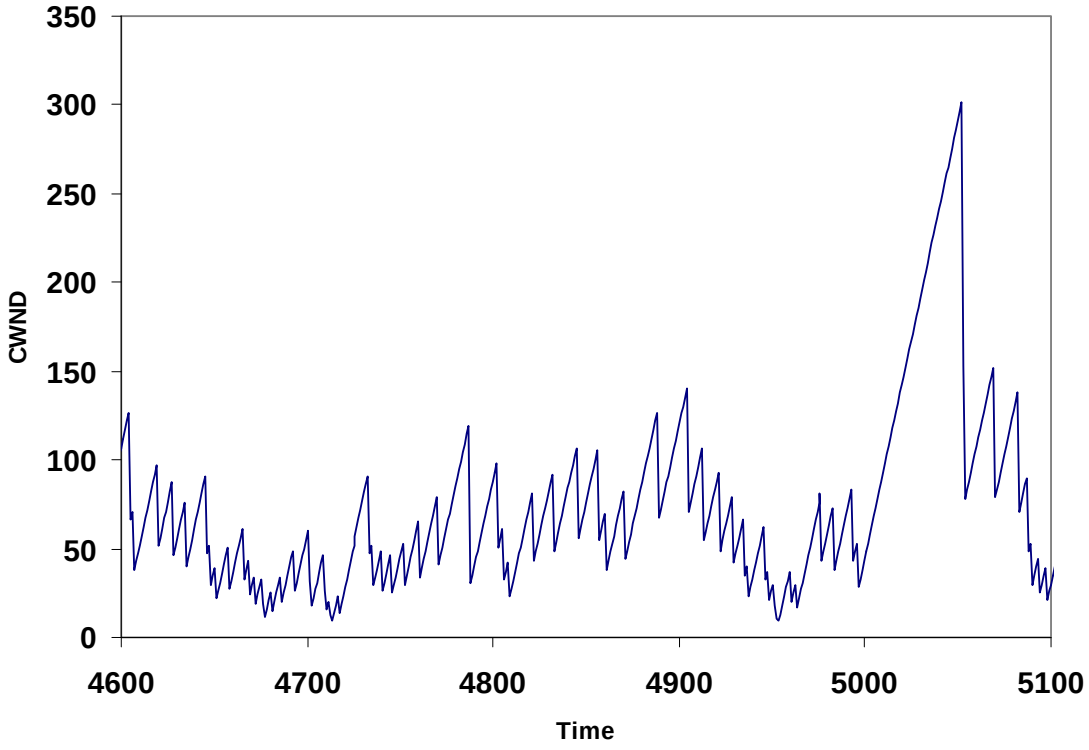

Figure 7-50. Change in Congestion Window (packets) under TCP Reno for Long-Lived Flow L2 during 500 Measurement Intervals (200 ms each) within TP2 under Condition 21

Table 7-11 reports the average, minimum and maximum goodputs (divided by 1000) provided by each congestion control algorithm for the long-distance, long-lived flow (**L1**) averaged over all 32 conditions for each of the three time periods. As expected, during TP1 the standard TCP Reno congestion control algorithm underperforms all of the alternate congestion control algorithms. This demonstrates that the alternate algorithms can provide higher goodputs than TCP when transmitting large files over long-delay, high-speed network paths, provided there is a low initial slow-start threshold. This effect is also evident for the moderate-distance (**L2**) and short-distance (**L3**) long-lived flows, as shown in Tables 7-12 and 7-13, but the effect diminishes significantly with decrease in propagation delay. The effect is also evident in Fig. 7-17. The inability of TCP

congestion control to quickly reach maximum transfer rate is also responsible for the fact that TCP Reno outputs fewer packets per measurement interval in TP1 (see Fig. 7-16). In fact, when considered across the entire 25-minute scenario, TCP Reno inputs significantly fewer packets under most conditions (see Fig. 7-44).

Table 7-11. Per Flow Goodputs (pps/1000) for Long-Lived Flow L1 Averaged over all 32 Conditions for Each Time Period

		BIC	CTCP	FAST	HSTCP	HTCP	Scalable	TCP	FAST-AT
TP1	Average	59.822	62.497	64.857	58.513	64.617	62.087	15.859	64.840
	Minimum	44.960	34.494	37.690	43.804	47.128	48.063	6.581	38.363
	Maximum	73.204	77.051	78.899	72.038	74.466	73.150	25.151	78.899
TP2	Average	8.232	5.830	2.880	9.939	6.934	13.775	3.318	2.745
	Minimum	0.604	0.260	0.240	0.508	0.494	0.660	0.320	0.191
	Maximum	51.007	35.010	23.003	61.699	59.662	53.940	18.459	26.766
TP3	Average	23.232	23.461	27.237	22.525	22.964	27.472	6.335	16.164
	Minimum	0.104	0.087	0.050	0.087	0.110	0.117	0.087	0.056
	Maximum	79.593	79.675	76.994	79.588	79.079	78.140	37.658	78.903

Table 7-12. Per Flow Goodputs (pps/1000) for Long-Lived Flow L2 Averaged over all 32 Conditions for Each Time Period

		BIC	CTCP	FAST	HSTCP	HTCP	Scalable	TCP	FAST-AT
TP1	Average	70.304	69.200	70.526	69.110	66.900	70.560	43.070	70.479
	Minimum	59.547	47.952	52.016	57.204	47.210	60.918	27.098	52.179
	Maximum	77.929	78.896	79.707	77.441	76.236	77.431	65.462	79.707
TP2	Average	32.860	26.996	19.188	34.727	26.716	40.243	26.732	17.222
	Minimum	2.981	1.706	1.850	3.189	1.706	2.394	2.444	1.466
	Maximum	80.000	76.982	64.945	79.986	79.984	79.979	72.793	65.099
TP3	Average	61.447	58.343	53.823	60.902	57.718	61.121	51.106	54.677
	Minimum	29.751	17.704	10.311	19.725	22.337	17.682	12.542	7.317
	Maximum	80.000	80.000	80.000	80.000	80.000	80.000	80.000	80.000

Table 7-13. Per Flow Goodputs (pps/1000) for Long-Lived Flow L3 Averaged over all 32 Conditions for Each Time Period

		BIC	CTCP	FAST	HSTCP	HTCP	Scalable	TCP	FAST-AT
TP1	Average	73.219	72.681	71.735	72.298	67.693	73.039	63.519	71.673
	Minimum	63.986	55.644	55.320	60.997	46.031	64.104	48.425	55.246
	Maximum	79.331	79.558	79.907	79.904	77.613	78.975	76.185	79.907
TP2	Average	29.485	25.822	18.281	30.809	22.005	35.495	25.882	16.238
	Minimum	2.030	1.925	1.672	1.632	1.063	1.152	1.424	1.150
	Maximum	79.985	76.669	74.230	79.672	79.181	79.997	79.685	73.810
TP3	Average	61.066	57.312	55.706	59.204	56.872	61.321	57.438	55.849
	Minimum	20.817	14.118	17.248	20.842	14.776	17.259	22.235	15.677
	Maximum	80.000	80.000	79.988	80.000	80.000	80.000	80.000	79.992

Considering TP3, we find that, on average, TCP Reno recovers significantly less well than the alternate congestion control algorithms for the long-distance (**L1**) and moderate-distance (**L2**) long-lived flows, but TCP Reno performs on a par with the alternate algorithms when recovering throughput on the short-distance (**L3**) long-lived flow. We also note that FAST-AT lags behind the other alternate congestion control algorithms when recovering on the long-distance (**L1**) long-lived flow. Further, FAST and FAST-AT tend to recover less well than other alternate congestion control algorithms on the moderate-distance (**L2**) and short-distance (**L3**) long-lived flows. In general, all congestion control algorithms provide similar goodputs across all time periods on the short-distance, long-lived flow, so differences in goodput arise only with sufficient propagation delay.

Considering TP2, Scalable TCP, BIC and HSTCP retain higher average goodputs on the long-distance (**L1**) and moderate-distance (**L2**) long-lived flows, while FAST and FAST-AT yield lower average goodputs. This tendency also appeared in the previous experiment, as discussed in Sec. 6.5.4.

In an effort to better understand similarities and differences among the congestion control algorithms, we next consider some of the detailed operations on long-lived flows, first under uncongested conditions and then under the most congested condition. We begin with the least congested condition (condition 8).

Table 7-14. Time (seconds) until Long-Lived Flows Reach Maximum Transfer Rate in TP1 for Condition 8

	Long-Lived Flow		
Algorithm	**L1**	**L2**	**L3**
BIC	**39.4**	**12.6**	**4.6**
CTCP	**14.8**	**6.0**	**2.8**
FAST	**8.0**	**2.2**	**1.0**
FAST-AT	**8.0**	**2.2**	**1.0**
HSTCP	**45.6**	**15.6**	**6.0**
HTCP	**30.2**	**21.4**	**15.0**
Scalable	**33.8**	**13.6**	**6.0**
TCP	**-----**	**112.8**	**30.6**

Table 7-14 shows for each long-lived flow, under the least congested condition, the time (in seconds) taken by each congestion control algorithm to reach its maximum transfer rate of 80×10^3 packets per second. The table reveals that TCP Reno takes much longer to achieve maximum rate than the other algorithms – in fact, TCP Reno does not reach the maximum rate on the long-distance, long-lived flow (**L1**) during TP1 (or any

other time period). Table 7-14 also reveals that FAST and FAST-AT converge most quickly to maximum rate, followed by CTCP. HTCP converges fairly quickly on the long-distance, long-lived flow but lags behind the other non-TCP algorithms in achieving maximum rate on the moderate- and short-distance, long-lived flows.

Table 7-15. Time (seconds) until Long-Lived Flows Recover Maximum Transfer Rate in TP3 for Condition 8

	Long-Lived Flow		
Algorithm	**L1**	**L2**	**L3**
BIC	**186.8**	**6.0**	**21.2**
CTCP	**5.2**	**2.8**	**0.6**
FAST	**18.2**	**0.0**	**1.6**
FAST-AT	**65.4**	**6.8**	**5.8**
HSTCP	**10.4**	**0.0**	**7.8**
HTCP	**26.4**	**0.0**	**3.4**
Scalable	**59.6**	**7.4**	**0.6**
TCP	**----**	**0.0**	**2.4**

Table 7-15 reports for each long-lived flow, under the least congested condition, the time (in seconds) taken by each congestion control algorithm to recover its maximum transfer rate of 80 x 10^3 packets per second after TP3 begins (at t=6000). The table suggests that BIC and FAST-AT are somewhat sluggish in recovery and that CTCP is quite good. The table is difficult to interpret, however, because the numbers seem somewhat inconsistent. For example, FAST, HTCP and TCP require no time to recover maximum transfer rate on long-lived flow **L2**. This means that, for these congestion control algorithms, flow **L2** had already recovered the maximum transfer rate before TP3 began. Perhaps better insight can be provided by examining the time series of goodput for each flow over TP2 and the beginning of TP3, specifically between t=4500 and t=6500, which covers 400 seconds (i.e., 2000 200 ms intervals). We provide eight time series (one per congestion control algorithm) for each long-lived flow.

Fig. 7-51 plots time series for goodput on long-lived flow L1. For the quickest recovering algorithm, CTCP, the plot reveals that goodput varies significantly during TP2, but goodput is generally improving after about t=5200. Further, by t=6000 CTCP has nearly regained its maximum transfer rate. Contrast this with HTCP (second quickest recovering) where goodput is trending generally down until t=6000, after which HTCP recovers quickly. Similarly, FAST shows widely varying goodput during TP2 and then

recovers quickly after *t*=6000. FAST-AT recovers less quickly because the α-parameter had been auto-tuned to 20, whereas FAST used a fixed α-parameter of 200.

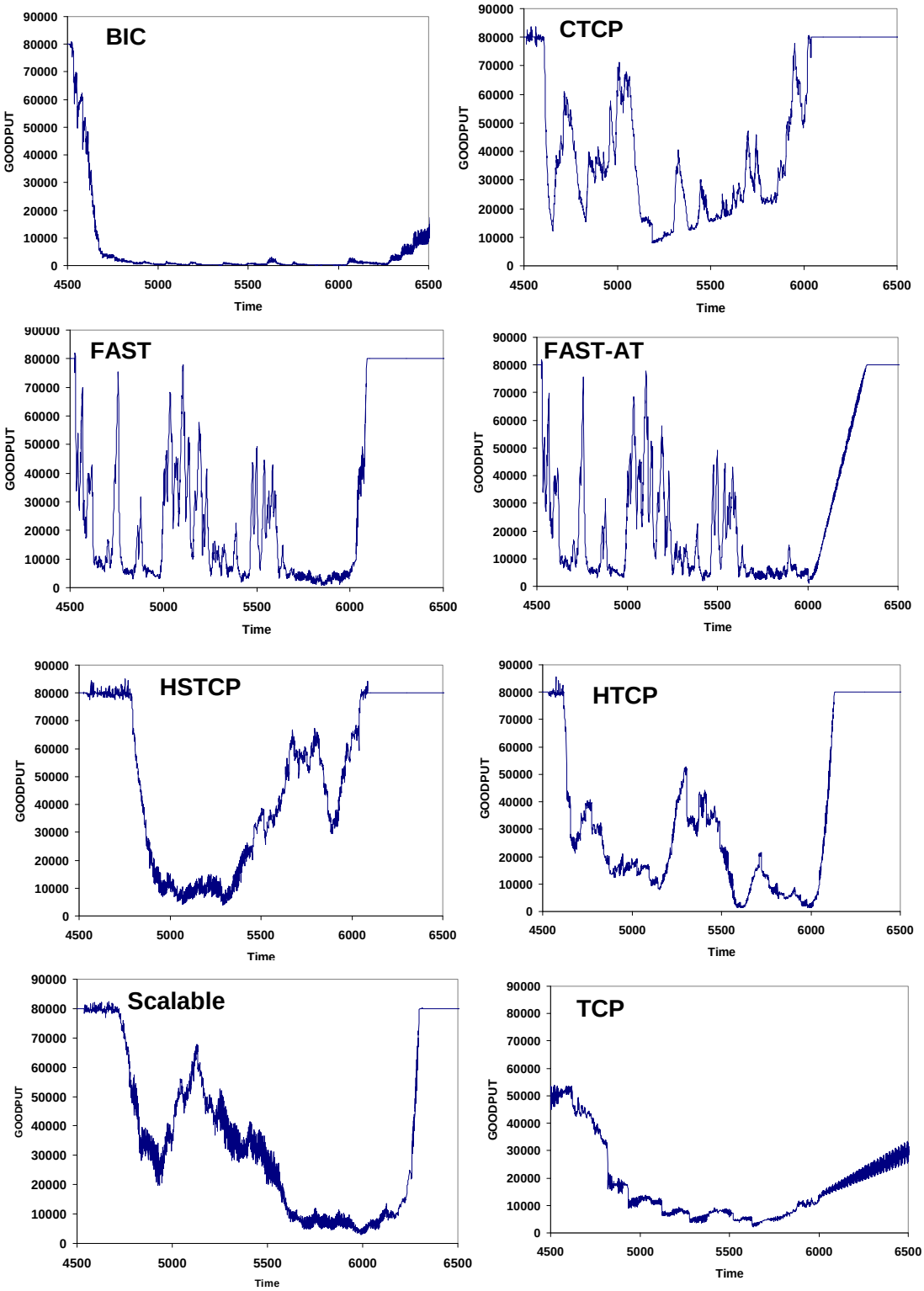

Figure 7-51. Goodput (pps) from *t*=4500 to *t*=6500 for each Congestion Control Algorithm on Long-Lived Flow L1 under Condition 8 – time is in 200 ms interval since beginning of simulation

Fig. 7-51 also shows that goodput for HSTCP is generally improving after about *t*=5400, while goodput for Scalable TCP is trending generally down until *t*=6000 and then begins to recover. On the other hand, goodput under BIC reaches a very low level by *t*=4800 and does not begin to recover until *t*=6250. Goodput under TCP bottoms at *t*=5500 and then begins to recover. While BIC does eventually recover its maximum transfer rate (not shown) during TP3, TCP never achieves the maximum transfer rate on flow **L1** during any time period for condition 8. This highlights the fact that the alternate congestion control algorithms can provide substantial goodput improvement for large files on high-delay, high-speed paths.

In thinking about the results for long-lived flows recall that the long-distance flow **L1** also exhibits the highest congestion from jumbo file transfers during TP2. The moderate-distance flow **L2** suffers least congestion from jumbo file transfers during TP2 and the short-distance flow **L3** suffers congestion at a level between that of **L1** and **L2**. This factor helps explain why it takes less time in Table 7-15 to recover maximum transfer rate on flow **L2** than on flow **L3**. We continue the analysis of results in Table 7-15 with a consideration of flow **L2**.

Fig. 7-52 plots time series for goodput on flow **L2** between *t*=4500 and *t*=6500. Given that flow **L2** traverses the least congested path during TP2, we expect quick recovery during TP3, as shown in Table 7-15, where four of the congestion control algorithms (including TCP Reno) have reestablished maximum transfer rate prior to *t*=6000. The time series plots reveal different behaviors during TP2 among the four congestion control algorithms in question. HSTCP goodput never falls below the maximum transfer rate.[1] HTCP goodput briefly falls below the maximum transfer rate (around *t*=5900) and then recovers quickly. FAST goodput oscillates markedly throughout TP2 and recovers its equilibrium at the maximum transfer rate just prior to *t*=6000. FAST-AT behaves similarly to FAST but recovers its equilibrium somewhat past *t*=6000. On the other hand, TCP Reno reaches a lowest goodput of about 50×10^3 pps at about *t*=5400 and then improves steadily, reaching maximum transfer rate just prior to *t*=6000. Thus, while Table 7-15 reports similar recovery times for a number of alternate congestion control algorithms on flow **L2** under condition 8, the time series reveal behavioral differences among these algorithms.

Fig. 7-52 also shows that CTCP goodput varies somewhat like FAST (and FAST-AT) but the evident oscillations tend to be less frequent with less change in amplitude. On the other hand, Scalable TCP retains maximum goodput until nearly *t*=5500, after which its goodput drops to about 50×10^3 pps, just before *t*=6000, and recovers to 80×10^3 pps by about *t*=6200. BIC goodput drops generally to under 10×10^3 pps at about t=5400 and then rises steadily, reaching 80×10^3 pps at about *t*=6100.

We continue this analysis by considering the temporal behavior of goodput for each congestion control algorithm on the short-distance, long-lived flow **L3**. Fig. 7-53 displays the relevant time series. Recall that flow **L3** faces moderate levels of congestion during TP2 due to competing jumbo file transfers.

[1] Note that the noise around the maximum transfer rate (80×10^3 pps) is due to a measurement artifact. We measure goodput as the rate at which packets are received by the receiver. Since access routers typically operate much faster than sources and receivers, there are periods during which goodput oscillates around the maximum transfer rate due to packet clumping. This oscillation would not appear if we had instead measured the rate at which the receiver emits ACK and NAK packets in response to arriving data packets.

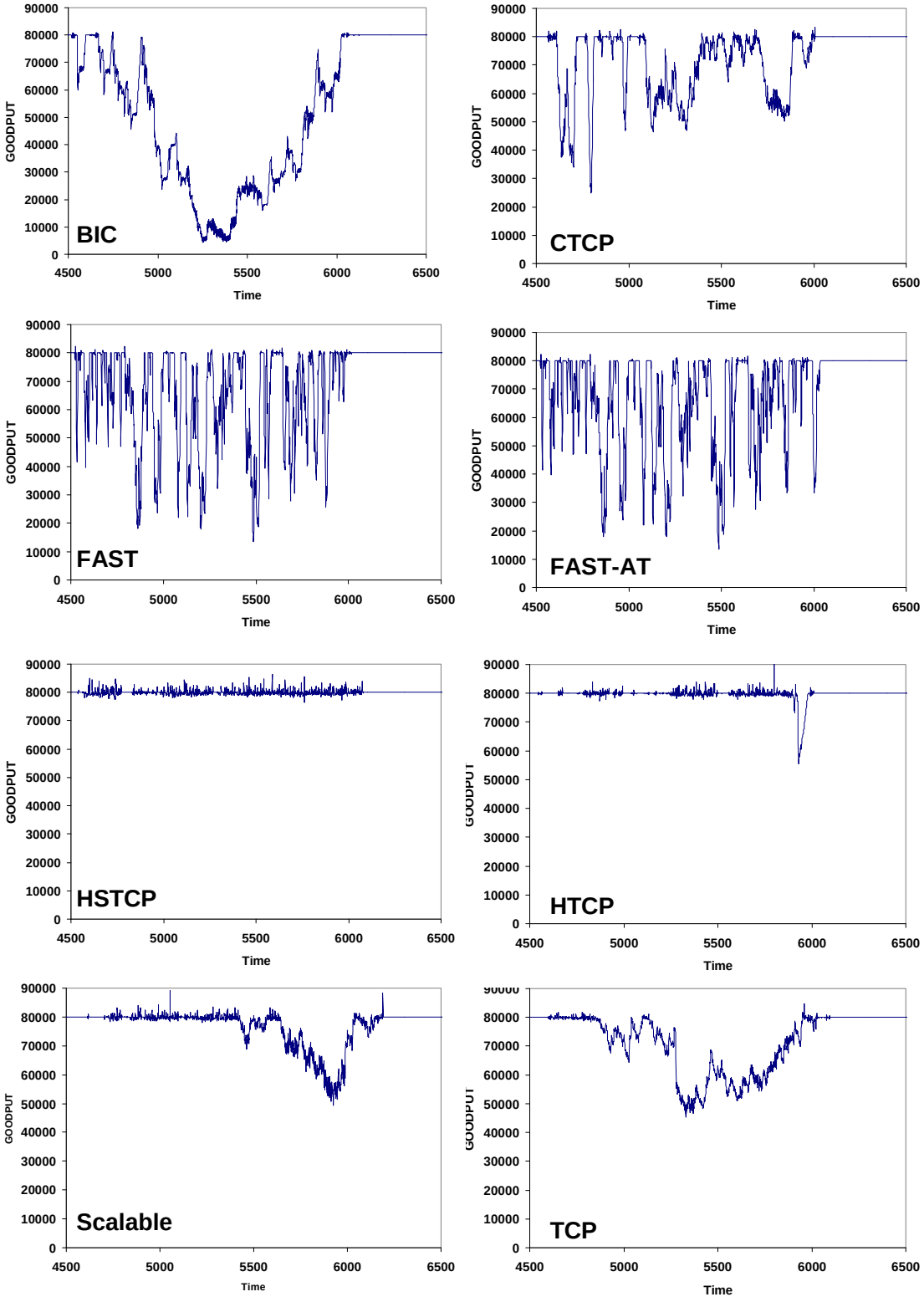

Figure 7-52. Goodput (pps) from *t*=4500 to *t*=6500 for each Congestion Control Algorithm on Long-Lived Flow L2 under Condition 8 – time is in 200 ms intervals since beginning of simulation

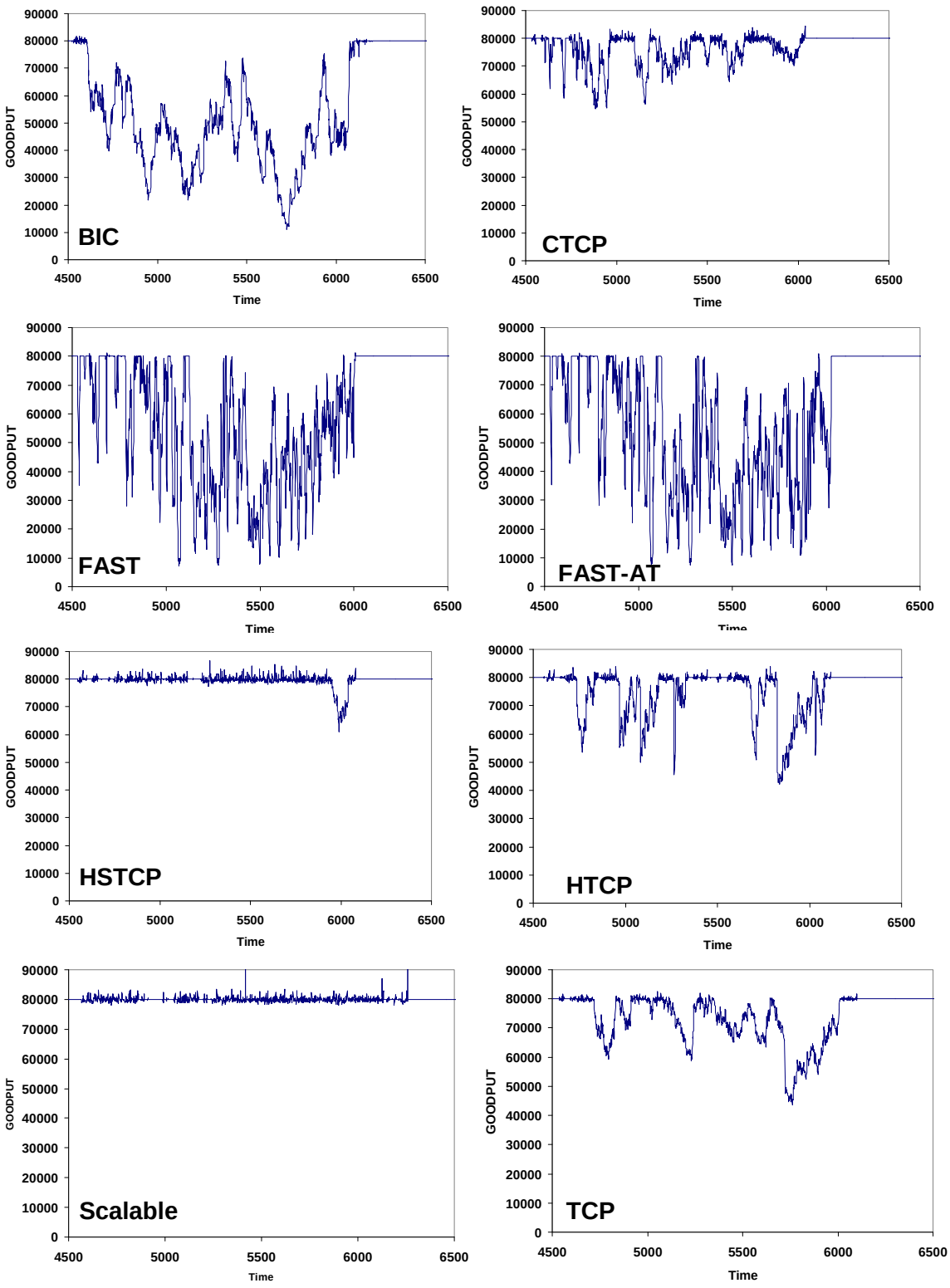

Figure 7-53. Goodput (pps) from *t*=4500 to *t*=6500 for each Congestion Control Algorithm on Long-Lived Flow L3 under Condition 8 – time is in 200 ms intervals since beginning of simulation

Fig. 7-53 shows that FAST and FAST-AT exhibit significant goodput oscillations during TP2, as also seen in Figs. 7-51 and 7-52. CTCP shows less frequent oscillations with lower amplitude. HSTCP and Scalable TCP tend to retain maximum transfer rate for most of TP2. HTCP goodput during TP2 looks very similar to goodput for TCP Reno. BIC appears to suffer two drops and recoveries in goodput during TP2.

Table 7-16 shows the average goodput (divided by 1000) under each congestion control algorithm for each long-lived flow during TP2 given condition 8. Under this condition and time period, BIC actually underperforms TCP Reno on all long-lived flows. FAST and FAST-AT provide no better goodput than TCP Reno, except in the case of the long-distance flow **L1**. We cannot generalize from looking at a single uncongested condition, so we consider information from three additional uncongested conditions: 14, 28 and 32. In this case, we limit our detailed analysis to consider only the long-distance, long-lived flow **L1**. Perhaps this additional analysis will suggest some patterns.

Table 7-16. Average Goodput (pps/1000) for Each Congestion Control Algorithm on Three Long-Lived Flows during TP2 under Condition 8

	Long-Lived Flow		
Algorithm	**L1**	**L2**	**L3**
BIC	**6.391**	**41.751**	**47.809**
CTCP	**34.488**	**69.332**	**75.435**
FAST	**18.068**	**64.444**	**53.162**
FAST-AT	**18.415**	**65.109**	**51.909**
HSTCP	**39.920**	**79.972**	**79.672**
HTCP	**24.794**	**79.845**	**73.935**
Scalable	**37.414**	**75.970**	**79.983**
TCP	**16.278**	**69.223**	**71.777**

Table 7-17 reports the lag time (in seconds) until each congestion control algorithm achieves maximum transfer rate on flow **L1** under three uncongested conditions. The relative ordering is the same as appeared in Table 7-14: FAST and FAST-AT reach maximum rate soonest, followed by CTCP, HTCP, Scalable TCP, BIC and HSTCP. TCP Reno does not achieve maximum transfer rate during TP1. The measured time lags suggest that FAST, FAST-AT, CTCP and HTCP can be grouped together as the set of algorithms providing superior quickness in attaining maximum transfer rate. Scalable TCP, BIC and HSTCP achieve less impressive quickness.

Table 7-18 reports the lag time until each congestion control algorithm recovers maximum transfer rate after TP2 on flow **L1** under uncongested conditions 14, 28 and 32. As expected, TCP Reno does not achieve maximum transfer rate during TP3. Surprisingly, perhaps, FAST-AT also does not recover to the maximum transfer rate.

This occurs because during TP2 FAST-AT auto-tunes the α-parameter from 200 to 20 to 8 as throughput falls on flow **L1**. Over the course of TP3 the α-parameter recovers as throughput rises but only reaches 20, which provides insufficient upward thrust on goodput.

Table 7-17. Time (seconds) until Long-Lived Flow L1 Reaches Maximum Transfer Rate in TP1 for Three Uncongested Conditions

	Condition		
Algorithm	**C14**	**C28**	**C32**
BIC	**120.6**	**120.6**	**120.6**
CTCP	**35.2**	**35.2**	**35.2**
FAST	**26.2**	**26.2**	**26.2**
FAST-AT	**26.2**	**26.2**	**26.2**
HSTCP	**128.0**	**128.0**	**128.0**
HTCP	**41.0**	**41.0**	**41.0**
Scalable	**78.2**	**78.2**	**78.2**
TCP	**-----**	**-----**	**-----**

Table 7-18. Time (seconds) until Long-Lived Flow L1 Recovers Maximum Transfer Rate in TP3 for Three Uncongested Conditions

	Condition		
Algorithm	**C14**	**C28**	**C32**
BIC	**228.6**	**186.8**	**172.2**
CTCP	**133.4**	**96.6**	**122.2**
FAST	**130.6**	**109.0**	**145.4**
FAST-AT	**-----**	**-----**	**-----**
HSTCP	**205.8**	**177.6**	**190.6**
HTCP	**145.2**	**174.2**	**125.0**
Scalable	**135.2**	**127.0**	**175.4**
TCP	**-----**	**-----**	**-----**

Among the other congestion control algorithms, Table 7-18 reveals that recovery time lags cannot be grouped as clearly as occurred for the initial lag in attaining maximum transfer rate. This makes sense because flow **L1** contends with great congestion during TP2. In general, Table 7-18 suggests that FAST and CTCP recover most quickly followed by Scalable TCP and HTCP. HSTCP and BIC appear to lag. Regarding the performance of FAST, the reader should remember that fewer flows operate simultaneously because under FAST flows have more difficulty connecting.

Table 7-19 reports average goodput (divided by 1000) on flow **L1** during TP2 under each of three uncongested conditions: 14, 28 and 32. Here, Scalable TCP, HSTCP and BIC tend to retain higher goodput under the contention of an increasing number of jumbo files during TP2. This behavior, also evident in the previous experiment, indicates that newly arriving flows have more difficulty obtaining a fair share of goodput under these three congestion control algorithms. CTCP and HTCP show some ability to retain goodput during TP2. FAST and FAST-AT appear to reduce goodput significantly in the face of increased congestion. TCP Reno did not reach high levels of goodput and so it is not surprising that it provides low goodput during TP2.

To better understand the measures reported in Tables 7-17, 7-18 and 7-19, we provide the related time series plots for each condition as Figs. 7-54 (condition 14), 7-55 (condition 28) and 7-56 (condition 32). These figures reveal that FAST and FAST-AT quickly reduce goodput on **L1** in reaction to congestion. Subsequently, as congestion clears (around t=6500) FAST quickly recovers maximum goodput. FAST-AT, on the other hand, recovers maximum goodput more slowly as the α-parameter is auto-tuned upward only every 200 s. The figures also show that TCP Reno achieves only about 25 % of the maximum transfer rate prior to TP2 and then resumes its linear increase in goodput as congestion clears. CTCP takes longer to reduce goodput in reaction to congestion but then recovers to the maximum transfer rate quickly after congestion begins to clear. HTCP shows a pattern similar to CTCP. BIC reduces goodput on flow **L1** slowly over a period of 200 s and then recovers over a period of about 100 s after congestion starts to clear. HSTCP shows a pattern similar to BIC. Scalable TCP loses goodput slowly on flow **L1** over 200 s but the minimum goodput stays higher (around 20×10^3 pps) than is the case for the other congestion control algorithms. Once congestion begins to clear, Scalable TCP recovers maximum goodput somewhat quickly (within about 80 s).

To conclude our analysis of the various congestion control algorithms performing on long-lived flows, we consider flow **L1** under the most congested condition (21). We examine performance over all three time periods. We expect to learn how the congestion control algorithms react during TP1 where they face more intense competition than was the case under less congested conditions (e.g., conditions 8, 14, 28 and 32). Under TP2 and TP3 we expect the congestion control algorithms to perform similarly because the congestion arising from jumbo files in TP2 is unlikely to clear during TP3. Table 7-20 reports the average goodput (divided by 1000) on flow **L1** for each congestion control algorithm in each of the three time periods. As expected, all algorithms provide very little goodput during TP3. The minor differences in goodput during TP2 appear due to variations in the rate at which the algorithms shed goodput in the face of intensifying congestion. These issues have been examined in earlier paragraphs. Here, we focus on TP1. To augment Table 7-20 we provide Fig. 7-57, which plots time series for goodput over all three time periods for each congestion control algorithm.

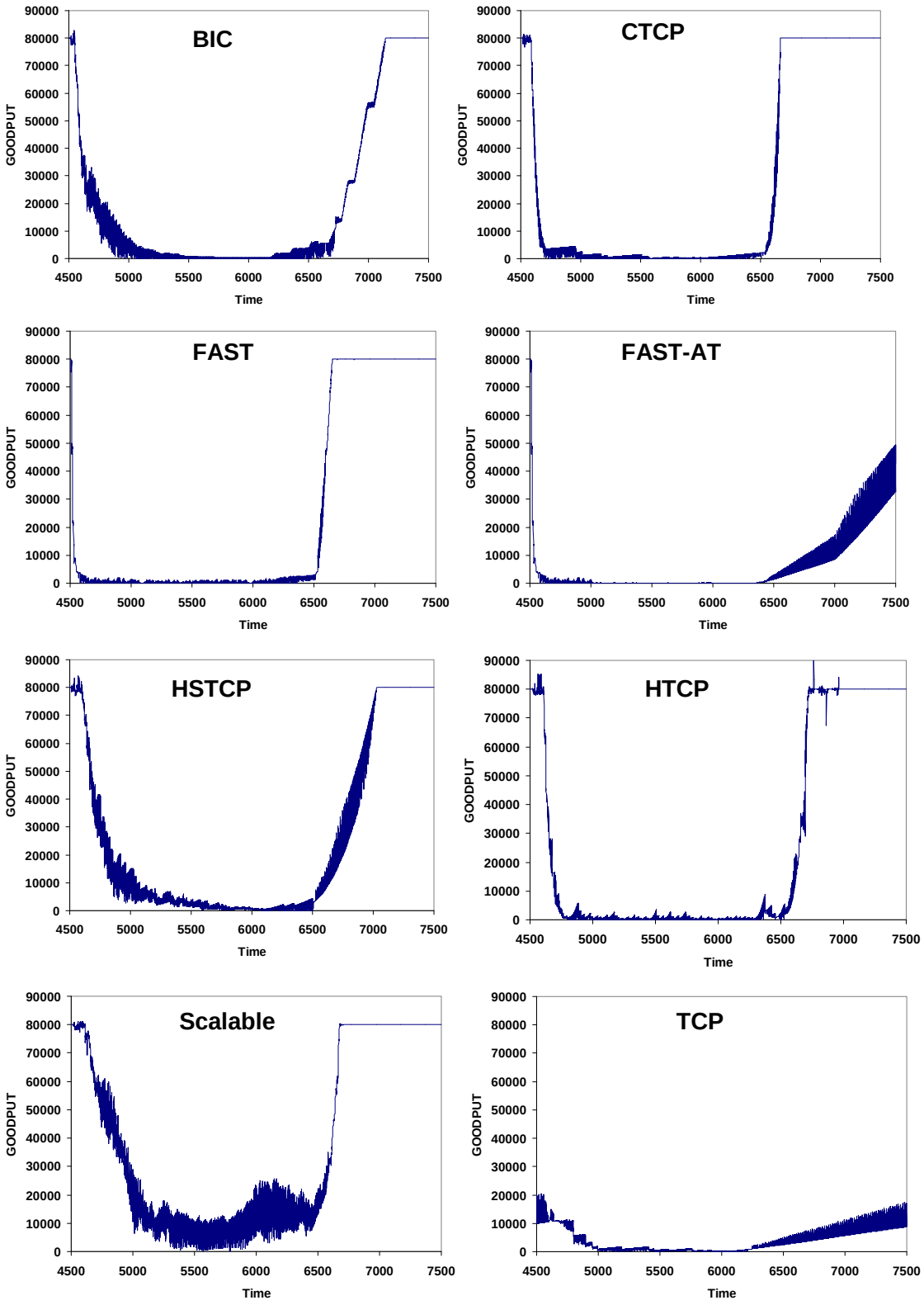

Figure 7-54. Goodput (pps) from *t*=4500 to *t*=7500 for each Congestion Control Algorithm on Long-Lived Flow L1 under Condition 14 – time is in 200 ms intervals since beginning of simulation

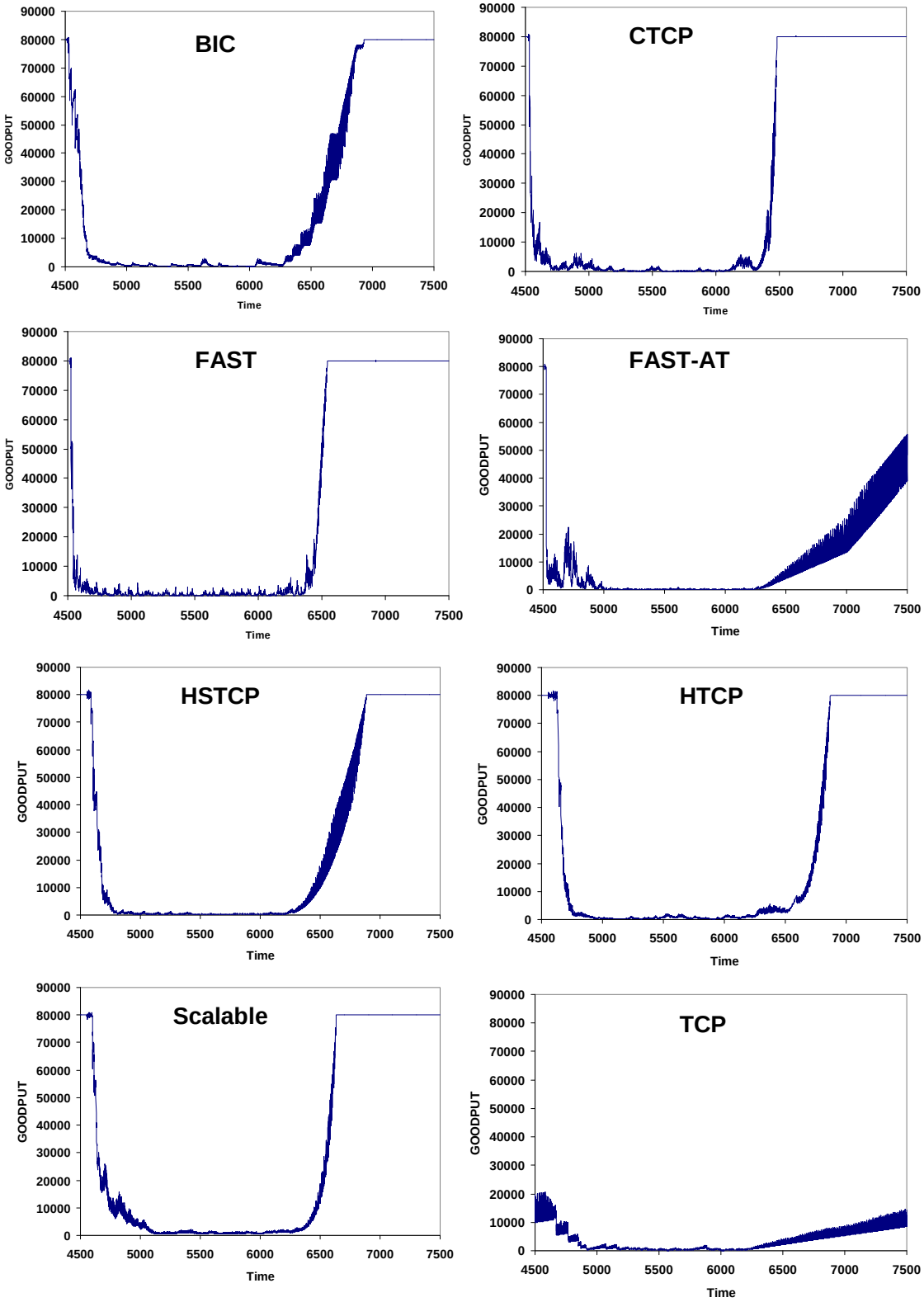

Figure 7-55. Goodput (pps) from *t*=4500 to *t*=7500 for each Congestion Control Algorithm on Long-Lived Flow L1 under Condition 28 – time is in 200 ms intervals since beginning of simulation

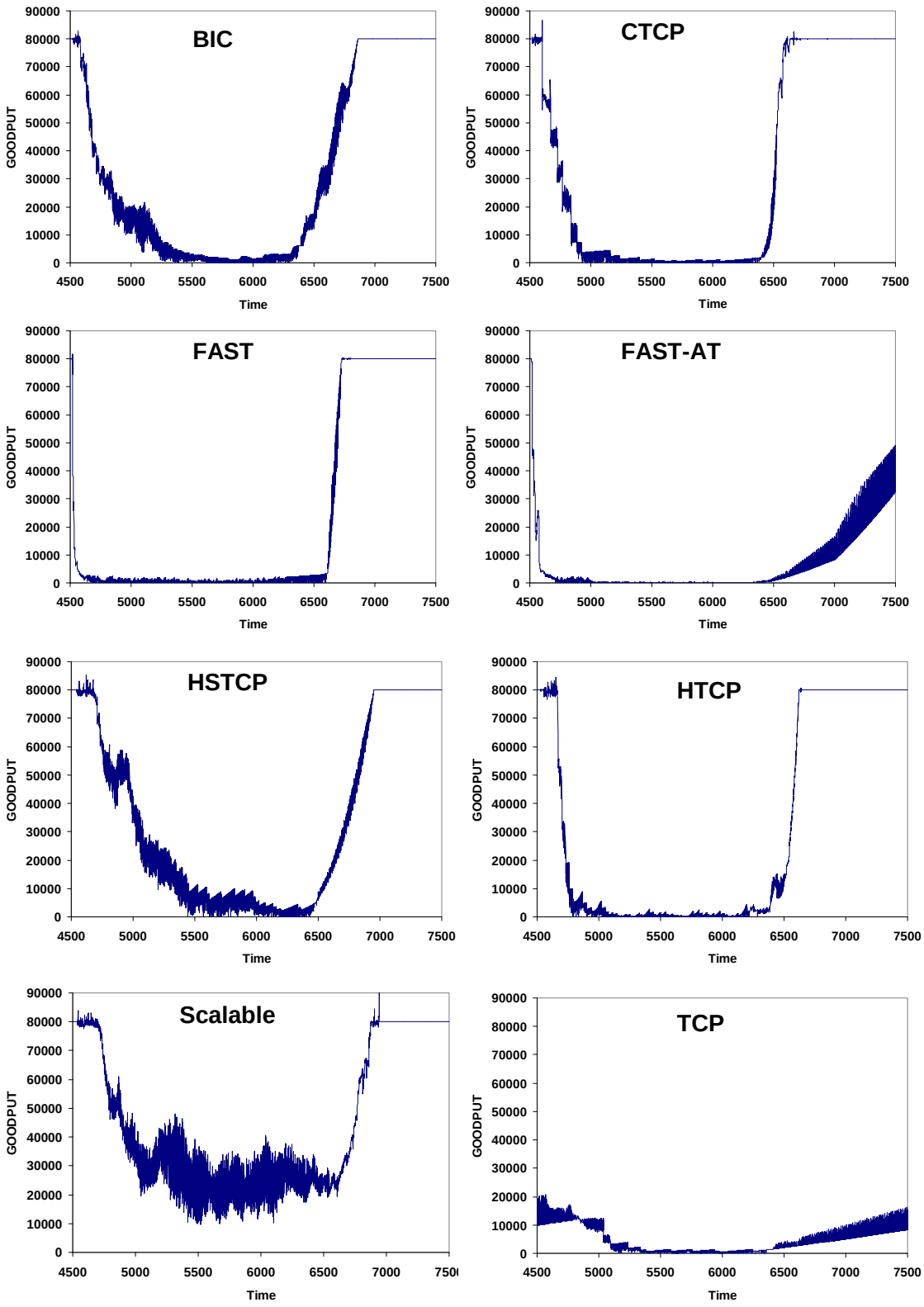

Figure 7-56. Goodput (pps) from *t*=4500 to *t*=7500 for each Congestion Control Algorithm on Long-Lived Flow L1 under Condition 32 – time is in 200 ms intervals since beginning of simulation

Table 7-19. Average Goodput (pps/1000) on Long-Lived Flow L1 in Time Period 2 for Each Congestion Control Algorithm under Each of Three Uncongested Conditions

	Condition		
Algorithm	**C14**	**C28**	**C32**
BIC	**9.533**	**6.391**	**16.951**
CTCP	**7.511**	**3.435**	**13.000**
FAST	**1.709**	**2.701**	**2.272**
FAST-AT	**1.505**	**2.859**	**2.690**
HSTCP	**15.114**	**7.792**	**29.338**
HTCP	**8.635**	**9.339**	**11.943**
Scalable	**24.756**	**10.085**	**38.446**
TCP	**3.160**	**2.737**	**5.333**

Table 7-20. Average Goodput (pps/1000) on Long-Lived Flow L1 for Each Congestion Control Algorithm in Each of the Three Time Periods under Most Congested Condition 21

	Time Period		
Algorithm	**TP1**	**TP2**	**TP3**
BIC	**50.154**	**0.909**	**0.116**
CTCP	**35.950**	**0.353**	**0.130**
FAST	**39.740**	**0.398**	**0.117**
FAST-AT	**40.368**	**0.375**	**0.086**
HSTCP	**50.880**	**0.540**	**0.117**
HTCP	**47.097**	**0.759**	**0.110**
Scalable	**55.591**	**1.061**	**0.117**
TCP	**25.106**	**0.546**	**0.112**

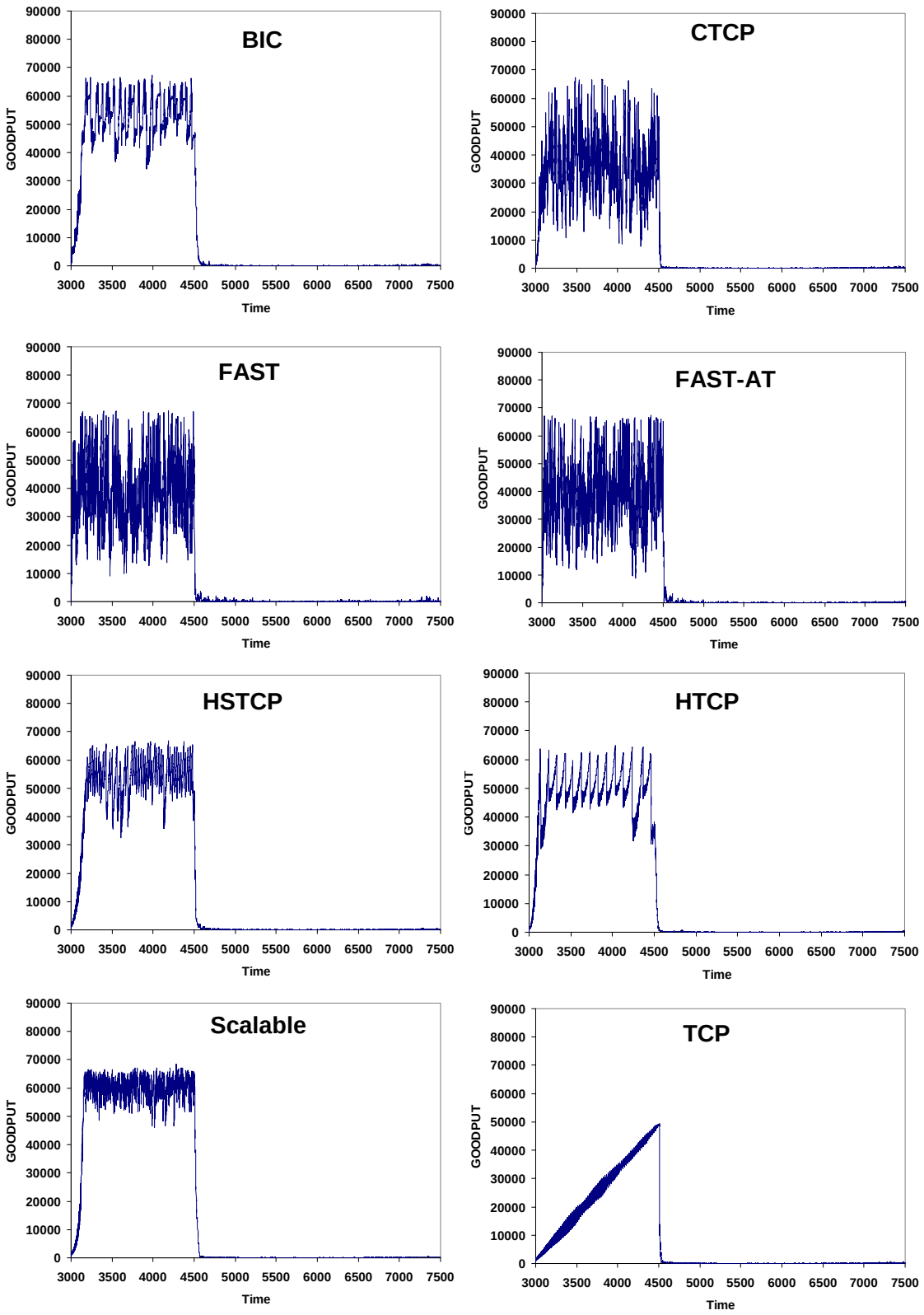

Figure 7-57. Goodput (pps) from *t*=3000 to *t*=7500 for each Congestion Control Algorithm on Long-Lived Flow L1 under Condition 21 – time is in 200 ms intervals since beginning of simulation

Given heavy congestion from competing Web traffic and document downloads during TP1 under condition 21, the ability of Scalable TCP, HSTCP and BIC to resist losing goodput allows then to achieve higher average goodput. As show in Fig. 7-57, Scalable TCP keeps goodput within the range of about 50×10^3 to 65×10^3 pps, HSTCP within the range of about 45×10^3 to 60×10^3 pps and BIC within the range of 40×10^3 to 65×10^3 pps. The higher, narrower goodput range of Scalable TCP accounts for higher average goodput. Fig. 7-57 shows that HTCP, with 4th highest average goodput, allows the range to vary between about 30×10^3 and 60×10^3 pps. CTCP, FAST and FAST-AT oscillate more frequently and with larger variation, ranging between about 15×10^3 and 65×10^3 pps during TP1. TCP Reno linearly increases over TP1 from about 5×10^3 pps at t=3000 to about 50×10^3 pps at t=4500.

Clearly, under many conditions and time periods for long-lived flows, and in the absence of a large initial slow-start threshold, the alternate congestion control algorithms provide improved goodput over TCP Reno. An exception to this occurs during TP2 when most of the algorithms cannot provide much goodput. Even in such cases, selected congestion control algorithms (Scalable TCP, HSTCP and BIC) tend to retain higher goodputs a bit longer than others. When ramping up toward maximum goodput under light to moderate Web traffic and document downloads, FAST, FAST-AT, CTCP and HTCP show some advantage in quickness over the other algorithms. When competing for goodput against heavy Web traffic and document downloads, Scalable TCP, HSTCP and BIC show some advantage in retaining higher goodput. When recovering from periods of intense jumbo file transfers, CTCP, FAST and HTCP show some advantage in recovering maximum goodput under uncongested conditions. These differences among the congestion control algorithms appear more readily under longer propagation delays, with differences shrinking along with falling propagation time.

What might be the practical implications of these findings? First, for alternate congestion control algorithms to gain a significant advantage in goodput over standard TCP, the file to be transferred must be large, the initial slow-start threshold must be low (relative to the file size), the source and receiver must both have high-speed network connections, the propagation delay must be long and the network path must be fast enough and uncongested enough to support a stream of traffic at the rate of the high-speed network connections between the source and receiver. This combination of conditions is likely to prove relatively rare within an operating network, as was the case for our simulated network. Of course, this combination of conditions is typically established (artificially) in support of attempts to show how fast a file can be transferred across a network path using a particular transport protocol. Second, any advantage for the alternate congestion control algorithms would likely be enhanced should the path show occasional packet losses due to noise (i.e., bit-error rate) or hardware malfunctions. This follows because some of the alternate congestion control algorithms (e.g., FAST, FAST-AT, CTCP and HTCP) recover more quickly from sporadic packet losses, while others (e.g., Scalable TCP, HSTCP and BIC) exhibit smaller rate reduction on a sporadic loss. Third, regular patterns of packet losses due to high congestion would limit any advantage of the alternate congestion control algorithms, excepting that Scalable TCP, HSTCP and BIC tend to retain goodput a bit longer than other algorithms in the face of congestion.

7.5.4 Finding #4

As in the previous experiment (see Sec. 6.5.3), CTCP (algorithm 2) can drive congestion window size to substantially higher values than the other congestion control algorithms we simulated. This behavior arose during TP3, as shown in Fig. 7-36, which analyzes average congestion window size. Detailed examination of the relevant time series revealed that this increase in congestion window size can be attributed solely to DD flows. The reason this occurs is the same as explained in Sec. 6.5.3. CTCP increases the delay window exponentially when no congestion had been detected and the actual congestion window is within 30 packets of the expected congestion window. This set of conditions can occur on DD flows as congestion eases at the onset of TP3.

Recall that during TP2 jumbo file transfers were initiated on DD flows, which introduced substantial congestion in directly connected access routers. At the onset of TP3 no further jumbo transfers are initiated and congestion eases as residual jumbo transfers complete. During this easing period, the congestion window on DD flows can increase – the rate of increase depends upon the level of congestion created during TP2. For example, Fig. 7-58 plots, for seven congestion control algorithms, the increase in average congestion window for DD flows during TP3 under condition 8 (most uncongested).

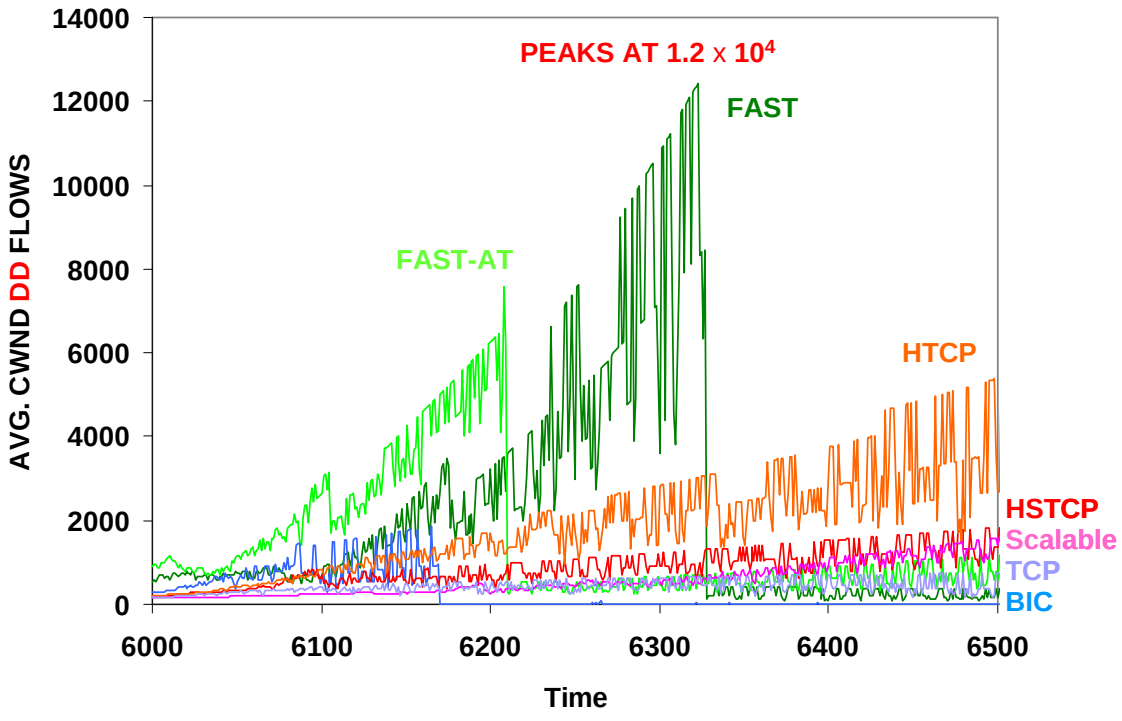

Figure 7-58. Average Congestion Window Size (packets) of DD Flows during TP3 under Condition 8 for BIC, FAST, FAST-AT, HSTCP, HTCP, Scalable TCP and TCP Reno – time is in 200 ms intervals since the beginning of the simulation

Fig. 7-58 shows that four (BIC, HSTCP, Scalable and TCP Reno) of the congestion control algorithms provide a linear increase (with a small slope) in average congestion window size, up to a maximum of about 10^3 packets. HTCP provides a similar

linear increase but with a higher slope and, thus, reaches a maximum congestion window size of about 4 x 10^3 packets. The increases for FAST-AT and FAST (which also appear approximately linear but with large slopes) peak at around 6 x 10^3 and 12 x 10^3 packets, respectively. The situation for CTCP is much different, as shown in Fig. 7-59, where under the same conditions the average congestion window size increases exponentially, reaching a peak of about 170 x 10^3 packets.

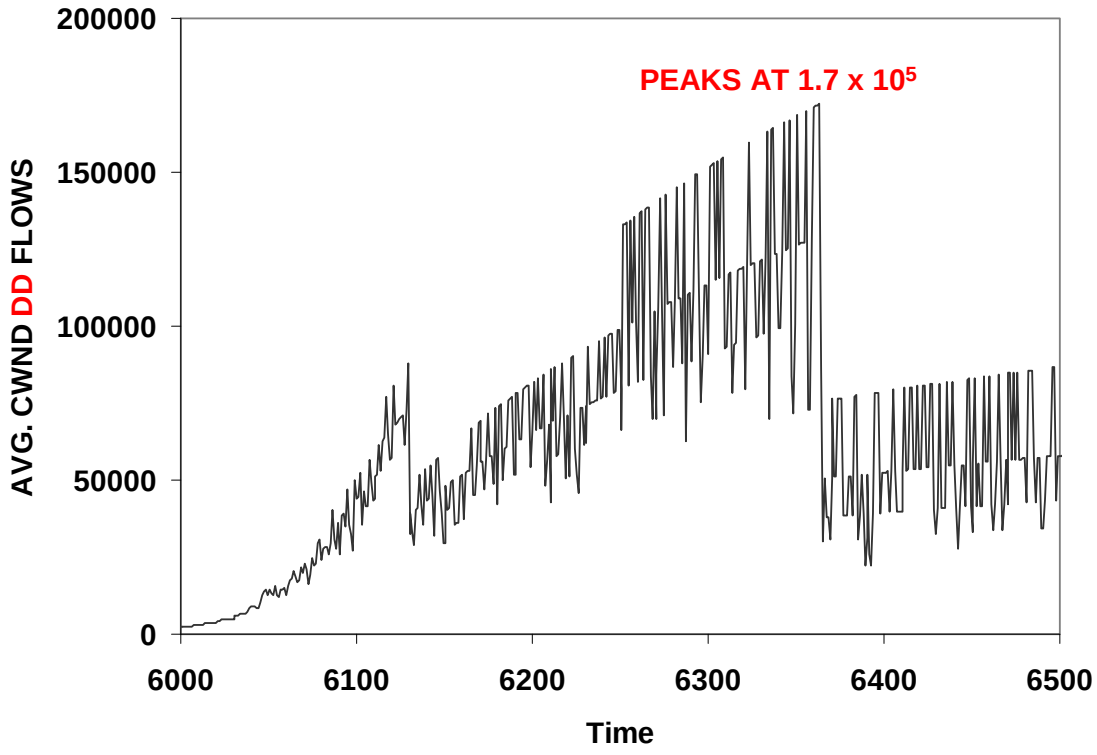

Figure 7-59. Average Congestion Window Size (packets) of DD Flows during TP3 under Condition 8 for CTCP – time is in 200 ms intervals since beginning of simulation

Overall, the congestion window size on DD flows during TP3 under condition 8 is lower for all congestion control algorithms as compared with results for the most uncongested condition (condition 12) in the previous experiment (see Figs. 6-67 and 6-68). The lower congestion window size is attributable to the order of magnitude lower network speeds used in the current experiment. While lower network speed bounds the ability of CTCP to increase the congestion window on DD flows under uncongested conditions during TP3, the general behavioral difference, first reported in Sec. 6.5.3, between CTCP and the other algorithms remains discernible.

7.5.5 Tendencies

While not consistently statistically significant across all conditions, buffer monitoring in the six directly connected access routers (B0a, C0a, E0a, F0a, I0a and K0a) revealed that Scalable TCP, BIC and HSTCP tend toward higher buffer utilizations than the other congestion control algorithms. For example, see Figs. 7-18, 7-30 and 7-40. This finding

appears consistent (at least with respect to Scalable TCP and BIC) with our measurements of buffer utilization when simulating the congestion control algorithms in a dumbbell topology (recall Sec. 5.4). This finding may also relate to tendencies of Scalable TCP to inject more packets into the network than other algorithms, to hold higher goodput on long-lived flows after the onset of jumbo file transfers in TP2 and to share goodput unfairly among competing flows.

7.6 Conclusions

In this section we described an experiment comparing alternate congestion control algorithms deployed in a scaled-down network with about an order of magnitude fewer sources and lower network speed than used in our previous experiment (described in Chapter 6). In addition, we reduced the initial slow-start threshold to a relatively low value and we added a congestion control regime: FAST with α-tuning enabled. We subjected each of eight algorithms to the same 32 conditions, which covered a range of congestion levels.

We demonstrated that FAST and FAST-AT exhibit similar influence on macroscopic network behavior and we showed that enabling α-tuning caused FAST-AT to recover less quickly than FAST when congestion eases after periods of increased contention. We also showed that, under the scenario and conditions of this experiment, the congestion control algorithms (aside from FAST and FAST-AT) exhibited indistinguishable macroscopic behavior and modest differences in experience for typical users. We showed that FAST and FAST-AT can exhibit distinctive, undesirable network-wide behavior, which grows more distinctive under increasing congestion. We also confirmed that the CTCP delay-window adjustment algorithm can lead to an exponential increase in congestion window size under particular circumstances associated with easing congestion. We identified some tendencies for Scalable TCP, BIC and HSTCP to utilize more buffers.

We were able to show that under specific, constrained circumstances the alternate congestion control algorithms can provide higher goodput than TCP Reno. For alternate congestion control algorithms to gain a significant advantage in goodput over the standard TCP algorithm, the file to be transferred must be large, the initial slow-start threshold must be low (relative to the file size), the source and receiver must both have high-speed network connections, the propagation delay must be long and the network path must be fast enough and uncongested enough to support a stream of traffic at the rate of the high-speed network connections between the source and receiver. The advantage of alternate congestion control algorithms may be expected to increase in the presence of sporadic losses. We were also able to identify some specific circumstances where particular alternate congestion control algorithms performed similarly. Under a low initial slow-start threshold and competing with typical Web traffic, FAST, FAST-AT, CTCP and HTCP tended to attain maximum transfer rate more quickly on long-lived flows than other algorithms. Under heavy congestion, Scalable TCP, BIC and HSTCP tended to retain higher goodput for a longer time on long-lived flows. Under easing congestion for long-lived flows, FAST and CTCP tended to recover maximum transfer rate more quickly than other algorithms. Overall, for long-lived flows, Scalable TCP tended to provide highest goodput but at the cost of some unfairness with respect to competing flows.

In the next two chapters, we shift our assumptions to explore alternate congestion control algorithms with richer traffic classes in a network under relatively low congestion. We simulate a network with a mix of sources, some operating under standard TCP congestion control procedures and some operating under an alternate congestion control algorithm. We consider conditions where most of the network uses standard TCP as well as conditions where most of the network uses an alternate algorithm. We also extend our traffic classes beyond Web browsing to include some proportion of downloading for larger files, such as software service packs and movies. We simulate a full hour of network operation under 32 different conditions and then compare macroscopic network behavior among the algorithms. We also investigate relative goodputs experienced by comparable flows. In Chapter 8 we examine a scaled-down network in two different cases: (1) with a high initial slow-start threshold and (2) with a low initial slow-start threshold. In Chapter 9 we examine a large, fast network with a high initial slow-start threshold. In these experiments, we aim to understand whether alternate congestion control algorithms might prove beneficial for flows with specific characteristics. We also aim to determine if particular congestion control algorithms might have deleterious effects on competing flows using standard TCP.