Performance Analysis of TCP Variants

Neeki Hushyar
UMass Amherst
nhushyar@cs.umass.edu

Angela Upreti
UMass Amherst
aupreti@cs.umass.edu

Keywords

TCP variants; Comparison; SACK; Reno; New Reno, Vegas; Tahoe

1. INTRODUCTION

Web traffic runs over TCP and given the scale at which the internet has grown and how much we rely on the web, it is important that the underlying transport protocol is able to operate at a reasonably low latency and provide high throughput. Network congestion causes packet loss which increases latency and reduces throughput. Using the right TCP variant saves both money and time. Amazon found that every 100ms latency cost them 1 percent in sales [1]. Higher latency/wait-time in web applications discourages clients from revisiting the cite. This is specially true for video applications such as youtube. Clients usually close the page if the latency is high. Fairness in the underlying transport layer is equally important. One flow should not hog most of the network bandwidth and throttle the other flows. Otherwise, different customers paying the same amount for internet service would get widely different performance. Metrics such as throughput, average and end-to-end latency, packet drop rate, fairness are different across different tcp variants.

Congestion control algorithm for TCP was proposed by Van Jacobson in response to 1986's 'congestion collapse'. The original TCP proposed by Jacobson contained elements such as dynamic window sizing, slow start, additive increase and multiplicative deacrease (AIMD). Since then, several variants of TCP have been proposed. TCP Reno adds fast recovery and fast retransmit to TCP Tahoe. TCP Reno's fast recovery and fast retransmit suffer when multiple packets are dropped from the same window. In presence of multiple packet loss, the TCP pipe often gets drained. NewReno has to wait for a retransmit timeout to be able to send data again. Consequently, multiple packet loss leads to low throughput in Reno. TCP NewReno was introduced to improve performance in presence of multiple packet loss. When in fast recovery, TCP NewReno interprets a single partial Ack as a signal to retransmit another packet rather. Unlike Reno, NewReno does not wait for 3 duplicate ACKs before retransmitting. TCP SACK is another TCP variant that solves the multiple packet drop problem in Reno. SACK allows the receiver to acknowledge any non sequential packets it has received. NewReno without SACK is unable to tell exactly which packets are missing. While NewReno can retransmit only one lost packet per RTT, TCP SACK can retransmit as many lost packets as the congestion window allows. Variant Vegas uses RTT as an estimate of congestion rather than using packet loss.

We conduct three different simulation studies in this paper:

1. Compare performances of Tahoe, Reno, NewReno, and Vegas under congestion.
2. Compare fairness between different TCP variants.
3. Study the influence of queuing disciplines (1)RED and (2) DropTail on TCP Sack and TCP Reno.

We show that performance under congestion is influenced by how the fast recovery works in the algorithm, the way the algorithms handle partial acks and how careful the algorithms are about congestion avoidance. Vegas and NewReno give good throughput under congestion. We found that variants that are more aggressive, tend to hog a larger share of bandwidth when used with other TCP variants or when started early. Queuing algorithms RED and DropTail show a trade-off between throughput and latency. In general, RED provides lower latency while Droptail gives higher throughput. This degree of this latency-throughput trade off is dependent on the TCP variant used.

2. METHODOLOGY

2.1 The Setup

We use ns2 to create the topology in Figure 1. The setup consists of five different nodes, N1- N5. The link between each pair of nodes is a duplex link with a capacity of 10Mb.

2.2 Calculation

NS2 trace files were used to calculate drop rate, average throughput, average end-to-end latency and end-to-end latency over time for combinations of different TCP variants and different queuing algorithms. The trace files were parsed using a python script because of the ease of text parsing with python. We used a python library called the Matplotlib to

WOODSTOCK '97 El Paso, Texas USA

DOI: 10.475/123_4

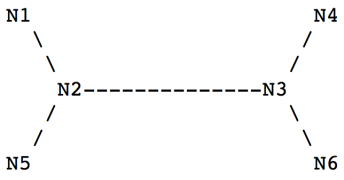

Figure 1: We attach TCP agent with FTP as the application and UDP agent with a CBR flow.

generate the plots. We use a bash script to automate a series of experiments with varying TCP variant, CBR rate, CBR packet size and queuing algorithms.

1. **Average Throughput:** To calculate the average throughput of a tcp stream, we divide the sum of packet sizes of all packets of the TCP stream that reached the destination by the total time of the tcp flow.

2. **Latency:** To calculate the end-to-end latency over time, we recorded the time it took for each packet to get from the source to the destination. We took the average of these latencies to get the average latency.

3. **Drop Rate:** The ns trace clearly identifies the dropped packets. To calculate the drop rate of a flow, we divided the total number of drops by the total number of packets in that flow.

We did multiple runs of our simulations to verify that we see the same trends and reach similar conclusions in terms of throughput, latency and the drop rate. Whenever possible, we averaged our results over multiple runs. To verify that our results are statistically significant, this study can be extended by incorporating t tests .

2.3 TCP Performance Under Congestion

This experiment was designed to test the performance of different TCP variants Tahoe, Reno, NewReno and Vegas under various levels of congestion. TCP agent was attached to start at N1 and sink at N4. FTP application was attached to the TCP agent to simulate traffic that is typical of FTP. CBR flow was added to go from N2 to N3 to create congestion in the link between N2 and N3. We varied the CBR rate from 1 Mbps to 10 Mbps to create different levels of congestion. We had the liberty of choosing the packet size to maintain the CBR. To study the effects of varying CBR packet sizes, we also simulated varying packet sizes, 500B to 10,000B, for all four TCP variants. We did not change the default ns2 buffer size.

2.4 Fairness between TCP variants

This experiment simulated two TCP flows at the same a time. The variant pairs that were simulated are (1) Tahoe/Tahoe (2) Reno/Reno (3) NewReno/Reno (4) Vegas/Vegas (5) NewReno/Vegas. Like before, to create varying levels of congestion, a CBR flow was added to originate at N2 and end at N3. The first TCP stream originated at added N1 and ended at N4. The second TCP stream originated at N5 and ended at N6. For each pair of the variants, we ran simulations for CBR ranging from 1 Mbps to 10 Mbps. The packet size was increased up from 500 for all the experiments. We started with 500B as the packet because it the size of an average IP packet size is around 600B. Accounting for the IP headers, we wanted a smaller packet size for UDP. We also started the two competing variants at different times and ran the simulations for a bit longer (15s) to check if a fair throughput could be achieved over time if one of the TCP streams was delayed.

2.5 Influence of Queuing

In this experiment, we study the influence of queuing algorithms RED and DropTail on TCP Reno and SACK. The set up for this experiment is the same as experiment 1. We first start the TCP stream, wait till it has passes the slow start phase(at 4s) and then start the CBR flow. We conduct the experiments for two different buffer sizes, 100 and a 300. To compare the performance of RED and DropTail in the TCP variants and the CBR flow, we compare the differences in throughput over time, end-to-end latency over time and the drop rate between the queuing disciplines.

3. EXPERIMENT 1: ANALYZING PERFORMANCE OF TCP VARIANTS

This experiment analyzed the performance of different TCP variants over a setup consisting of a single TCP stream and a single CBR flow. The performance of the TCP variants were analyzed over a range of different packet sizes and transmission rates assigned to the CBR. We analyze the results of this experiment in two parts. First, by looking at the performance of the variants over an increasing range of CBR packet sizes, while the rate of the CBR is constant at 2Mbps. Second, we look at the performance of the variants over an increasing range of CBR flows - from 1 Mbps to 10 Mbps, while the size of the initial packet is held constant at 3,000 bytes.

When holding the CBR steady, at 2Mbps, none of the variants experienced any packet loss over the range of increasing packet sizes, from 500 bytes to 8,000 bytes. The average and end-to-end latency as well as the throughput for TCPs Reno, NewReno and Tahoe were all the same. The throughput hovered around 1 Mbps as the size of packets increased and the average latency hovered around .02 seconds per segment. The average and end-to-end latency of TCP Vegas was slightly smaller, likely because Vegas never saturates the full capacity of the link. Doing so, would imply congestion and the throughput would decrease. The throughput for Vegas hovered at just under .99 Mbps as the size of the packets increased and the latency hovered around .005 seconds.

When the packet size is held steady, at 3,000 bytes and the CBR flow is increased from 1 Mbps to 10 Mbps, all four variants suffer packet loss. Reflected in Figure 3, TCPs Reno, NewReno and Tahoe suffer their first packet drops at a CBR of 7 Mbps. TCP Vegas does not suffer any packet drops, but the throughput is consistently lower than the other variants, as it is reducing its congestion window size to reflect the increasing RTT of the packets. TCP Vegas takes preventative measures to avoid packet loss. The other variants make changes only in the aftermath of packet loss. To prevent congestion, Vegas records the round trip time (RTT) of packets and adjusts its congestion window according to any delay. Our experiments never reached a point at which TCP Vegas experienced packet loss.

Each variant is consistent, only in that they wait until receiving 3 duplicate-acknowledgements or retransmit time-

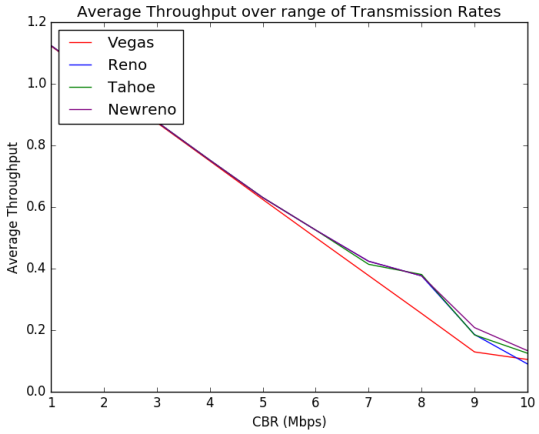

Figure 2: This figure plots the average throughput of each TCP variant over a range of CBR flow rates. The packet size is initialized to 3,000 bytes.

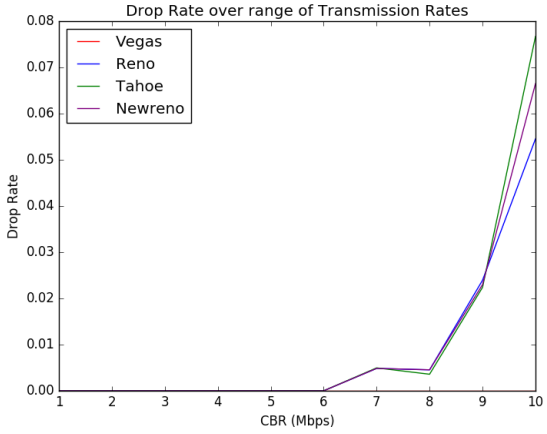

Figure 3: This figure plots the number of packets dropped by each TCP variant over a range of CBR flow rates. The packet size is initialized to 3,000 bytes.

outs before retransmitting the first lost packet.

Given low-to-moderate levels of congestion, TCP Vegas is able to efficiently decrease its congestion window size before experiencing packet loss, and thus is able to maintain zero packet loss while its counterparts are unable to. NewReno, handles multiple packet losses better than Reno and Tahoe. After the first packet loss is identified by a triple duplicate acknowledgement, or by a retransmission timeout, NewReno assumes that any packet following a partial acknowledgment is lost. This makes the retransmission of packets much faster and accounts for the increase in throughput experienced by NewReno in Figure 2.

4. EXPERIMENT 2: ANALYZING FAIRNESS BETWEEN TCP VARIANTS

This experiment used two TCP flows and a single CBR flow, to analyze the fairness between the TCP flows among different pairs of TCP variants. We tested different combinations of variants, including assigning TCP Vegas to both flows, TCP Reno to both flows, NewReno to one flow and Reno to another, and NewReno to one flow and Vegas to another. Further, we ran two slightly different simulations. The first simulation analyzes fairness in a network with a CBR flow rate of 1 Mbps and an initial packet size of 1000 bytes. The second simulation analyzes fairness is a network with a CBR flow rate of 10 Mbps and an initial packet size of 10,000 bytes. The first test simulates a network with little-to-no congestion, and the second test simulates a network with high congestion and packet loss.

4.1 Simulation 1: Little-to-No Network Congestion

In the network with no congestion, the throughputs of the individual streams given the combinations of Reno/Reno and NewReno/Reno were all equal, at .56 Mbps. This is to be expected. There were no packet losses, therefore NewReno stream acted as a Reno stream. This is because the variants only differ in what they do after multiple packet drops. After the first packet drop, both variants use fast retransmit and inflate their congestion windows to allow for outstanding packets between the source and destination to complete their transmissions.

The throughputs of the two Vegas streams, were .55 and .6 Mbps. This can be a result of one stream calculating a higher than expected RTT and reducing the congestion window, allowing the other to maintain and increase the cwnd size. Unlike Reno and NewReno, Vegas is not deterministic given no packet drops. The largest discrepancy between any two streams we tested, was that of NewReno and Vegas. The average throughput of NewReno was .98 Mbps, and the throughput of Vegas was .15 Mpbs. The average latencies of each pair of variants in this low congestion network, were very close.

TCP Vegas senses congestion through larger than expected RTT values, and proceeds to decrease its congestion window. TCP NewReno continuously increases its window size until loss actually occurs. Even though there were no packet drops and little to no congestion, the aggressive nature of TCP NewReno monopolizes the bandwidth of the streams, reducing the throughput of the more cautious TCP Vegas.

4.2 Simulation 2: High Network Congestion

Each combination of variants in the second simulation saw high amounts of congestion. This simulation consisted of a CBR flow with a rate of 10 Mbps and initialized all packet sizes to 10,000 bytes. Although the combination of TCP Vegas, once again resulted in similar throughputs for individual streams - at about .1 Mbps each, we note that these throughputs are much smaller than what was scene in the previous simulation. The throughput was much lower because each stream sensed the congestion caused by the other flow; however, there were no packet drops, instead the congestion window was continuously decreased. The combination of Reno/Reno also saw lower, but equal throughputs, with each stream maintaining an average throughput of about .2 Mbps. In this case, we saw 23 packet drops. Still, the fast recovery and fast retransmit process is the same for both of these streams, so the similar throughputs are to be expected.

The combination of NewReno/Reno resulted in a larger throughput discrepancy given the congested network. NewReno, as expected achieved the slightly higher throughput of .23 Mbps where as Reno's throughput was .185 Mbps. There were 30 total packets dropped. If the simulation had run for longer, we predict the number of dropped packets would increase and the discrepancy between the throughputs of the NewReno and Reno streams would grow even larger. FIgure 4 depicts the number of dropped packets for each variant when combined with another.

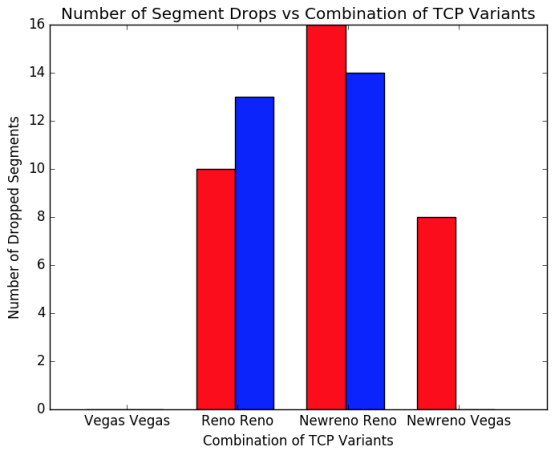

Figure 4: This figure depicts the number of segments dropped for each variant, when sharing bandwidth with another TCP stream.

Despite the fact that both Reno and NewReno inflate their congestion window size by the number of duplicate acknowledgments, Reno requires any additional packet losses to be detected by the retrasmission timeouts or 3 duplicate acknowledgments. NewReno retransmits lost packets more efficiently by assuming that the next partial acknowledgement marks a lost packet which is immediately retransmitted. The increased throughput of NewReno is a direct result of this assumption. Ultimately both Reno and NewReno deflate their congestion windows, but only after outstanding packets have been properly acknowledged by the receiver.

This discrepancy is more dramatic when assigning one stream to TCP NewReno and another to TCP Vegas. NewReno had an average throughput of .26 Mpbs and Vegas at .1 Mpbs. Only 8 packets were dropped in this simulation, all by the TCP NewReno stream. NewReno utilizes an increasing amount of the link before and in the immediate aftermath of a packet loss occur. Vegas, on the other hand, continuously decreases its congestion window as it perceives the increased utilization by NewReno, as congestion. However, we notice the discrepancy in throughputs here, is not as large as the discrepancy we saw in the first simulation. This is because the packets dropped by NewReno, although inflating the congestion window initially, ultimately deflated the window, reducing the size of the packets being released into the network. The throughputs of each stream on the shared network are depicted in Figure 5.

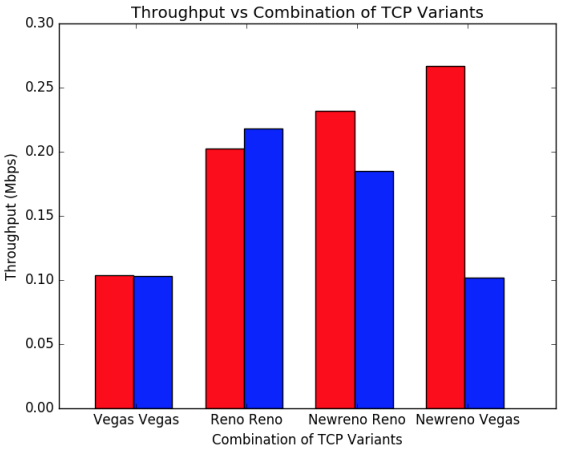

Figure 5: This figure depicts the throughput for each variant, when sharing bandwidth with another TCP stream.

5. EXPERIMENT 3: INFLUENCE OF QUEUING

This experiment analyzes the performance of TCP Reno and TCP SACK using the queuing algorithms: DropTail and RED. We ran these experiments for the buffer sizes of 100 and 300. When the CBR flow starts at the 4s mark, we see the throughput of the TCP stream start to drop for both RED and DropTail. There is a larger drop in throughput for RED compared to DropTail for both Reno and SACK. The combination of SACK and RED shows the largest drop in throughput. Figure 6 and Figure 7 plot these results for throughput.

The low throughput for SACK and RED combination can be explained by the inherent design of these two algorithms. In situations involving multiple packet loss, TCP SACK is aggressive about retransmitting the lost packets. SACK uses partial acks as a signal to wait until all outstanding, or yet to be acknowledged packets are received before coming out of fast recovery. Random Early Detection (RED) accepts all packets when the queue is below some threshhold, and randomly accepts packets when the queue is above the threshhold but still less then the maximum capacity. This allows it to reduce the power of large and fast packet burst which would otherwise monoplize the bandwidth and fill up the queue (in the case of DropTail). While used in combination with SACK, sending multiple lost packets per RTT results in the retransmitted packets being dropped. In such a sce-

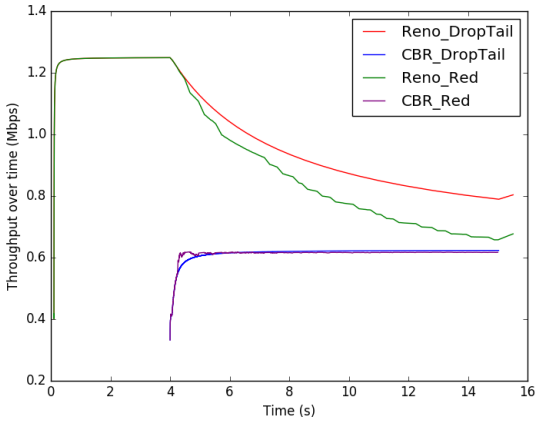

Figure 6: This graph plots throughput over time between CBR and TCP Reno stream for both RED and DropTail queuing algorithms. The buffer size is held at 300 packets.

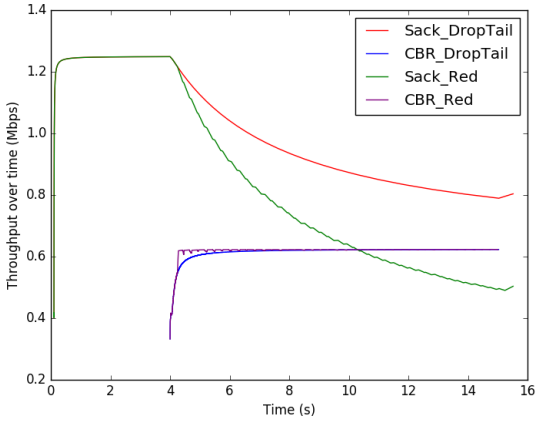

Figure 7: This graph plots throughput over time between CBR and TCP Sack stream for RED and DropTail queuing algorithms. The buffer size is held at 300 packets.

nario, SACK cannot avoid retransmission timeout (RTO). The combination of Reno and RED does slightly better because Reno only has the foresight to keep track of a single lost packet. It can only retransmit one lost packet per window so there is a smaller chance of RTO.

We also generated plots for end-to-end latency over time and the drop rates (8). After 4s mark, we see that DropTail has larger end-to-end latency and RED reduces the latency. While the latency of DropTail seems to be near constant, the latency of RED fluctuates. The latency of RED is significantly less in comparison to DropTail. However, this reduced latency is coming at a cost of throughput. The result for CBR and Reno was similar. DropTail has higher latency because it fills up the queue, and drops any further packets when the queue is full. Full queue results in larger end-to-end delays. RED on the other hand does not fill up the queue. It uses a threshold value to decide when to drop a packet. If there is congestion, far exceeding the threshold used in RED, packets are randomly dropped. How the

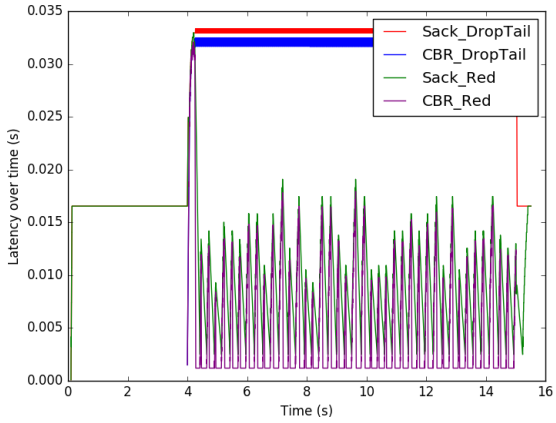

Figure 8: The graph shows latency over time for TCP SACK and CBR while using DropTail and RED queuing algorithms.

bandwidth was shared depended on the TCP variant and the queuing algorithm. With DropTail, TCP streams got a larger share of bandwidth compared to CBR which started a little late. Starting both flows at the same instant resulted in more proportionate share of the bandwidth. RED and Reno and RED and SACK lead to a fair share of bandwidth over time.

6. CONCLUSION

These experiments simulated the performance of different TCP variants in networks with varying degrees of congestion. We analyzed the independent throughputs, latencies and drop rates of TCPs Tahoe, Reno, NewReno and Vegas. Further, we analyzed the performance of combinations of TCP variants. The dramatic performance differences between combinations such as TCP Vegas and TCP Reno/NewReno emphasize how important it is to be aware of the different variants on a single network. As we saw, when specifying TCP Vegas to one stream and TCP NewReno to another stream on the same network, the throughput of Vegas was drained almost completely, as NewReno contin-

ued to increase congestion window, forcing Vegas to back down. This compromised the fairness of afforded to the streams. On the other hand, multiple Vegas streams or multiple Reno streams, maintained fairness in the network. The fairness also depended on when the two streams were started. Starting the less aggressive TCP variant early helps mitigate some of the unfairness.

Effects of queuing disciplines on Reno and SACK were also simulated. We found a combination of TCP variant and queuing algorithm (RED and SACK) that does not work well in terms of throughput. This was because of multiple RTOs. RED and DropTail presented a trade-off between throughput and latency. RED lowered the latency while lowering throughput. DropTail had higher latency but provided higher throughput as well.

These experiments stress the importance of simulating and analyzing the performance of a specific network under various levels of congestion. The performance, measured in terms of end-to-end latency, throughput and drop ratr depend heavily on the kinds of TCP variants deployed and the queuing algorithms. Simulation helps us assess if a specific network will achieve the desired performance metrics and provide fairness. Choosing an inefficient TCP variant, an unfair combination of TCP variants or a queuing algorithm which does not meet the needs of the type of data exiting the network, can be detrimental to a network. These network configurations must be carefully analyzed whether it be in a corporate, government or academic network. Achieving a fair share of bandwidth in an environment where different TCP variants and transport protocols are used is difficult.

On a final note, our simulations showed that TCP NewReno maintained high throughput on its own as well as when combined with other variants. However, it would be interesting to extend these tests to further study situations in which the immediate retransmision of packets caused by partial acknowledgments were only adding congestion to a network. It's possible that partial acknowledgments do not signify packet loss. In this case retransmission is not necessary and is only adding more congestion to a network.

7. REFERENCES

[1] R. Kohavi and R. Longbotham. Online Experiments: Lessons Learned. *Computer*, 40(9):103–105, Sept. 2007.