CSCI 4310/6310 Project 2

Project 2 TCP Slow Start Threshold Scaling Option

Full project due Wednesday December 1st, 11:59:59PM.

In this project you will implement a new TCP option that supports modifying the TCP Reno slow start behavior.

TCP Slow Start Threshold Scaling

You will implement a new TCP option in the kernel that should be on for all TCP Reno connections. It will use the option kind `253` (Experimental TCP Option), and will consist of the option kind (253), option length, and a 2 byte integer that is the "slow start threshold scaling factor", which we will call tcp_ssthresh_scale.

If the option is observed by a socket on receipt of a SYN or SYN ACK packet, the socket should use the provided tcp_ssthresh_scale for the remainder of the connection. This number can be any value between 1 and 100. If the socket is using TCP Reno as the congestion control algorithm when it receives this option, then any time a loss event happens that would cause the slow start threshold to be `1/2 * the current congestion window`, it should instead be adjusted to be `(1/(0.1 * tcp_ssthresh_scale)) * the current congestion window`.

If during the handshake you do not receive a Slow Start Threshold Scaling option, you should assume tcp_ssthresh_scale = 20 (i.e. the window will be cut in half if no option is provided).

To verify this behavior, whenever the slow start threshold is updated, print a message to your log with the following information: "KERNET src: [source port] dst: [destination port] old: [ssthresh before update] new: [ssthresh after update]" where the values in brackets are variables you should populate.

You may want to set up port forwarding from your host OS to guest OS so that you can have your kernel only be the sender or receiver instead of always seeing this effect as both the sender and receiver.

Experiment and Deliverables

Provide one or more .pcapng traces (made by Wireshark or tcpdump) and the relevant portions of your kernel log (in one or more .txt files) that show your system working with and without the option enabled at loss rates of 0%, 10%, and 50%. For each of these loss rates, also experiment with different values of the slow start threshold scaling factor.

Your write up should be at least 2 pages double-spaced (or 500 words) *per group member*. Code segments, screenshots, etc. do not count towards this length. Describe any issues you had when developing, any interesting code discoveries you made as you went through this project, any mechanism you used to change your test numbers (i.e. did you hard code them in? Make a new system call that looks like connect()? Use a sysctl variable? Kconfig option? Something else?), and how a user that installed your patch could change the slow start threshold scaling numbers as well (you can assume they are willing to recompile the kernel every time).

Also discuss your experimental procedure and your findings about the scaling factor (a figure or two is probably very helpful here) - did the factor impact performance? Was there an optimal value? What kind of

test did you use, and what was the time scale and amount of data transfered? The exploration is open-ended so as long as you come up with some design and results (even if they're "the factor doesn't seem to matter at all"), you should receive full credit for the "Exploration" points.

Tips

1. You might end up doing *printk* debugging fairly often. Be careful about overflowing the log buffer by printing too much too fast.
2. You may want to disable the check for updates (in Ubuntu with Gnome open Software & Updates, go to the Updates tab, and set "Automatically check for updates" to "Never") - this will help prevent TCP calls you didn't expect.
3. You may want to pad your options with *NOP* to keep it 32-bit aligned.
4. Try to get options attached and parsed before worrying about how to store the state and make congestion window updates.
5. Remember to leave yourself enough time to either design a way to easily change parameters, or set aside a *LOT* of time for doing a ton of rebuilds and restarts during data collection.

Useful Commands (requires NETEM enabled in .config)

To drop 10% of packets on loopback: `sudo tc qdisc add dev lo root netem loss 10%`

To view the rules on loopback: `sudo tc qdisc show dev lo`

To modify the rule (assuming you didn't delete it) to drop 50% of packets on loopback:
`sudo tc qdisc change dev lo root netem loss 50%`

To remove the rule: `sudo tc qdisc delete dev lo root`

If you want to try this on a different device, such as your "internet" use "eth0" instead of "lo" for the device ("dev") argument.

To check what current congestion control algorithm is: `sysctl -a | grep  net.ipv4.tcp_congestion_control`

To set it to Reno: `sudo sysctl -w net.ipv4.tcp_congestion_control=reno`

Project 2 Submission to Submitty:

1. **report.pdf** is the full report as described above. Do not put your names on the report.
2. **patch.diff** is your patch file for your full submission. Do not put your name in the code or *EXTRAVERSION* of the **Makefile**, but do pick an *EXTRAVERSION* that is likely to be unique such as a dictionary word. "-project2" is probably not going to be unique.
3. **README_Instructor.pdf** should include a final division of labor (who did what, how much time everyone spent). Ideally this division should be approximately equal. It should also include any notes that you want the instructor to read that you wouldn't want a peer reviewer to read.
4. Any optional files, including any Wireshark traces. You should submit at least one Wireshark-readable trace (.pcapng) demonstrating your option in action.

Grading

- 10 points for implementation
- 5 points for documentation and development discussion
- 15 points for experimentation description and discussion of results
- 20 points for writing quality