Congestion Control
For Interactive Real-Time Communication

The IAB and IAB/IRTF Workshop

July 28, 2012

=== Welcome and Logistics (Workshop organizers) ===

Alissa Cooper and Spencer Dawkins have been dragooned as note-takers.

This workshop was held under the IETF/IRTF “note well” IPR policy.

We said we would attribute statements in minutes and in the workshop report.

=== Keynote (Mark Handley) ===

= The intention of this keynote talk was to provide background for later discussions.

= What is congestion control about?

Bulk transfer. Implication: if network isn't congested somewhere, something is broken. Congestion is normal when you're doing bulk transfer, as long as you avoid congestion collapse and achieve some kind of fairness.

Lots of problems have already been fixed – modern TCP congestion control behavior, path MTU discovery, etc.

= Primary goals of congestion control

1. Avoid congestion collapse. Network must work. This is the most important thing.

2. Get some sort of fairness. All users get some service.

= Goal 1: Avoiding congestion collapse

= Congestion behavior

Want goodput to saturate at level network can cope with. This is the goal. Unfortunately you often hit some threshold at which network becomes overloaded and goodput goes down. Congestive collapse.

= Why does this happen?

1. Classical problem: TCP gets into unnecessary retransmissions.

This was fixed in late 80s by adjusting TCP retransmission timers and adding congestion control.

2. Fragmentation. Loss of fragments leads to loss of packets. Fragmentation can be a huge loss multiplier.

This was fixed by adding MTU discovery and early packet discard (in ATM networks).

3. Paths clogged with packets that are discarded before they reach the receiver.

Example: UDP flows blasting at higher rate than TCP flows, downstream link drops all the packets it can't buffer. You have a busy network with zero useful throughput. There are two ways of dealing with congestion:

a. end-to-end congestion control

b. virtual-circuit style guarantees

We don't want to do (b) so we need to do (a).

TCP congestion control has functioned well since 1988.

= Goal 2: Fairness

TCP does fairness by having a window of packets that have been sent but not acknowledged, and adjusting the window size according to simple rules. Each RTT, the sender increases window size by 1 packet; if a packet is lost, halve the window size and start increasing again.

= Why does TCP achieve a sort of fairness?

TCP achieves fairness over a reasonably long period of time. Two flows trying to increase their window sizes at the same time, will both back off when they encounter losses, and through successive iterations of losses, the two flows even out. Over a series of successive losses, you get approximate fairness between two flows, if they both have data to transmit. No particular fairness at any specific instance in time, but some sort of fairness over reasonably long period of time.

= What throughput does TCP give you?

avg sending rate T = sqrt(3/2)*B / R * sqrt(p) where R is RTT in secs, B is packet size in bytes, p is packet drop rate

Proportional to 1/RTT, 1/sqrt(loss rate)

= Is TCP's fairness what we want?

Probably not, if we could design it over. 1/RTT is not ideal relation. ISPs can deliver low loss by adding big buffers. This is really not a bonus and Jim Gettys has pointed this out. If you add buffers, the RTT goes up because packets also spend time in buffers on the way to the receiver. With intermediate buffers, loss rate goes down but you achieve same throughput.

1/sqrt(p) is not ideal either, especially. for high-speed flows.

So this isn't ideal, but it's what's out there for TCP and it avoids starvation.

= So what are we precisely controlling?

In TCP, we control the number of packets per second sent. This doesn't work for real-time media.

= Video congestion control

30fps x 1500 bytes = 260 kbps

Above the 1500-byte frame size, you send more than one packet per frame; below this rate, what to do? Send smaller packets?

= Where's the bottleneck?

In a wired environment, the bottleneck is in bps. It doesn't matter how many packets/sec, as long as you include headers in calculation.

In a WiFi environment, the MAC acquisition delay dominates. You want to reduce packets/sec to relieve congestion. Sending smaller packets doesn't work, because the MAC acquisition delay is constant for any packet size.

But sending larger packets is in direct contradiction to what you want to do for video. Application does not want to buffer several frames together before you can send them, because it’s trying to minimize delays. Buffering frames before you send them increases delay.

= Minimal primary goals for congestion control (from network point of view)

1. Avoid congestion control

2. Avoid starvation – for both TCP flows and real-time flows, sharing same FIFO queue

= Question from Eric Rescorla on Jabber: Not clear that we want to avoid starvation of TCP flows. If I'm in a video conference, not clear that I don't want to starve TCP. If you have bandwidth X, and your video is unacceptable at anything less than 0.9X and your TCP is unacceptable at anything less than 0.5X, figuring out what you do has to be a policy question before it’s a congestion control question.

Mark: That's in your case, but in a corporate IT case, IT may want to make sure TCP is not starved. You may be in control of your network, but you may not want to starve TCP on the network.

Eric Rescorla: But the notion of not starving TCP flows is wrong.

Mark: Didn't say you have to be fair, I said you can't starve them.

= Goals from application point of view

1. Want the application to be robust. Want to keep working no matter what.
2. Predictable behavior.

3. Low latency.

= Robust behavior

Good quality when network is working well. Application still works when network is working poorly. In either case, loss is low enough for session to still be useful.

= Predictable behavior

Variable quality in multimedia is bad for users. Users rate sessions at the minimum quality of the entire session. What you like for application is in tension with what you might want from network congestion control.

= Low latency

If you're sharing with TCP on congested link, good luck. Bufferbloat gives unwanted latency.

Delay-based congestion control can keep latency low. But if you're competing with TCP in the same queue, you'll still be out of luck. Real solution here is to fix bufferbloat: Weighted Fair Queuing (WFQ), with a good ISP traffic shaper that shapes to 95% of DSL sync rate, etc. But these are unusual on the Internet today. Applications today tend to have to deal with whatever they're given.

= Streaming vs. Interactive media

Streaming: can use buffering to smooth out. Streaming over TCP is common, although that’s not a good answer. Doing it over UDP is probably better.

Interactive: Can't afford to buffer, or latency will kill interactivity. It’s important to separate these two uses of media (streaming and interactive). We're talking here about interactive media.

= TCP-Friendly Rate Control (TFRC)

TFRC was the result of a bunch of work ten years ago, and it illustrates a bunch of problems

Goal: Same throughput as TCP but smoother transmit rate.

Measures loss, RTT and uses TCP to model sending rate.

= TFRC problem

If you've got a link with low statistical multiplexing, you can get bad oscillations. You hit the maximum rate of a bottleneck link, get a lot of loss, and then peak again. Very small buffers do okay, but anything bigger gives oscillations. That’s bad for both network and applications.

Solution: Throughput adapted on short time scales based on RTT. Has built-in delay-based adaptation mechanism to avoid these oscillations.

Requires good short-term RTT measurements.

= TFRC doesn't work well with real applications

TFRC assumes network is in charge of the codec. Assumes congestion control clocks packets onto network. Assumes codec can produce data at a range of demanded rates. This is probably not what real applications want. It's probably fine for streaming, but probably not good for interactive media.

= Video is inherently variable

Context sensitivity: motion, scene changes, etc.

MPEG-2: You have a variety of frame times (p-frame, i-frame, b-frame), all different sizes.

It’s hard to produce data at an exact rate w/o buffering or ugly quality changes.

TFRC is a bad match for interactive video.

= What if codec is in charge?

Network tells codec mean rate to send, codec matches mean rate, but not on short timescales.

This leads back to low statmux problem. You get oscillation like TFRC.

So inherently the codec and the network would like to be in charge, but they can't both be in charge.

= How to avoid low statmux problem: linkspeed characterization

Van Jacobson's pathchar and follow-on work

You send short packet trains, measure timing accurately, and infer linkspeed from relative delay. If you know linkspeed, you can avoid exceeding it. Congestion control can give you an approximate rate, but you must not exceed linkspeed. This is a hybrid between codec being in charge (most of the time), and network being in charge.

These work well for some links and not others. E.g., WiFi not good. Fading, bitrate adaption, etc. – wireless link speed can change in less than a single RTT

Fair queuing can also have interesting interactions with linkspeed characterization.

So this is not a magic bullet, but another technique you might want to bring in to congestion control.

= RTP

Not sure if RTP (with RTCP) is good for congestion control.

RTCP give feedback, but can't send it often enough to try to avoid bumping into linkspeed.

RTCP probably doesn't give us what we like, but we shouldn't be limited by it. We wrote RTCP, and we can revise it if we know what we want out of it.

= Circuit breakers

If the sender measures loss rate and RTT, and runs with a loose “cap”, and the sending rate isn't too high for too long, no problem. Otherwise, you need a circuit breaker that says “stop”. You avoid congestion collapse, but circuit breakers don’t help you get good performance on an UNCONGESTED path. We want more than just avoiding collapse.

= Summary

We know how to do good congestion control, but only if congestion control is in charge, and that's not acceptable for real-time applications.

Only know how to do good congestion control if we change packet/sec rate and not packet size.

There are many facets here, and we need to keep application requirements in mind (no buffering). This is the hard part: video is inherently variable. It can take advice from congestion control about how fast it should go, but we don't yet have a compromise between what the network would like and what congestion control would like.

= Q from Keith Winstein: Have heard that TFRC tries to adapt slower than TCP. Do we need different solutions for cases with different [linkspeeds?]

Mark: TFRC adaptation is definitely slower than what TCP does. It's a compromise -- you don't want TCP behavior if you're trying to fit video into that envelope. Is it too slow for links that vary fast? Even TCP can't cope with congestion that varies on sub-RTT time scales. We don't know how to solve that problem.

= Q from Bob Briscoe: Clarification for the room -- you were talking about Reno TCP, which is a smaller proportion of TCP these days. Cubic and Compound are much more aggressive than Reno, particularly as you start moving to faster rates.

Mark: Not trying to say that model is a good model, but overall goal of trying not to starve TCP flows is what we were doing with TFRC.

Bob: Not meaning that you should have covered those, just that we have another problem now that we have new congestion control mechanisms that are more aggressive. Real-time flows have a problem with the more aggressive flows.

Mark: Compound is substantially more aggressive only when there's a lot more capacity in the network. Cubic is more aggressive across the board.

= Q from Cullen: You said "good luck" if you want interactive competing with TCP. Can you say more about your thoughts about how much we could have those in separate queues on the Internet?

Mark: If you're in the same drop-tail queue as a TCP flow, you're going to get the same latency as TCP in that queue. But that’s not inherent - you could be in different queues, e.g., with Weighted Fair Queuing.

I don't know how prevalent that is out on the Internet. My home Technicolor gateway does Weighted Fair Queuing and it's obviously doing deep packet inspection (DPI) to figure out which flows go in a higher priority queue. My VOIP calls are not affected by large uploads as a result. Some form of Fair Queuing would be good for avoiding that, but the question is how to avoid other traffic putting itself in high priority queues? Multiple queues work if people don't try to game the system. Fair Queuing would be a substantial improvement to the status quo. But it's not a panacea and you can't assume that you have it. We would be wise to encourage it.

=== One control to rule them all (Michael Welzl) ===

Defending a congestion manager-like thing.

How we use the Internet today

We have gotten used to the horrible service that the Internet provides today. Downloading while listening to Spotify and the streaming stops. To use Skype video, must stop doing everything else. Two different downloads may have different priorities that I can't control.

This is about performance. Example:

I wanted to create a congestion manager. Fairness between two TCP transfers -- one short transfer starting in slow start after the transfer has already started. Was using large default queue parameter in Linux. So queue had already grown when second flow was trying to start. Behavior to user looks poor -- 1MB file takes 4.5 seconds and 64MB file takes 6 seconds. Scheduling them both over the same transfer gives much more reasonable performance. MY main argument in favor of congestion manager is this more reasonable performance.

Doing congestion manager in general is very hard. Only want to do this when we cross a shared bottleneck. But this might not even be the case with same destination IPs because of ECMP. And may have overly aggressive congestion control -- browsers opening multiple transfers, e.g..

But we could do this right for RTCWEB. We're assuming everything is going across one 5-tuple.

Ted Hardie: Many applications will have more than one party. So with multi-party, there will be >1 5-tuple.

Michael Welzl: This assumes one party.

Ted Hardie: That does not match RTCWEB's use cases.

Michael Welzl: But this will work at least for the 1-party case. This is an argument for combined congestion control of RTP and SCTP.

Ted Hardie: In paper sounded like presuming only 2 people on the link.

Michael Welzl: Idea is to apply this only in cases where you can correctly assume that packets go between two users. In multi-party case you apply it pair-wise.

Bernard Aboba: Single controller for any two parties is what he's talking about.

Ted Hardie: Agree but in a lot of cases you may have congestion between two parties in an application context but not between one party and some other party. You can't assume that you have congestion on same link every time. With 4 people playing poker, I might have congestion on link to Mary.

Michael Welzl: Ok, 4 people playing poker. One congestion controller from me to you, from me to him, from me to her.

Ted Hardie: Ok. Still does not solve all of RTCWEB's problems.

Michael Welzl: True.

Lots of benefits of congestion manager.

1. Control fairness between streams when you can schedule between them. Outcome is not streams fighting at bottleneck, but you scheduling them.
2. Reduce queuing delay because flows aren't fighting at bottleneck.
3. Short flows can benefit because they can be mapped onto longer transfers. Skip slow start.
4. Less feedback will be needed. If you have RTP and SCTP, you get RTP ACKs all the time, but you're ignoring feedback that you get from SCTP about the same link.

Randall Jesup: I like this idea, but it doesn't scale itself across more than one 5-tuple. On the web, bottlenecks tend to be in access, on one end or other or both. Finding ways to determine if bottleneck is local or far end or some place in between, that would be useful in extending this across more than a 5-tuple, even if it's not guaranteed to be correct. Knowledge of flows outgoing or incoming to your machine on your access link would help.

Michael Welzl: Agree. Shared bottleneck detection research looks good. But IETF has a history of not standardizing them. We could look into it.

Harald Alvestrand: IETF has a history of not standardizing lots of things. I was confused by your presentation because your problem statement was unrelated to your solution. None of your three bad stories at the beginning will be impacted by your proposed solution. With one 5-tuple (actually a 6-tuple, because you need a DSCP codepoint), that can be a building block for solving the problems in your stories.

Magnus Westerlund: Trying to expand to multi-party case, especially with different bottlenecks, is important to consider seriously. It might not be easy to solve but should at least be considered.

Eric Rescorla: I don't know why people think we can solve this.

Bill Ver Steeg: Don't boil the ocean. If you know what last mile bandwidth is -- coffee shop, airplane, and access -- even knowing broad granularity would help a lot.

=== Data (Cullen Jennings) ===

== Keith Winstein, paper 28 ==

Measurements of Verizon LTE network. Computer sends packets out of Verizon LTE network and back to same computer. Precise measures of one-way delay.

Findings: almost infinite queue. No drops until 5 sec delay at least. Have seen end-to-end delay of 40 sec. This isn't just LTE - 3G is the same, BTW.

Poisson distribution of packet arrivals. Measuring rate is not easy and it varies by time. By 10x factor in short time scale, so RTCP jitter estimate is bad. Highly long-tailed delays.

Bill Ver Steeg: Do we know radio induced delay separate from network delay?

Keith: Know that it produces a Poisson distribution of packet arrivals.

Magnus: Bounded by 100s of ms. This happens when the network buffers for a user to see if he moves out of bad cell.

Keith: Operators do not yet believe that this is true. We presented this at ICCRG, and they did not believe that they would hold a packet for 5 sec. They're the fastest on CNET.com.

Jim Gettys: Have back channel info that queues may be in backhaul too.

Michael Welzl: Not clear where queues are.

Supposition is that with TCP buffer is always full. But if an e2e protocol said I'd like to have a max 100ms buffer, how much throughput do we forfeit? It's possible for a protocol to cap e2e delay while sacrificing some throughput.

Magnus: I don't think it's transport; it's trying to keep capacity in the cell.

Keith: Verizon is not changing this any time soon in their network. This probably has to be solved e2e.

Jari: That's not completely correct. Some of your points are fundamental -- delay varies because of radio. Others are configurable, and could be changed if we could convince people to change them. In my lab people are trying to understand how queuing impact radio network traffic and they push out solutions in our products.

Keith: In case where queues don't drop packets, the problem is unpredictability of future rate. IF you have no uncertainty, you can send at a rate where you won't encounter a queue. The variability is what makes you sacrifice throughput.

Bob Briscoe: So really message is that RTCP needs more measurements to describe probability distribution?

Keith: Agree.

== Zahed: ECN (paper 4) ==

How effective is ECN on LTE?

Experiment: simulation of real-time voice

Compared ECN-based adaptation with implicit methods like packet loss and delay.

LTE “enhanced Node B” (eNB) with ECN-based adaption does better than implicit methods. ECN can detect congestion before it happens.

Eric Rescorla: What fraction of routers does ECN marking and what fraction of OSes allow applications to see ECN markings?

Bernard: There is a paper on the agenda about this.

Keith: Linux lets you read ECN from user land.

Jahed: If ECN is good, it should be promoted.

Magnus: At least for LTE it is specified that it should be supported.

Mo Zahaty: We need to make sure that the solution can handle the case where ECN is deployed. Need to give incentive for operators to turn it on. Every single network router supports it.

Gettys: But home routers do not, broadband gateways do not.

Mo: Those are fast upgrade cycles.

Room: No.

Welzl: A lot of routers will clear ECN. I think we should use ECN on RTP.

Gettys: Bauer, Beverly paper from last year: There are unfortunate problems beyond just clearing the bits. Three years ago OpenWRT tried to turn on ECN by default in their home routers because there were too many blackholes they couldn't talk to as a result. Don't think we have information about whether we can turn it on globally. That doesn't mean that it isn't useful for LTE and 3G talking to data centers. Home networks are different.

Bob: They couldn't turn on ECN in their routers because they couldn't contact someone? Don't understand.

Gettys: Let's take it offline.

== Markku, Paper 9 ==

How is audio affected by concurrent TCP flows?

With one concurrent TCP flow, voice is impacted on startup. Six TCP flows destroy quality of call.

Drops are due to jitter, not losses.

Initial Window size = 10 (IW10) makes this problem significantly worse

Bernard: Was this with IW10?

Markku: Graphs on slide are with Initial Window size = 3 (IW3). Another graph in paper compares IW10 with IW3.

Bob: In paper you said it seemed to reduce the quality long after the IW had gone away?

Markku: Did not look closely, because those flows end up opening more windows, and there are no losses. You open more windows until your payload runs out. That just increases the queuing delay.

Bob: Wondered whether it's because the bursts of packets don't pace out so soon, so they last longer.

Markku: Good question.

== Mo Zahaty, paper 25: TFRC ==

Fundamental: Rate that TCP achieves and therefore what you get with TFRC, if you use it as a cap rather than a floor --

Focusing on CLUE as well as RTCWEB (multimegabit flows). Not much room to achieve those with strict TFRC. This results from the rate equation -- it's trying to simulate window-based control and that might not be the right thing for interactive multimedia flows that are not acknowledged.

Newer work on MulTFRC. Tries to tune the equation to cases with multiple flows. With 4 flows, which would be sort of competitive with what web browsers launch on initial browser fetch, performance is better, high-bandwidth CLUE applications are more likely.

We need some kind of application-weighted fairness so application can determine request to congestion controller. Flow-rate fairness won't work for decent media quality.

== Cullen, paper 11 ==

Things go badly wrong when TCP shares with RT. Video performance degrades when sharing with TCP -- that is not "fair." Things don’t get better after adaption – things get worse.

Adaption takes time. With coexisting flows, they take a long time to adapt to a situation where video performance is bad.

We need to figure this out because people need to deploy something that is going to work. Video adapts very slowly because you want to keep the video smooth.

Randall: Are those simulation results?

Cullen: They are real applications running over a simulated link with 250ms latency and .5Mbps bandwidth. All the results are highly sensitive to changes in those parameters.

Eric Rescorla: Does TCP take over empty link as quickly as a full link?

Cullen: How quickly this happened was pretty much the same in both cases, but that's a vague memory.

Randall: Did you look at startup performance? Some video congestion control algorithms are more aggressive at the start, and then they slow down.

Cullen: Very hard to run experiments to figure this out. Guess is that it's not only that they're more aggressive at the start but at keeping that bandwidth even when conditions change.

Colin Perkins: Should try with multiple TCP flows.

Harald: Encouraged by the results because all of the products looked like they were trying to make it work rather than just grab all the bandwidth they could.

Gettys: Netflix runs a TCP connection as fast as it can (bursting an entire congestion window of packets at line rate) and then goes idle for 10 seconds. Will have TCP flows that go as fast as they can and then go on and off. Particularly entertaining with bufferbloat.

Keith: Do you see bitrate jump in discrete jumps or is it smoother?

Cullen: We saw fairly smooth -- jumps of less than 30% and more than 5%. With Netflix/iTunes/Dash, story is very different. We didn't see doublings.

Keith: Did you measure queuing delay incurred?

Cullen: No. We were just looking at vague interaction.

Randall: How big was obvious difference in upward versus downward adaption?

Cullen: changing parameters of the experiment changes the results a lot, so no great answers on this. None of the applications did additive increase and multiplicative decrease.

== Vincent Roca, FEC, paper 30 ==

Took video encoded at higher and lower bitrate. Added redundancy to lower bitrate version so that bandwidth between both was equal.

Quality of unprotected video was lower and more variable than protected video. So it makes sense to reduce video transmission rate to add some FEC encoding.

Where should we put redundancy? There are several techniques for this.

How should we put in redundancy? It's important to remember that not all video frames are equally valuable (i-frames, etc.). It's important to remember that loss rates aren't random in the real world, when we're simulating.

Did some other experiments to answer the questions above. There seems to be no clear winner as to where redundancy should be -- depends on the video. But it seems more appropriate to use FEC encoding than to put encoding in video coding part (? Both note-takers are confused here?).

Block FEC codes -- sliding encoding window for packets not acknowledged. We used Tetrys. It makes sense to do this.

Randall: To keep in mind when we talk about simulations: It's easy to dial in a loss rate on a simulator. Loss rate is usually from tail drop, very rarely random loss that we see in the wild.

Vincent: True. We consider that in any situation there will be loss from time to time. We did use fixed rate of loss. So it makes sense how to recover from random losses.

Randall: The without-detection graph seems to imply fixed i-frames that are far apart; therefore quality goes down over long interval. No active repair on RTT+buffer time frames, which is what real interactive video applications do.

Stuart Cheshire: Were you modeling interactive video or streaming video? As far as I know, FEC always adds delay to encoding pipeline because the way you recover an error is to have packets before and after it. For streaming video that's fine, for real-time interactive, how much extra delay does this add?

Vincent: There are several ways to do FEC encoding. One is to accumulate source packets until you have K packets and then you do FEC and continue to transmit. The larger the window, the better the correction but also the larger the delay. For on-the-fly encoding, the coding delay is only dependent on coding rate, not on block size.

Stuart: Was your video interactive or streaming? Were the differences visible to users?

Vincent: I don't remember the conditions. Yes, visible to users. There was some extra delay.

Stephen Botzko: We use a proprietary block encoding scheme in our products. You can achieve FEC block size of 100ms for a 500kbps connection. To some degree the buffer delay we need to handle jitter anyway covers the block delay. This could be part of the solution but not a replacement for congestion control.

Vincent: Errors will happen anyway.

== Stefan Holmer, RRTCC, paper 6 ==

Simulation of two flows (A and B) sharing links and flow C with constant rate. Flow A is RRTCC controlled. Links they share are 2Mbps. Flow B is also RRTCC flow.

RRTCC looks at inter-arrival times to infer congestion. Because flow A is seen as noisier (more jittery) than flow B, we filter the inter-arrival time of flow A more than B.

Since there is constant threshold for deciding whether we're overusing the link, the flow that is filtered less has a faster reaction time and will back off faster. Flow B loses against flow A.

Most interesting is the estimated jitter variance -- the noisier flow looks more noisy over time and therefore gets filtered more. It also detects overuse more rapidly. But over time the noisiness converges because the noisier flow has taken over the bigger part of the pipe – noisier flows end up with a permanent advantage.

Possible solution: constant amount of filtering no matter how noisy different flows are. That means dropping an interesting part of the noise-adaptive filter. That does solve the problem though. This is what most delay-based congestion control algorithms do today, they aren't jitter-adaptive.

Randall: What sort of results do other congestion mechanisms show across the same topology? What happens when TCP is competing with itself over that topology? Absolute fairness isn't required. TCP doesn't get that.

Stefan: For TCP, the long time average will reach something similar to what we found when we took out noise adaptation. For TCP if you have one link with random losses and the other one doesn't, the first one will back off due to non-congestion related losses.

Randall: How often are random losses actually occurring on the links we're looking at? Most losses are in some way or another congestion-related. Are there real random losses?

Bill Ver Steeg: In enterprise you're correct – no non-congestion losses. In service provider network, that’s not true. Must be careful about these assumptions.

Bill Ver Steeg: A lot of simulations assume FTP cross traffic.

=== Constraints (Hannes Tschofenig) ===

== Jim Gettys

Latency budget is always overdrawn at every location and every layer. Can't give away time unnecessarily anywhere. No one will ever say their latency was too low, and gamers would love to see >5 ms latency to avoid getting “fragged”.

I worked on coast-to-coast videoconferencing 17 years ago, and we may be worse off now than we were when I started.

Hannes: For interactive gaming, the servers will pick people who are near you if you don't care who you're shooting. And they do some path prediction.

Gettys: We have huge amounts of jitter in the network now. Up to 1 second.

So packet trains leave a data center, they arrive at edge of network, they become very long, and HOL blocking creates latency spikes. Equipment could have multiple queues, but customers don't see more than one queue on most paths, so you're stuck.

Where the worst bottleneck is varies. In the edge there are at least two -- WiFi link and broadband link -- and it's often dynamically changing back and forth. Need to work at breaking up huge packet trains.

Causes are smart hardware, web and web sites that cause 4-12 TCP connections to be opened at once, and ACK merging in broadband gear that destroys ACK clocking.

“Smart hardware” isn't – it assumes that bandwidth is the only thing that matters, not latency.

Get transient latency at 100-200ms -- it's not just steady state, it's also transient.

Buffers are dimensioned incorrectly and you can't even determine what correct might be. Many devices and people sharing one link in a house can fill queues for everyone in the house. Uplink tends to be worse -- buffers are same size in both directions.

We don't have tools to point fingers at where the problems are. You can yell at your kids to stop downloading large files when you're on Skype.

These problems can occur anywhere, but ALL broadband gear has these problems. Netalyzer data shows buffer problems across cable, fiber, DSL. "We're screwed, guys."

Cullen: What you're describing is going to impact voice as much or more than video. So however bad this is for Skype, voice actually seems to be oddly working pretty well if "the Internet is broken."

Gettys: Until you run one bulk data transfer.

Cullen: WebEx does that all the time.

Room: People police themselves -- they stop their downloads.

Cullen: But Windows and Apple users are always downloading stuff in the background.

Stuart: We use LEDBAT for that stuff. Uploading is much worse -- uploading to YouTube. But even for downloading it was a problem.

Gettys: I did a simple test with YouTube -- it makes Google Search run 16x slower.

Keith: Agree with Jim's message, but I'm more optimistic. Efforts of bufferbloat community have been laudable but they haven't reached major American network providers.

Gettys: DOCSIS change ratified last year that allowed operator to change buffer size. So that will roll out this year or next year and will help if people replace their modems. But COMCAST seems to be the only cable company that understands this.

Keith: Verizon and Hughes don't know this.

Gettys: Operators look at Speedtest.net, and that doesn't measure round-trip latency on a loaded link. We need ways to point the finger at where the problem is.

Bob: There is some hope here. We have swayed OFCOM and the FCC that the comparisons they do between ISPs should include latency under load.

Gettys: We got one test into the US measurement but people aren't aware that this is something they should be beating up their operators about.

Gettys: I've talked to the Speedtest.net founder about getting this added but they're doing a major site upgrade now.

Lars: We're not at the bufferbloat workshop. Is it hopeless to do congestion control for real-time now?

Gettys: No, because then we serialize the problem.

Lars: So the question is what can we do now that will help now and fix problems when bufferbloat also gets fixed?

Stuart: We need to get some kind of Fair Queuing into as many routers as we can. Without that, all of the good congestion control we do, however well my client behaves, there's nothing I can do when I'm at the back of a buffer. Everything we do here needs to be paired with segregated queuing.

== Matt Mathis

Agree. Throughput-maximizing protocols require segregated queuing. TCP-friendly is an oxymoron. At all major bottlenecks you need to have some way of not scheduling real-time traffic behind bulk and transactional traffic.

If you use TCP for doing streaming and there are large buffers in the network, you always send a chunk at full line rate. So for essentially all video streaming you get 64k chunks and that’s 42 full-length packets, back to back at line rate, on every connection, every few seconds. There are parts of the network where you see this all the time. No equipment can retransmit that without buffering it. It takes a lot of complexity in the application to fix this. The only real solution is that throughput-maximizing and delay-minimizing protocols need to be in separate queues.

“TCP-friendly” is a misnomer.

Ted Hardie: If you have a playout buffer in the application itself, managing that buffer is easier than trying to ensure that packets arrive at exactly the right time. With a playout buffer, as long as it's ahead of what I'm watching, I don't care.

Matt: It's easier to say "I want the next chunk of the file."

Ted: Huge problem when you use HTTP for this.

Matt: It doesn't matter what protocol you use for this.

Ted: We need to distinguish this from interactive video – that’s completely different.

Bob: Once you've got fair queuing in a queue, it's very difficult to remove it. If interactive video is scaling its rate according to what is going on in the world, and it's sharing a queue with TCP, it's going to have a variable rate . . . Dumb fair queuing that we currently would have if we recommended this is going to come get us in the future.

Gettys: This is why I put "fair" in quotes because “fair” is in the eye of the beholder. It differs depending where you are.

Bob: If we recommend "fair" queuing, no one is going to notice the quotes.

Matt: That's why I say segregated queuing. With different AQMs, the scheduler might be complicated.

Gettys: Need a better name than fair.

Cullen: Of the many ways available, how can we separate traffic into 2 queues?

Magnus: NSIS (hilarity ensued). DPI.

== Hannes summarizing John Leslie

How do you combine different requirements between different layers? Semantic is simple at application layer but needs to be combined in a meaningful way to lower layers.

Other aspects: Cullen had hinted at what vendors actually will do, some other people discussed business models and how operators won't implement anything unless they can charge for it, instead of just giving it away to OTT providers.

Matt: What we need to have built into these protocols is instrumentation about data that arrives out of window. If you're having a bad day, you can click a panel that tells you how bad your network sucks and complain to your ISP. This will cause market scope.

Hannes: It's not entirely out of scope. The way that measurements are done puts pressure on how people do this.

Matt: Chaired the IPPM working group, which started 10 years too early. Its intent was to do these metrics, to qualify networks. Maybe this problem needs to be redone or looked at.

Jari: Like the metrics idea, but we need to define technically what to measure. We have SpeedTest.net, but we need SuckageTest.net as well.

Jari: Question about transport and application layer working together. Does there need to be some interworking between those layers and the network layer?

Matt: Suppose you have an application running voice, video, and background data. If you're trying to do synchronized video, it might make sense to put video and audio in same queuing class. On the other hand, if there's no hope of synchronizing, you will give up on video timing and you want voice to jump ahead of video i-frames.

Jari: That leads me to a different question, which is how ambitious do we want to be? That sounds too ambitious.

Matt: It's something we could say SHOULD today and someone will cross it out.

Jari: So today we say do ECN and for other cases it's complicated.

Gettys: I'm happy to have my home router drop a pile of packets that happen to all be the same frame. If I can run one out of every 4 frames and sync the audio, I'm better dropping out those middle packets. We can make things smarter.

Matt: And gamers are tuning their bandwidth shaping already.

Gettys: Yes. And they use DiffServ marking because they notice that since most routers are Linux boxes, they can win in at least one direction. DiffServ has been deployed; it's in routine use by gamers.

Lars: What are the constraints for which we think we can actually build solutions? Right now maybe we can't do much to get people to have segregated queues. Maybe you can detect the problem and alert the user that there's a problem and they need to upgrade their routers.

Gettys: The enemy of the good is the perfect. As the gamers have showed, you can go home today and make most of your problems go away, if you know what knobs to turn in your home router.

Lars: Are we going to require or assume that people have done that, or can we benefit people who have not done it?

Matt: No. We need to target people who are kicking their kids off the network when they need to do RT. Doing something that works well enough when you're competing with TCP and giving them tools to fix their problems.

Lars: I'm worried that people say you're screwed if you're stuck in a queue behind TCP.

Matt: Even with ideal AQM and LEDBAT you might still be screwed.

Gettys: You have a large degree of control over upstream in your home router. You usually have a lot more bandwidth in downstream and final bottleneck is your wireless link. Trying to engineer around multi-second delays is insane.

Matt: TCP is almost certainly too aggressive in small windows – more aggressive than we want. Circumstantial evidence from Bob suggests you want a window of 40 packets. So below 40 packets, standard TCP is more aggressive than ideal.

Cullen: Experience with metrics at IETF is that we create metrics that are easy. It's not enough to know your network isn't good. If we're going to do metrics, they will not be useful if your broadband provider tells you it's your wireless access, if your wireless access tells you it's your antenna. We need metrics that tell you who to call when there's a problem.

Gettys: I've seen a commercial tool that claims to do that – it plots where the problem is at this instant.

Matt: It's a hard problem. The problem is there's always a one-hop ambiguity in traceroute.

Cullen: But ICMP-based traceroute has nothing to do with UDP.

Matt: There's an old draft around using fine-grained timestamps.

Matt: This problem is solvable. If my suckage is different from my neighbor for the same source...

Randall: About how there is no solution to being driven into the ground by a large flow. But once you realize you're competing and you're losing, you can transition to a loss-based solution.

Matt: Then you win on throughput but you lose on queuing delay.

Randall: Yes, this is what you do when you realize you've already lost.

Matt: How much energy is it worth to solve that problem?

Randall: If you don't have bufferbloat -- not every node does -- you might have workable delay.

Matt: Many users have a way of working around these problems.

Randall: But sometimes it's the neighbor who's causing the problems.

Hannes: If you assume people shout at their kids when they're using Skype, then the problem is much easier because your real-time traffic doesn't compete with anything else, so you can just discuss on frequency of feedback or other details.

Matt: It's easy to do the easy part of the problem. Fill a mostly idle link with real-time traffic. How much more scope do you get out of that? I don't know.

Gettys: In the home, we have 4 locations of problems. OS upstream direction that will hopefully get fixed over next few years, home router in both directions, premises CPE (cable modem), and broadband headend. The two in the middle are inexpensive devices -- $150 total at most -- so it's reasonable to posit, in an era where we need to upgrade for v6 anyway, this hardware will get swapped out. It's not a lot of money on the Internet or vis a vis ISP revenue.

Randall: Those are not large dollars on our scale, but in third world countries routers will stay in place a lot longer. On cable networks, you're sharing your bandwidth with your neighbor, you can yell at them to stop, especially on the upstream.

Colin Perkins: You would be surprised how long it takes to upgrade devices even in developed world. But I'd like to know how long and how often bufferbloat is a problem on typical users. What proportion are they suffering from this? My impression is that it is occasional and high-impact.

Gettys: Our traffic is dramatically changing. A lot of us consume large amounts of video, so I posit that the frequency of problems is getting more and more common – more and more video will give you more and more problems. Dave Thaat thought to use variance in NTP data to do global observation to find out how much bufferbloat is hurting people.

Colin: We should figure this out.

Matt: In applications, there is a control loop that happens: let's do more until people complain and then back off. If you look at the level of sharding, when you visit CNN.com you typically get more than 60 connections. The content providers have actually usurped TCP congestion control, and the reason they do that is because for most of their users it works better. There's no way the buffer can be big enough to hold 60 connections -- it's a lot of data.

Lars: Still hearing that segregated queues are good and if you have bufferbloat you're screwed. So we can say we're going to assume that users have upgraded their home routers. Or is there something we can otherwise?

Cullen: Totally agree that all of this is happening. Voice is working for a very large number of situations well. So obviously I would rather see that expanded so it worked other places where there is now bufferbloat. But I'm worried about the approach that says "let's do the minimal" and that's what we did with voice -- nothing. If we do the same with video, we're going to be in trouble. It's going to fail. So how do we deal with the congestion problem that video might cause. In the class of space where voice is currently working, that's where I'm worried about video.

Lars: So in the class of space where voice is working, we're worried about video failing.

Welzl: The point of my stuff about coordinating flows together is for us to not shoot ourselves in the foot. The point is if you have video in the browser it doesn't cause bufferbloat.

Bill Ver Steeg: Three things:

1. flows competing with flows on one system
2. flows competing with flows in one home
3. flows competing with my neighbor's flows

The first two we can work on, the third we cannot until we recognize DiffServ markings across boundaries.

Gettys: Problem is that we have more and more going on in the home. Problem is only going to keep getting worse, but it's not hopeless. Just don't know how to engineer around this.

Matt: Narrowest problem that makes sense is to avoid self-inflicted queuing delay. Then making it work with competing TCP is a notch above that, putting in instrumentation is a notch above that.

=== 13:30 - 15:00: Desirable Properties of Solutions (Lars Eggert) ===

Lars: Not going to start with desirable properties of solutions, but going to pick up on where we left from the last session.

It doesn't look like we will be able to accommodate all different kinds of constraint scenarios -- varying delays, cross-traffic types, etc. So question is under what conditions do we believe we can successfully develop a solution? I want to have a solution that works under well under the largest space possible, but this problem is so hard that we are unlikely to succeed in that.

If it's only me and my media flow, do we want to be able to solve for that? If I have a bufferbloated router, maybe not - we may not be able to solve that in the browser. Not sure we understand which conditions we can handle -- de-bufferbloated routers, deployed ECN, etc. What can we do now given that the network is as it is? What can we do in the future hoping that the network evolves in a certain way? Hopefully we can develop an approach for now that is simple and works ok and that we can extend in the future (or that doesn't fight with its current incarnation) when the network supports it.

Matt: I painted a dire picture earlier today, but it was more negative than necessary. You don’t have to have queue segregation, you just have to get to a point where there’s idle capacity, and a queue that can accept your packets. Making that work well is doable and would help.

Lars: Can you explain it again?

Matt: In typical networks, the reason voice works is because networks are underloaded. Requiring segregated queues assumes you're sharing the network with other greedy traffic. As long as there is idle capacity and when your packets arrive and the queue is empty, it doesn't matter if queues are segregated. So the idea is to design congestion control assuming some free capacity is available. It doesn't want to cause delay. It might require social engineering to ensure there is free queue space. There may be corner cases you can't solve for, for example, with LEDBAT when the LEDBAT threshold is too high. Likewise for a bunch of other cases, if you have a really good AQM and there's bulk traffic, I might be okay if my delay tolerance is high enough. These special cases might be the majority of the network in some environments today.

Stuart: I said I thought queue segregation was required but Bob has pointed out to me that it's not necessary for every router to segregate. The key is that a queue not be full. If you have ingress filtering to ensure that incoming traffic is not being abusive, you don't have to repeat that everywhere in the network. You don't need queue segregation; you need some mechanism to ensure there is space in the queue.

Ted: Looking at Mo's TFRC slides. If you look at FHD (>2Mbps), even in worst case scenario for video, you're not trying to fill the pipe. We have situations where we have senders we perceive as being abusive, but their aim is not to fill the pipe. They will not produce media streams that grow to 10Mbps because their sending rate is auto-rate-limited by the production of the video. We need to ask ourselves if we're trying to get TCP to be friendly to media streams which are already rate-limited. Or are we asking media streams that are already rate-limited to be TCP friendly? Quoting McGregor: "It's really not good to be TCP-friendly because it's not going to return the favor." The question I'm asking is, if the desired properties we want are no starvation, fairness, and effective goodput for the offered loads, is the only thing we're willing to consider changes in RTP control? Or are we willing to consider changes in TCP congestion control? That's a bigger, broader field.

Lars: Given urgent need for deployment -- time scale for changing TCP congestion control will happen on time scale as deploying ECN. It will happen but not quickly. This is the difference between TCP and media; with TCP, more bandwidth is always better. Even below maximum bandwidth.

Gettys: Camera resolution keeps getting better and better, and we’re not at eye resolution yet, although video has gone from 12 Mb/s to 4 Mb/s in the last 10 years. So that's sort of true, but entirely.

Gettys: A desirable property of a solution is presuming we can get the network fixed, the solution will work properly. LEDBAT goes back to competing with TCP when delays get low. I would like that not to haunt us.

Lars: In what cases do we think we can build a solution for the network as it is now? And in the future? I want a little bit of a scoping discussion.

Bob: The problem is TCP fills the queues. We're talking about how to change queues and change RTP, but we’re not talking about changing TCP. We can change TCP as quickly as RTP.

Eric Rescorla: No.

Bob: We can define it as quickly.

Bob: Doing something else that doesn't solve the problem will take until the death of the universe. We ought to be trying to solve the problem.

Harald: The idea that there is a single problem is dangerous.

Bob: I'm just saying don't rule it out.

Harald: This workshop was convened with a fairly broad scope because there are multiple things that need to be teased out. We need self-fairness. Self-fairness can be doled out quickly. We need to defeat bufferbloat. That’s a longer timescale. We need to change TCP, and that’s a third time scale. There are multiple problems and probably multiple efforts. If this workshop gives us a more common understanding of what the separation is between the different problems, that's good.

Lars: My problem is to implement an algorithm in the browser that handles congestion. The other things will change in parallel. But neither this meeting nor the IETF RMCAT BoF will take on what we can do for congestion control in media that will work okay now and in the future.

Harald: Disagree about the scoping because I think this workshop is about real-time media on the network, including identifying multiple issues. Identifying an algorithm that can be implemented in the browser is the subject for the Thursday RMCAT BoF. That is one thing we know we need to do. This workshop should seek to define what the other problems are.

Lars: Frontloading the discussion so the BoF doesn't fail was a goal of this workshop. This needs time to discuss. Talking about the related issues is important. But the main issue is if there is something that the IETF can take on.

Eric Rescorla: I heard earlier that TCP will send data as fast as it can. That's true from when TCP is used for file transfer, but if it's used for WebSockets, the dynamics are different. WebSockets and SPDY are changing the behavior of TCP.

Colin: Agree with Eric Rescorla. We can't change TCP quickly, but the way TCP is being used is changing quickly. And we can change the way TCP is used. If we're putting stuff into web browsers and that impacts web pages, then maybe the browsers ought to go talk to themselves. The Netflix-style applications are huge users of TCP and those applications can change quickly. Some things they do cause problems and those people are aware of it. Things are not fixed.

Matt: Some of us have been plotting to change TCP for years. Another thing that helps is real-time is almost always slower than best effort. Best effort is often orders of magnitude faster than RT. SPDY makes TCP behave more like it was intended to behave. The sharding is what makes TCP more like 1/p, much more brutal.

Mo: I think we need the solution to be able to evolve quickly. I think there is consensus that it needs to be broader, more like Michael's model. Entity could be in OS kernel or home router, not only in the browser. A solution should have some model for how it could scale to all those places.

Lars: There was a bunch of work on this in the mid-90s. This is a compelling idea. It would be awesome if you could share information about a bottleneck with all the flows sharing the bottleneck, but it's hard. It's an awfully big architectural change.

Bill Ver Steeg: real-time flows don't consume the network only as long as they adapt appropriately. This is a much bigger problem than it seems like. So if we can emphasize the need for aggressive downshift and light upshift that would help.

Carsten Borman: Solution that we come up with would need to work in today's common (bad) case. But it should be able to immediately take advantage of improvements in this area.

Lars: In the common worst case situation would you include a heavily bufferbloated router?

Carsten: [both note-takers missed this]

Michael Welzl: I would like to stop discussing bufferbloat. It's not leading us anywhere. We should consider avoiding self-inflicted queuing. That's as much as we can do right now.

Stuart: I believe we can design congestion control algorithms that work on mostly idle networks. Skype mostly works. The situation is not as bad as it's portrayed. The key goal is that a congestion control mechanism should not tread on its own toes and cause its own delays. Something like queue segregation can be developed. Those are separable problems and we should work on them in parallel.

Stuart: Also, Netflix is not real-time. Same with Apple TV.

Eric Rescorla: Agree with Stuart. If I'm using some sort of real-time application on a congestion link with other people, it shouldn't destroy my network. It would be nice if I could run two video calls at once and have them not kill each other. I'm okay with yelling across the room at my kids to get off.

Stefan: If we're limiting the scope to ensure that we don't have self-induced queues, does that mean we're touching TCP?

Lars: No, we can't fix that.

Stefan: So we're fine with having queues if caused by TCP.

Lars: TCP won't back off, no matter what we do. We want media flows to be self-fair because otherwise we have nothing.

Cullen: Three problems of interest:

1) Broadband uplink. Things get dropped here a lot.
2) Broadband downlink, DSL or cable modem. Different problem.
3) Local, particularly wireless network. Lots of weird things happen there for different reasons.

There are three different places that are important to think about.

Lars: Roll back to my intro: jittery latency like we saw earlier on the 3G plots is a path characteristic where we can ask ourselves, "Is this a link that we intend to work over? Or not, unless they fix the underlying network?" For any path characteristic, we have to ask what the range is that we intend to work over.

Mo: Back to self-inflicted harm avoidance: congestion control should be designed so that even uncoordinated instances are still self-fair. Everything in the browser should be fair, but independent instances should be fair, you shouldn't have to yell at your kids.

Lars: Like TCP, self-stabilizing without explicit communication.

Gettys: We're very often adjacent to a bottleneck. Just a major step forward would be to even try to get the single host running the RTCWEB stuff to behave properly and put its own packets on the wire appropriately. Right now OS is probably doing too much queuing before it even goes and gets stuck in your wireless. There is a lot of goodness there.

Lars: True, but it's outside the IETF.

Gettys: But if you tell the people implementing the stacks what is needed -- they're optimizing for data centers right now. Line-rate bursts are fine. Those stacks are insane. They don't have any semblance of fairness.

Lars: If we could reduce e2e delay, that would be nice, but I don't see that as the crucial thing. I don't think the few ms we can save there are the lowest hanging fruit, other than asking OS developers to pay attention.

Gettys: You actually have control over these buffers.

Colin: The thing to not forget: whatever we do cannot make it worse than it is.

Lars: My definition of “working” would be “working nicely”. It's not a free-for-all.

Keith: Can we distinguish different purposes for what TCP congestion control is for? One reason is to deal with contention on bottlenecks. But even if we were alone, would still need TCP's clocking just to adapt the rate to the link. Netflix/YouTube-style applications need that. Even if you weren't concerned with congestion control, you would still need something to choose the rate to send at. Are those separate problems?

Lars: They are the same problem, maybe. I haven't thought about it much.

Magnus: Today most applications are limited by app's desire of how much data it will send.

Keith: I could be sending faster than the link rate though.

Magnus: Different solutions, for example, IMS tries to solve this ... the reality of what the link can take ... it's a problem but I'm not sure applications agree on how to solve it.

Lars: I think the IAB sent a letter to RealMedia about its congestion behavior: if loss exceeds a certain threshold, they were just sending twice.

Magnus: General problem is that we avoid congestion collapse, and that we get some type of self-fairness. That needs to work over all types of links, which means having to cope with really bad behaviors. Mechanisms to do that must be robust against buffer-filling.

Lars: Right, Mark said it's two things, avoiding collapse and fairness. Collapse avoidance should work everywhere. Circuit breaker is one mechanism. But self-fairness is harder and may not work everywhere.

Welzl: Proposal for congestion manager: We should standardize as a minimum element is a flow-state exchange. The flow can store info about its existence, and you can retrieve that information on the host. Based on that, you can implement a congestion manager. Then we can limit at first to one RTCWEB 5-tuple, but it can be extended later.

Lars: IPPM has been doing metrics for a long time. It's incredibly difficult to accurately find a metric. Hard part is defining exactly what it is that you stick in there. Not saying this is a bad idea, only that it’s hard.

Welzl: Agree.

Mo: We need to have unique media start-up behavior in mind. Even IW10 is nothing compared to HD video slow in start-up. This question needs its own two-hour discussion and should be separated from congestion avoidance, just like TCP.

Lars: But it's not really because start-up behavior affects ongoing performance of other flows. You can’t blast at line rate at the beginning of a video stream. You need to start slow enough to not cause congestion to others.

Mo: That should explicitly be part of desired properties and solutions.

Lars: IF you start as fast as TCP starts, that's safe.

Mo: 10 seconds or 3?

Lars: Can calculate how long it takes -- it will be an interesting discussion.

Harald: I'm not sure we can always trust everyone. For instance, if I have a browser and I'm telling the congestion manager on the computer that I used priorities from 1 to 10 and you have a browser and you're using priorities 20 to 30, they will always compare one way. We might want to be a little skeptical of being cooperative.

Lars: Always a problem with priorities that no one should misrepresent.

Randall: Start-up issue needs discussion. If you have a congestion manager, you might have a better idea of what your local bottlenecks are, how WiFi is affecting you. That information can be used to seed the search for the bandwidth limit. Maybe I use the bandwidth from my last call to seed the next call.

Lars: Maybe you start with one codec and then ramp it up gradually. Maybe it doesn't work well, I don't know.

Randall: For a real-time video application, you really need to have most of your bandwidth right away.

Lars: That's not how the Internet works if you don't have quality of service.

Cullen: It's really hard to decide what to do when you're writing an application. The instantaneous experience in the first part of a call is highly determinative of people's perception of the call. Vendors are using vague heuristics from the last call to figure out what to do on the next call. Lots of people do this and I don't think that will go away. We could give advice about how to do it. The real time people will say the i-frame needs to be there straight away, but we have a lot of ICE RTs before that needs to be there, so we could take advantage of pre-information with a couple simple measurements we can get pretty quickly. We don't break the bridge, but we get a heavy load over the bridge ahead of time.

Lars: Agree we want to describe something here that works. Ramping up quickly is not always the best -- if we're on separate calls and yours drops because mine ramped up quickly, that's bad news.

Ted: We have had a lot of people who seem to be worried about the sky falling. The sky already fell. RTCWEB is trying to replace existing proprietary technologies. It may ramp up the amount of use we're expecting, but it's not doing much that wasn't being done by Flash or Skype. Good news, this is not a totally novel context.

Ted: Are we trying to tell JS programmers what they should do? Are we telling browser developers what they should do? Are we trying to tell the network when it sees traffic what it is that it should do? We need clarity on those questions. Some are more tractable than others.

Lars: I don't think we're talking to JS developers other than for setting policy. We're not talking to the network people, unless we're talking about deploying QoS, ECN, de-bufferbloating things. We're talking about browser vendors.

Ted: So you said the most that we're telling JS is how to talk about stack ranking. Browsers are the focus here. Network will get told what browsers are doing so they can be incentivized to deploy ECN, etc.

Lars: That's my personal view.

Colin: On startup -- you need good quality video quickly, but I don't know how quickly. For a phone call I need to be able to say hello pretty quickly, but for video, how quickly? 1s or 0.5s? We need to understand that, going into any WG meeting. Audio requirements are different from video requirements.

Lars: We were already wasting so much time on handshakes and other crap that we need to send data right away (that's one argument). You can parallelize sending the media and doing security checks in parallel; just don't render the media until your security checks are done.

Xiao Ching (Cisco): Could we also leverage compression, rate adaptation for video? Are there even better ways? If we don't have perfect congestion control but achieve utilization of 90%, but also build in robustness to losses due to congestion, we might be able to get a better experience? Also, people's subjective perception of video tends to have a lot of slack. Quality degradation during scene changes gets totally ignored. Instead of solving hard problem of congestion control in isolation, if we consider it in broader context of opportunities offered by real-time media, that might help.

Lars: People have said that strict TCP-friendliness is not a useful thing to aim for. We should be able to exploit specific properties of media.

Bill Ver Steeg: Re how to get a good bandwidth estimate at the start -- There's work going on that would allow us to explicitly ask what the last mile bandwidth is. While we're thinking about what we can infer from the network, we should also think about what we can ask.

Lars: Bandwidth or available capacity?

Bill Ver Steeg: Need to figure that out.

Lars: If it's bandwidth, that's an upper bound, if it's available capacity, that's even better. ECN is something else - if it's deployed you should use it

Eric Rescorla: Having video lag the audio really sucks. It’s incredibly distracting, and makes people think the video is not working.

Mo: One desirable property is decoupling from any media knowledge, generalized to not be media-specific. We should just focus on characteristics of the flows, low-latency and variable rate but capped. Knowledge about encodings and FEC are good to know but they shouldn't be dependencies because those technologies change so fast.

Lars: We want a solution that, absent any knowledge about video encoding works, but if you have the knowledge, you should use it.

Mo: But we need a set of metrics that are purely network-related.

Lars: Agree, but I'm wondering about what optimization possibilities you give up if you do that.

Mo: A lot of people are moving away from FEC now; many video streams have no i-frames at all, so we need to keep that separate.

Jari: Good scope would be to think about what the browser implementer can actually do independently. We've been deploying ECN and de-bufferbloating, but we should be supporting the browser developer. When browser is developed, you may be able to ask the PCP router for some information, you might ask for ECN, you might not get any of this, and then you fall back to something based on delays and drops.

Randall: When video and audio are not treated equally, you have problems. Frozen video is worse than no video. Segregating video and audio is problematic. It’s far better to dramatically ramp back and try to solve the problem than to freeze things or let the video lag.

Matt: I view congestion control as an algorithm problem, not a protocol problem. There are three things that are different about these kinds of systems. Natural lumpiness of video means algorithms can look at correlation between instantaneous delay and instantaneous [? Both note takers missed this ?] can help. Also, different parts of the signal have different values. And different parts of the signal have different delay tolerances.

Magnus: Ensuring maximum video quality is a competitive product space. We shouldn't focus too much on this aspect therefore. Having interactive real-time media, we should think about examples of behavior that we have, but we know that different use cases have different requirements for behaviors. Video is one high-bandwidth use, but it's not the only one. Need to think about virtual worlds, more parameterized ways of measuring things. We have high-bandwidth audio cases as well. Behaviors will change over time.

Gettys: There is a lot of presumption about one-on-one teleconferencing because we have experience with it. But other important scenarios are different. IF you're broadcasting a talk, high-quality video may matter more than lip-synced audio/video. Need to know all the use cases.

Maire: We're making a lot of assumptions about how we use applications. The flip side of another use case is the opposite is the hearing impaired -- they drove this industry before the rest of us got involved. We should not limit this technology being embraced by as many people as possible.

Randall: That's a case where the tradeoffs all change -- having the highest possible, most steady frame rate possible is critical if you are doing sign language.

Maire: 25-30 frames per second is super important.

Lars: We don't want to preclude those use cases from working. But it's also going to be difficult to realize those requirements over all Internet paths.

Magnus: WEBRTC might be driver for the moment, but we’re not only talking about browsers.

Bob: On metrics and how you might compare the effect of one application on another. Think about the idea of congestion-volume and congestion bit-rate. Congestion bit-rate measures how much responsiveness they have to getting out of the way. Total volume over a whole real-time flow divided by the volume that gets through the network -- can think of that as normalized amount of congestion it caused. So if you took congestion volume and divided by the total volume, it gives you a metric for how friendly the flow is to other things.

Ted: It tells you how friendly but not how successful. Which bit you lose matters a lot. If you're looking at the congestion mechanism and trying to normalize it, this is too academic for things we're going to tell browser vendors.

Bob: But the bit you lose is the academic part. This tells you how much someone else is costing you. This is about the effect you have on someone else, and you don’t know the value of the bits that THEY lost.

Lars: This is a metric for comparing congestion controllers. If you ran this metric against the output of another controller, you could know the differences between them. This also ignores that the packet causes 200ms of delay.

Randall: Couple other things to keep in mind:

1) Changes don't just happen because of competing flows coming and going. Can also be changes in underlying wireless network -- can move from one 3G or 4G network to another, for example. Speed and accuracy of detecting these problems are both important, and those are contradictory. Fast and right are hard to both achieve.
2) Another big piece of congestion puzzle is bursts of delay, spikes of delay, especially at start-up time. A lot of flows that we're competing with are short-term flows on your machine or another one, so most of the bits in the flow are sent before you get good feedback on congestion. This

gets worse with IW10. Web sites are good at sharding. That is very short-term and congestion control mechanisms don't even get a chance to react.

Stuart: Big fan of congestion-volume concept. You often hear about people complaining about bandwidth hogs. The reality is that a good congestion control algorithm, if person A is using 10x as much bandwidth but it's only when the network is idle. If they aren't filling up buffers that affect you when they are full, they're not doing any harm. But if person B is sending less but in horrible bursts, they are causing more problems.

Gettys: I think I just lost my train of thought.

Lars: Bufferbloat?

Room: (Laughter.)

Zahed: Moving between different access technologies -- you can have different capacities dimensioned.

Lars: Congestion control guys know that adaptation is important.

Zahed: Like the idea of congestion-volume. You can actually see like if you're really contributing to congestion, and you can react on that.

Gettys: Speed is much more determined by latency than bandwidth.

Mo: Transition from proprietary solutions to standardized solutions -- need to be able to interoperate with previous proprietary variants.

Lars: Still need to be fair.

Lars: We didn't look at the slides for this session at all, and there's good stuff in there – please check it out!

=== What can we do? (Bernard Aboba) ===

== Ted Hardie

Pokerstars is the canonical example – poker application with video and audio, instead of text and chat. The important person at any moment is the person who is betting. Have fairness within 5-tuple, but you may also have a ranking among flows. Want to catch video from the bettor.

In WebRTC architecture, the JS API takes care of relatively little congestion control. All of the heavy lifting is done by browser and network. We don't want to change that - we don't want JS API to have to take on more complex RTP processing.

== Harald Alvestrand, papers 3 and 6

Tried to match up desirable properties of solutions with what is happening within current RTCWeb description.

When we get into congestion situation, we want to reduce bitrate of streams that can spare it and add bitrate to streams that can benefit. Browser does not know these priorities, but JS application does. So JS must rank streams and browser must implement the prioritization.

Doing important stuff first means we must have a communication path to show importance. JS API does this.

We want handles for classification at other nodes. RTCWeb uses standard SRTP for media. But SRTP over DTLS looks like one flow, so any prioritization within that needs to be done by the browser.

We need to detect proper flow groupings. RTCWeb application knows which flows semantically belong together. The working group has done a proposal using DSCP markings, but that’s not standardized.

Stefan has done a delay-based congestion control proposal. Original design was based on theory that we can retrofit into existing RTP-based systems. The proposal measures changes in the one-way delay to receiver. Bandwidth estimates can be decreased when delay increases, and increased when delay has been decreasing/stable. This only needs feedback once in a while compared to ACK-based mechanisms. Current version is slightly more flexible than original version, using TMMBR for new messages.

Bernard: Is feedback purely RTCP stuff?

Harald: Receiver looks only at timestamp at the moment. As part of further work that is needed, we want to consider packet loss.

Bernard: And response is not multiplicative?

Stefan: This could use AIMD response curve, it doesn’t currently. More work can be done in that area.

Stuart: How do you measure one-way delay?

Harald: We don't. We look at the RTP timestamps of the incoming packets and the time that they arrive and we measure the change in the delay from one packet's arrival to the next timestamp's arrival.

Ted: Packet inter-arrival time for one-way is the only sensible way to do that.

Stuart: Computer clock drift can be shockingly bad, but I’m not getting a sense that this is a problem in practice. Depending on the connection, if clocks drift, how do you disambiguate that?

Stefan: Clock drift tends to be much slower than delay variations, so you should be able to compensate.

Stuart: Question is how to compensate.

Stefan: Indeed.

Stuart: NTP will reject time values that are off by more than 5 hundred parts per million. Drift is more than I expected it to be.

Randall: Drift is not a problem in my experience with a very similar algorithm, even when the drift is orders of magnitude larger than NTP.

Randall: My proposal is to make decrease proportional to the level of change in delay, as compared to constant as it is today. My experience is that works better in practice.

Zahed: With multiple flows, will they both get same capacity? How do you model capacity estimation? In different cells you can estimate different capacities than in same cell, for example. Model doesn't really work for different frame lengths.

Stefan: We have a model for estimating changes in queue length in the network. That model gives the shared capacity. We aren't factoring in differences between frame sizes and the time it takes to transmit those frames. That part isn't that important for this algorithm to work, you could ignore it.

Zahed: I agree, just wanted to make sure that you don't assume something that is not true.

Xiao: Is there a guarantee that you will converge and not oscillate? If delay is increasing, decreasing, or within a certain threshold, then you increase or decrease -- it was a 3x3 matrix. Is there any guarantee that the state machine doesn't keep jumping between increasing and decreasing?

Stefan: You have to keep trying to increase and use available bandwidth.

Xiao: Would you make smaller steps?

Stefan: It's possible but not in the draft.

Hannes: Are those design details too far in the weeds and don't matter that much? From a browser perspective, aren't there more important properties?

Harald: We were asked to present the details.

Ted: I model this as an application talking to the browser which has a little network in it.

== Xiaoqing Zhu, paper 12

When high-end flow competing with low-end flow, we want each one to have same relative bandwidth, e.g., 50% of whatever they can achieve. So bandwidth naturally gives higher rate for telepresence and lower rate for WebEx. We can easily layer on top of that a self-described priority level.

Bernard: So the same endpoint is doing telepresence and WebEx?

Xiaoqing: No, two different endpoints competing over one bottleneck. Trick is to achieve fairness without communication within competing senders.

No network assistance from ECN/PCN today, but we wanted to leverage full suite from loss, delay, ECN, PCN.

Sender takes in periodic congestion reports (delay, ECN, PCN). The sender makes its own rate calculation based on that and application-level priority. The algorithm is designed to converge on order of 10 seconds even when these parameters are changing. Scheme can also be extended to combine with FEC. So you can squeeze out video rate and put in adaptive FEC if you know what's coming.

Example: 10 flows competing over 30Mbps bottleneck. They converge to 2 different tiers of rate without any explicit scheduling. PCN used with 90% target link utilization.

Lars: PCN only works within one DiffServ domain, right? And that’s not end-to-end.

Xiaoqing: For PCN we only consider where it's at the edge of the network.

Bob: They're just using PCN markings in the queue, not the whole PCN architecture.

Keith: Why is 10 seconds of convergence not way too long?

Xiaoqing: If you change your priority level once every 10 seconds, algorithm converges faster than that. It's on the order of seconds.

Keith: On 4G it can go from 2 Mbps to 0.2Mbps quickly -- can't keep sending at 2Mbps for 10s of seconds.

Xiaoqing: It won't converge there, but it will try to track the underlying dynamic.

Keith: If you have a step change, how long does it take to converge?

Xiaoqing: Several seconds. It depends on how much damage you induce. Sender buffer can be tweaked to mitigate.

Keith: Inducing a several second e2e delay is problematic for video conference.

Xiaoqing: It's not a several second delay, that's the convergence time.

Bernard: Are you using RTCP as input?

Xiaoqing: Not now.

== Sanjeev, papers 27 and 13

We need congestion control at low congestion level. Loss-based TCP and TFRC will push queues to full length. Delay-based techniques are problematic on noisy networks (3G, 4G, WiFi, etc.). Might infer delay as being congestion when it's not.

Bill Ver Steeg: Is there a way to have the link layer to inform the higher layers that the loss it just saw was because of me?

Sanjeev: May be possible.

Available bandwidth estimation (ABE) will often suffer from similar issues to delay-based techniques.

DCTCP: infers congestion level using ECN markings. We can average markings to infer congestion level.

When you don't have ECN, you should look at queuing delay, which itself might be noisy. So you can look at one-way queuing delay trend. If it's increasing, that's a sign of congestion. To guarantee fairness need higher-rate changing curve to be above the lower-rate changing curve.

For virtualization-type applications, using hybrid rate plus delay essentially you can pace out the burst so you don't get false congestion signals.

We should use all possible signals available, ECN, loss, delay, one-way delay trend. And think about what IETF can do to incentivize broader ECN deployment.

Ali from Cisco: Why do we need to achieve lowest level of congestion for best performance?

Sanjeev: IF you minimize congestion for real-time media --

Ali: But you might still be within delay bound.

Sanjeev: But lower congestion usually means lower delay. This gives you a lower probability.

Mo: I like the conclusion of using all the signals. I haven't seen anyone show how to harness all the signals properly. How do all the feedback loops interact? I think we need to get some serious work towards a cohesive model. Google's model augmented into a model that takes more factors into effect would be great.

Sanjeev: You don't want to falsely assume congestion.

Colin: Observation that the common filter parameters don't matter much is equivalent to observation that if you're not the one causing congestion, you can't do anything.

== Colin Perkins, paper 19

One key difference from TCP bulk transfer application is that we've got much more restricted scope for adaptation. Many codecs have limited set of rates and it takes time to change. It helps congestion control algorithm produce stable results if you know what you can adapt to.

Video data can be bursty, sometimes because of periodic i-frame or to repair errors. If we can control when the codec sends bursts -- there's no point in having a codec generate a new i-frame if the network can't take it. So if you could negotiation between the codec and the congestion controller, could time the i-frames better. Might benefit from closer coupling of codec and congestion control, but this requires coupling between layers and additional feedback from the codec.

We usually talk about feedback as feedback on packet loss. IF we're using RTCP, we can tune rates. But we don't want to send more feedback than media. If you tune RTCP appropriately and use AVPF, you can get feedback on interesting events, but not feedback on every packet, which is ok. TCP responds to one burst of loss in an RTT. But we've got a limited amount of feedback bandwidth. Maybe we can just report a lost slice or picture rather than packet. Sending PLI/SLI is less bits to send back in terms of feedback, and sender knows which packets were lost anyway. RTCP can send richer semantic feedback.

Bob: One innovation of DCTCP was that it responds less to small events and more to big events, but you're arguing for the opposite.

Colin: I want to say what semantic event has happened -- loss of a whole picture, or just part of it? So that's a measure of how much.

Bob: So it's not just the event --

Colin: It's what the event means.

Bob: What message would this look like?

Colin: There are RTCP packets to say which portion of an image has been lost. There's no reason we can't extend this.

Bob: Why does that feedback need to be in the protocol?

Colin: Sending 1 packet to say you lost the picture is much more efficient than sending one packet per packet lost.

Bob: But the higher level thing doesn't help it know what was lost.

Colin: Does it matter? You've got to recover them all.

Bob: It doesn't know what its congestion response should be though.

Colin: Sure. If you're trying to be TCP-friendly, this is useless. IF you're trying to do something that avoids congestion collapse, this might be good enough given limited feedback bandwidth.

Harald: This illustrates nicely that we might be talking about two separate things when we talk about feedback. One is what the bit-rate is that you can currently send and the other is how to recover from the bad state you're in. Semantic loss seems to be the better fit for the latter. That argues for calculation of loss that can be done at the receiver.

Colin: With congestion control people usually talk about just feedbacking on loss. But we've got a more flexible feedback mechanism that doesn't send very much.

Harald: We violently agree.

== John Leslie, paper 14

“Trying to come up with a solution that won't necessarily happen in my lifetime.”

We should be looking into "backpressure" in addition to ECN, CoDel. Backpressure is how long uplink [both note takers missed this].

At transport layer most important thing is to impersonate a reliable heartbeat. ECN can go in both directions. Signaling congestion to the application layer needs to be standardized.

At application layer, we can always play with video frame rate. Application has best shot at being minimally disruptive. Audio "silence" can be more or less aggressive. Need explicit real-time signals for lip-sync.

Colin: All of this is in RTP already and has been for at least 15 years? You can do all of these things in your application now if you want. It's in the RTP sender report information. I'll send you a pointer.

John: They are now done from receiver to sender; they need to be done at the transport layer.

Colin: These are all at the transport layer. The sender can do them.

John: Question is what signal it has to know what to do.

Colin: Type of feedback is something this group has to decide. Sender has scope to do this.

Bufferbloat is a problem. Traditional AQM cannot solve the problem, will need to look into CoDel to solve the problem.

Problems come from competing traffic. We are probably going to have to have an effort at standardizing CoDel. Backpressure is not ready for standardization.

== Dirk Kutscher, paper 15

Solution to congestion in existing real-time systems like IMS is QoS, dedicated queues, etc. This works but it's not what we have in mind with RTCWEB. We want to do this without telco framework, with no control architecture and no QoS framework.

Two problems:

1) Impact of real-time flows on TCP. May be perceived as large enough threat to operator to trigger retaliation, endangering RTCWEB deployment.
2) Impact of TCP on real-time flows. Poor queue management yields unhappy RTCWEB users.

This talk is focusing on the first problem.

If real-time starts behaving unfriendly and affects the business of operators, they have all the machinery to block/filter it. So our job is to enable real-time senders to not cause damage, making operator intervention unnecessary, and to provide protocols for application-agnostic network policing, especially in mobile networks. It would be better to do it this way than to give operators the excuse to fall back on DPI as a dubious way to police real-time traffic.

There are environments where ECN can be used (data centers, mobile networks are introducing it). So don't be pessimistic about it.

Bob introduced the congestion volume idea and we leverage that in our paper.

== Jim Gettys, paper 32

There is no single bullet. We have lots of mechanisms, operating on different timescales.

Really like fq_codel: fair queuing is only 2% of CPU on GigE. Problem recently discovered in CoDel that Kathleen and Dan were working on.

Current broadband has single bloated queue. DiffServ markings will be games. Andrew McGregor has suggested looking at incoming markings to set outgoing markings.

Home routers are a disaster. Brand new hardware is running software 5 years behind current Linux.

IPv6 deployment is now gated by bufferbloat, broken home routers, and a broken funding model for getting anything replaced.

Advanced version of OpenWRT is out there running CoDel. Urge people to come fix it.

== Discussion

Lars: Do we see commonalities in the solution proposals? One thing that came up earlier was that we don't want to self-inflict queues. Seems like we need some sort of hybrid approach, to be able to detect legacy TCP and get a share of capacity and if we don't detect it, we don't have to be TCP-friendly. It's almost like Compound TCP for media.

Bernard: Is there any reason to use TFRC if you detect that nobody else is competing?

Sanjeev: No.

Lars: I don't think we want to even use it otherwise. We don't want to do worse than TFRC.

Bernard: IF you're not competing, why start filling up the queue?

Stuart: I'm nervous of proposing that we have two different modes because modes are fragile and you can be in the wrong one. If we see TCP putting 5 seconds of traffic into the uplink, then you'll go aggressive and your video chat will have 5s of delay. You need some way to get out of that mode, so you have to stop sending video for 5s, which is also bad. I wonder if it's better to push back on the user and say if you're getting a crap experience, you need a new home gateway.

Colin: Should we be using TFRC? No. Whether or not it works as congestion control, it's too hard to implement.

Bob: Not quite sure how to put this as a solution, but you can use the fact that TCP responds to push it out of the way. So you can send spikes to TCP to get it out of the queue (Bernard: send an i-frame). I don’t know whether this is a solution or something that triggers the network operators to shut you down.

Lars: We don't want to have spikes.

Bob: Proprietary applications may be tempted to do that. What opinion do we have of that?

Randall: Earlier today Stefan and I were discussing that idea, whether or not it was possible to trick TCP into giving us some of the bandwidth back. I don't know if it's possible or if it is usable. I suspect “no”. Moreover, there are mechanisms by which, when you're losing to a TCP flow, you could use a mechanism like CXTCP, which uses a probabilistic effect on the congestion window to switch between modes, low-delay and high-delay. There are tuning parameters for this. Not saying it's the best solution, but it's a workable solution for the case where you have loss.

Lars: There's a region where there's no point in even trying. But there is a region where there are 100-200ms queues that TCP generates, and in order to play in that region we need to put some pressure on the queue. So we don't want to be limited only to a delay-based mechanism for those situations.

Gettys: The problem with trying to interfere that much with TCP is that you had to have filled this big queue.

Bob: You don't have to fill the queue, it's already full.

Gettys: Then you will have induced jitter.

Mo: I hope we don't standardize a layer 4 jam signal. This will induce oscillation.

Mo: IF we get AQM deployed and low latency, we still need to be fair to all the flows. We can't throw TFRC out.

Lars: We're not throwing away friendliness, just the TFRC algorithm.

Ted: Haven't heard anybody saying that what we get today is insufficient – that RTP is going to give us congestive collapse. Are we pretty much talking only about fairness, and not starvation? Is there data that says Skype, Chatroulette, etc., did not cause collapse and therefore we will not do that either?

Lars: I think this is the easier problem. The circuit breakers probably work better for that purpose. Real-time requires low latency -- if we inject more data into the network it's going to create more queues.

Gettys: Nightmare is probably not congestion collapse, but oscillation. I have one anecdotal story about this, and that’s not data.

Xiaoping: Real strain in the design of congestion control is between multiple tradeoffs. IF everyone was happy with 64kbps video, we can avoid collapse – we would already be through. For real-time video we don't want to lose link utilization without building up the queue. All of these purposes are conflicting. Plus fairness, not all videos are created equal.

Welzl: There's a delay-gradient mechanism being presented in ICCRG on Monday, designed to compete with normal TCP.

Mo: Biggest challenge for the whole group is going to be Dash and LEDBAT traffic. Interactive is not going to be the biggest problem on the Internet. Netflix is always going to dominate. I think the critical area to look at is fairness/starvation with those kinds of flows.

Lars: Is there any hope that Netflix would use real-time congestion control?

Bill Ver Steeg: No. They use HTTP caching, and that's it.

=== Summary and Conclusions (Hannes Tschofenig) ===

== What can we do?

There are things you can do in the network versus the end hosts. Absent any changes in the network, is there work we can do? Answer: yes.

How do you create incentives to get ECN marking, CoDel, QoS, de-bufferbloat happening on the network? Is there work in the area of measurements that we can do to help? Answer: Yes. Maybe we can look at previous IETF work, the FCC mailing list, etc.

Is it useful to develop a browser-only congestion control mechanism? (Assuming idle network capacity.) Answer: yes.

There is a wide range of delay variation even in non-congested networks. It might be worth exploring the causes further, especially in cellular networks.

Cullen: Would be nice to get more data about the ranges, especially if we're doing delay-based algorithms.

Hannes: This is within our reach given that people here work for cellular operators and vendors.

== Two solution tracks

1) Change stuff in the network. Get ECN deployed, queue segregation, classification of traffic (DPI, QoS signaling). This will take a long time, and it’s mostly work that needs to happen outside the IETF.
2) Avoiding self-inflicted queuing. Approach is to ensure that network doesn't get congested, focusing on idle networks. Congestion control for real-time media that browsers send. This can be done on shorter timeframe. Some recommendations to change browser behavior (SPDY multiplexing, the congestion manager, change the way browsers use TCP).

Jari: ECN change is in multiple devices. What requires you to wait is that you have to upgrade routers, etc. But we should build in ability to listen to those signals in real-time congestion control. It's not an either/or.

Cullen: We're solving for when networks are idle. The IETF got a group of people together for a day to talk about avoiding congestion, and the answer was to have uncongested networks?

Magnus: Several of the examples seem to be outside the scope of what we can do inside and outside the IETF. Changing the way TCP is used in browsers is done outside the IETF --

Hannes: But the line is gray, SPDY is coming into IETF potentially.

Magnus: Sure. Fixing Netflix streaming behavior is outside scope.

Bill Ver Steeg: We should talk about DASH.

Magnus: DASH behaves that way for multiple reasons

Bernard: Focusing too much on protocols. In scope of algorithms, we had choices -- network idle or not, Stuart's point about whether you can get out of one mode. Algorithm may not change whether the network changes happen or not.

Matt: There's another partition here which is what's in scope for the IETF. Algorithms for real-time congestion control are in scope, dealing with TCP in the browser is not in scope. Also, w.r.t. idle networking, when QoS was done 10 years ago, end systems were lame so we didn't use QoS. Deploying more capacity was easier and cheaper. QoS technology was early and end systems can use it now.

Magnus: What do you consider short term?

Hannes: Shorter than long term. Maybe two year time frame.

Magnus: Two years is reasonable. I don't think we will finish RMCAT in that amount of time, more like 3-5 years. We might get circuit breakers and proprietary mechanisms done in 2 years.

Colin: Reasonable set of places to start. We need to be careful about scoping for short-term issues. Want to keep the RMCAT BoF focused on things which are achievable.

Hannes: Not quite clear what exactly you would have standardization on at the end. Seems like some of it is architectural -- if you have ECN, you should use it, etc.

Colin: Goal of the RMCAT BoF is to form a working group. So if that happens we'll need to define the scope. The BoF lines up with the start of the short-term goals. The proposed RMCAT charter is aiming for a working group that considers the type of things on the next slide.

== Design Aspects

Delay-based versus loss-based. Preference for delay-based. Need algorithm to react to all signals including delay, loss, ECN. If we have ECN deployed and running, we should use it.

Media is inherently variable. Codecs have limited scope for adaptation.

Feedback signals come in various forms

Jari: How to differentiate solutions? Delay versus loss, but we're already converging on delay.

Hannes: Seems like there's preference for delay, but some people advocate hybrid.

Jari: Another discussion is whether to push TCP out of the way or not. And I'm basically wondering if there are other outstanding questions? It seems all decided on the slide.

Hannes: I think most people agree there needs to be a congestion manager, but how it works and creating a sensible algorithm is tough.

Jari: That is one additional item on the list of what we need to decide when we design this thing.

Cullen: On delay versus loss, we didn't really compare the two. Stuff in browsers is going to be competing with WebSockets and all kinds of TCP stuff. So delay-based is fine but it needs to have a fall-back mode where it doesn't lose badly against loss-based flows.

Hannes: Relates to the question of when it competes with flows all originating from one browser versus if you're competing against an independent TCP flow on the same link -- that is not a short-term solution, that is a long-term solution.

Cullen: TCP from the browser is scheduled from the OS and we're talking about writing a scheduler in the browser. We won’t even have integrated scheduling on one machine. The two flows might as well be on separate machines.

Hannes: I was not assuming that the congestion manager would only be in the browser. Because how do you only compete against a TCP flow even if it only comes from one device?

Randall: You have to fall back into loss-based mode with higher delay. You have to handle that case without losing horribly.

Hannes: We should add a point about this.

Cullen: Needs to not fail when competing with ... (didn't finish)

Lars: If you're a browser doing real-time maybe you won't open a million connections, or you'll shorten the window.

Hannes: But the case we just discussed is where you cannot influence that.

Lars: In the browser there are a few things you can do.

Eric Rescorla: Do we have consensus in favor of delay-based algorithms?

Hannes: I thought I heard that.

Lars: They are nice. But we also want to have a mechanism to be able to grab a share of bandwidth if there is a TCP on the path, to not be starved, and if you can run with that capacity, then you have to give up.

Harald: The point is not that we prefer delay-based algorithms, but that we would like to operate in a mode that does not require loss, and delay-based is the only mode that works there. I think of delay, ECN, and loss as three modes.

Lars: Delay-based works great if you have FIFO queues. You can see the queue growing. IF you have AQM you don't see queue growing and you need ECN. If you don't have ECN, you're competing against TCP and you're fighting for queue space.

Eric Rescorla: Only the third bullet on the slide says something. No reason to believe we need to listen to all of the signals. We need three things.

1) Set of signals,
2) 2) Algorithm that you need to know what to do,
3) 3) What you actually do to change your sending rate.

Mo: We have to make sure that we don't fall apart when AQM does work well, like LEDBAT. It's not a question of one versus the other; they all have to be modeled cohesively at all times. Nothing is "versus" anything else.

Mo: Media should be the stimulus that gets us to think about this, but we should focus on the traffic characteristics and provide a solution for that. Nothing I've heard today would hurt TCP. TCP will benefit from this. The sawtooth is bad for throughput and delay. These things are useful to all transport protocols.

Cullen: Don't like "prefer to operate in non-loss mode." What we're trying to do here is achieve low latency. To not have loss you need big buffers, which is terrible for latency – it’s exactly the wrong thing to do.

Matt: We seem to be forgetting that video is inherently lumpy, independent of anything that’s happening in the network. Want to see correlation between signal events and jitter as a signal on the list of signals.

Cullen: Concerned because video is getting less lumpy than it used to be. This is an assumption. Everyone who does video, the max bandwidth is the peak that you provision for. Once you’ve provisioned for the peaks, the valleys between the lumps are free. You can fill up that space, so video becomes less lumpy. We've all been slicing it different ways to make it less lumpy.

Matt: You could take that back, it's a choice to make video smoother.

Randall: We have more than 2 classes of media: audio, video, low-delay data. Video has a very different characteristic than other media traffic - much more bursty and unpredictable.

Welzl: The charter for RMCAT calls for a framework for providing new congestion control algorithms that will fit in the scope of RTP. Priorities for different packet types, priorities between flows, varying packet sizes, different interpretations of TCP-friendliness, and these all fit in the short-term approach.

Lars: We haven't talked much about the interface between the codec and the congestion controller. Some people want the congestion controller to be application-agnostic; others say that there are things to take advantage of. But that’s completely undefined right now. Put it on the slide and ask what we need here.

Wes: Start-up behavior is an unknown now as well – needs to be on the slide.

Jari: What is the difference between this and the RMCAT BoF?

Harald: IF you start talking about changing TCP or deploying CoDel at the BoF, the chairs will tell you you're out of scope.

Colin: The aim of the BoF is to form a working group, not to design the solution in the room.

See you at the BoF!