Question 1

In *Confused, Timid and Unstable: Picking a Video Streaming Rate is Hard* the authors discuss the problems with automatic video quality selection in the presence of a competing TCP flow, and investigate methods for circumventing these problems.

Initially, the authors show how under ideal circumstances for the video streaming client (it being the only user of the network), the quality selection method is reasonable, and produces good results. When a competing TCP network flow is introduced, however, the interaction between client video buffering and TCP *cwnd* timeout causes a sharp downward spiral of video quality, often bottoming out at the worst quality. Clearly this is not ideal: the client would (presumably) like to have the video streamed at the maximum quality given the bandwidth available.

To attempt to better understand the problems inherent in modern video streaming clients, they then ran a series of experiments, using a custom client, tweaking various parts and recording any change in video throughput. They found that using a “less conservative” algorithm (allowing more perceived bandwidth to be claimed by the streaming client), using better filtering, or increasing video segment size, all appeared to prevent the downward spiral problem, in testing.

The custom client used was an attempted clone of one service’s original client, rather than a true modification, so some assumptions were made about how the original client was implemented, based on data shown in the paper. It was pointed out that the custom client did not exactly match the original client, but it was close enough in order to mimic it for the purpose of testing, as shown by comparing the behaviour of the two clients.

The experiments conducted, and the assumptions made in these, all seemed valid, and a compelling set of data was produced. The authors showed that there are methods which could potentially be implemented to deal with the problem of quality auto-selection failing in the presence of another network flow. Additionally, another paper was referred to in which a similar set of experiments and results showed that when two video streams (instead of a video stream and a generic TCP stream, as in this paper) were used to test the auto-selection methods, the same issue arose. This would suggest that the problem outlined and potentially solved in this paper is common to other situations, and the fixes implemented by the authors could have further applications.

The paper did not, however, detail problems or solutions for other situations in which a video client may be struggling to secure enough bandwidth. Perhaps the clients available from streaming services are designed to be resilient to much more than competing TCP flows, and therefore have to be more conservative. Further exploration into this could produce interesting results, and would likely be required before any of the suggested client changes in this paper could be implemented and presented to users.

On the whole, the paper suggested some interesting solutions to what appears to be a very real problem, and demonstrated their effectiveness in the specific scenarios given. Were these changes validated against a wider range of problems, they could potentially be implemented to good effect, enhancing the user experience of video streaming clients.

In *Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE* the authors propose, detail, and demonstrate an algorithm for video streaming which improves on modern solutions.

The authors set up a system of metrics for measuring streaming client performance, based on instability, inefficiency, and unfairness. Instability is how regularly the client switches between video qualities, where a higher instability is detrimental to the user experience, an irritation. Inefficiency is where a player uses less than the available bandwidth, again, detrimental to the user experience, preventing the user from watching in the best possible quality. Lastly, unfairness is used to describe the situation where several clients are using the same bottleneck link simultaneously, and fail to converge to a roughly equal share of the bandwidth per client. A higher unfairness implies a greater average difference between bandwidth being used per client. These metrics seem a reasonable measure of video player performance, covering most of the desirable (or in this case, undesirable) performance aspects of a video streaming client.

The authors then propose various methods (combined into an algorithm; FESTIVE) of dealing with problems inherent to several modern video streaming clients, and implement these solutions in a custom client. They ensure that the techniques they are using do not require any fundamental changes to any part of the distribution methods, network, or client software, and so could be implemented easily in the current streaming ecosystem. Three main techniques were combined to create the FESTIVE algorithm: “randomized scheduling,” the randomization of chunk request timings to remove start-time biases; “stateful and delayed bitrate update,” a method of selecting bitrates more fitting to a smooth curve allowing multiple clients to converge to a fair, shared rate; and “harmonic bandwidth estimation,” using a harmonic mean to estimate bandwidth to reduce the impact of outlying measurements and maintain a smooth estimation (especially important in the presence of request randomization).

A set of proofs is given, to evaluate the potential benefits of the authors’ algorithm. Some assumptions are made to enable these proofs, but the same assumptions are not maintained when the algorithm is put to the test, where, rather, harsher conditions are used, and the algorithm demonstrably delivers the same benefits.

The authors then proceed to show, using the metrics they have defined, that their solution outperforms these commercial clients to varying degrees. They cover cases in which their algorithm operates alongside other network traffic, including competing algorithms, and still maintains an increase in performance relative to the commercial alternatives. They go on to reference situations they have not explored, such as the FESTIVE algorithm operating alongside legacy players, and other situations for further research and consideration.

The authors present a compelling argument, supported with relevant data, for the usage of the FESTIVE algorithm. The techniques they suggest and implement appear logical and effective, and improve upon previous designs for video streaming clients. The only concern I would have is that they measure the performance of their solution using a set of metrics they created, and as such some further, independent, examination of the algorithm to ensure the validity of their findings would be useful. Performance outside of the steady-state (for example, on player startup) could present an issue, and there are likely other similar issues which would need to be investigated before such an algorithm could be implemented. There may be additional issues, undocumented in this paper, that FESTIVE does not solve (or perhaps even compounds) which could prevent it from being a successful streaming algorithm.

In *Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale* the authors present a technique to better estimate bandwidth (poor bandwidth estimation being the root cause of poor quality auto-selection) through probing, instead of relying on video transfer rates.

The authors begin by outlining the problems with bandwidth estimation based on video throughput, and propose a solution based around “Probing AND Adapting” (PANDA) when an accurate estimation of bandwidth cannot be found using traditional methods. They describe the situation in which the classic technique does produce an accurate estimate, and show that it is in a rare enough case that their proposal is reasonable and necessary. The PANDA algorithm is then described, shown as a simple modification to a conventional algorithm.

A set of experimental scenarios are then given, and the authors compare PANDA to a selection of other streaming algorithms, including a full TCP flow. A set of metrics, defined in another paper to measure the performance of various algorithms, is used to benchmark performance here. PANDA is shown to perform well, generally better than other available algorithms. Additionally, they show that even in the worst case, when PANDA competes against a full TCP stream (the most aggressive of potential competition) it still performs adequately. Outside of the worst case, in the presence of other competing player streams PANDA maintains good performance, generally superior to the alternatives shown.

The techniques used in PANDA are justified against the strategy of keeping the TCP connection active at all times to prevent the on-off pattern from affecting the stream. This preservation of the on-off pattern seemingly helps panda to quickly switch between playback rates when appropriate, maintaining the nimble aspect of the algorithm.

The algorithm presented here seems to be an improvement over the referenced FESTIVE algorithm, exhibiting similar performance, but with better scaling and faster response to changing bandwidth. As such, it would appear to be a superior solution, and perhaps stays closer to the traditional, conventional algorithms, avoiding heuristic approaches. This more adaptive approach could lead to better results moving forward, where unknown factors could upset the balance of tuning used in the FESTIVE algorithm.

PANDA, however, is relatively simple, changing little from the conventional approach. As such, there is likely potential for further improvement, but overall the technique used by this algorithm seems a potentially useful contribution to the world of streaming algorithms which could be built upon moving forward.

In *A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service* the authors conducted a series of experiments on the popular video streaming service *Netflix*, running trials with various versions of an adaptive bit-rate (ABR) algorithm and comparing the results of various techniques. This algorithm was unique in its usage of the buffer (commonly implemented in streaming clients to allow some of a video to be pre-loaded prior to viewing) to decide on the bit-rate of the video requested by the client.

The authors proposed the use of the buffer for this as a logical conclusion of ABR algorithms: since the purpose of such algorithms is to maintain a reasonable amount of data in the buffer, and prevent rebuffering events, the algorithm should use the amount of data in the buffer as an indicator of the bit-rate that should be requested. The theory behind such an algorithm was detailed, with a set of assumptions made to ease the proving of this. When carrying out the experiments, however, tweaks were made to the base algorithm allowing it to cope with the removal of these assumptions, which would not be present in the real world.

The algorithm functioned by partitioning the buffer into a “reservoir,” “cushion,” and “upper reservoir.” The rate of video requested was then chosen dependent on how much data was in the buffer, changing as it crossed various thresholds. As the video was stored in discrete rates, the change between quality involved a jump instead of a smooth transition, and this was accounted for.

The goals of the algorithm were to minimize rebuffering events, and maximise video quality. The experiments, carried out on Netflix’s network, involved a great many users in a live scenario. Netflix’s regular algorithm was used as a target standard to beat, and a “control” algorithm (simply streaming at the worst quality, to prevent rebuffer events) was used to provide a baseline count of rebuffer events that could not be avoided.

In the experiments, several versions of the algorithm were tested. The original proposed version; a variant better equipped to handle variable bit-rate; a variant further improving on this by by using more traditional rate estimation during the startup phase (where the buffer would be empty, and could not be used as a data source, this variant produced some of the best results); and another variant which produced similar results but produced a better user experience by limiting the number of switches between video rates.

The buffer-based approach proposed appears to be a sensible one. The paper makes a convincing argument for its adoption, having run live tests on users of one of the largest (if not the largest) video streaming service in the world. This certainly proves that it could be deployed, and the data shows that it was an improvement over Netflix’s regular streaming algorithm. Even still, there is likely room for improvement, if this is truly one of the early buffer-based ABR algorithms. While two others are referenced, they implement different strategies, which could potentially be incorporated, as well as other potential improvements. The fast start-up technique used, for instance, seems relatively basic: a proof-of-concept that could be expanded on using known methods.

Question 2

The four papers reviewed all deal with parts of the broader topic of providing high-quality, smooth, and user-experience focussed video streams.

In *Confused, Timid and Unstable: Picking a Video Streaming Rate is Hard* the general problem of streaming video over HTTP (and dealing with TCP's *cwnd* especially) are examined, and some solutions trialed which could help in alleviating these issues. The authors declare the intention of the paper as one of pure research interest, an investigation into the topic, which contains potential ideas on dealing with the issue, rather than a proposed solution.

In *Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE* however, the authors of this paper examine the same problem, and come up with a fairly comprehensive solution: the FESTIVE algorithm. They present a set of metrics, and therefore performance targets, for streaming algorithms, and run experiments to prove that FESTIVE does indeed improve upon existing techniques and algorithms. They even go as far as to compare the code complexity of their improvements to that of an open-source streaming framework, showing the relatively tiny size of their changes as well as a lack of increase in system requirements. Overall, they do not propose anything revolutionary, more an iteration on classic algorithms that pushes the boundaries of performance.

In *Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale*, the authors propose a more innovative change to the traditional algorithm. By implementing a probing method, they break away from the problem of basing prior rate on video transfer speed. They demonstrate improvements over classical algorithms, as well as FESTIVE, especially in the presence of more streaming clients, adding scalability. They show that their algorithm (PANDA) is more nimble and quicker to change between requested video rates where appropriate, the main improvement over FESTIVE. PANDA additionally handles competition with full-blown TCP network flows acceptably well, as a side effect of their strategy. They remark that simply beating the TCP problem in a brute-force manner, by keeping a probe connection active to prevent the *cwnd* resetting is not a healthy approach to the problem, and simply produces more issues.

Finally, in *A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service* a totally different approach is proposed. Here, the authors argue for the use of a buffer-based algorithm, as the buffer is the variable which is to be controlled to provide the ideal user experience. They provide evidence of various variants of the algorithm working, in a live deployment, by collecting data from the video streaming giant *Netflix*. They show several versions of their algorithm, eventually outlining a version which improves on the standard algorithm used by *Netflix* by a reasonable margin. They end by proposing a new philosophy for the design of adaptive bit-rate algorithms: focus on the buffer, a true measure of capacity, rather than methods of estimating capacity from video transfer rates.

The four papers represent a history or development of HTTP video streaming algorithms, concluding with the final paper suggesting a switch to buffer-based methods. They demonstrate a compelling argument for this strategy, and prove its viability in a large commercial network. The algorithm they detail could be seen as a basic first version, however, and there is likely room for further improvement, or at the very least, tuning.