Online Meetings Quality Analysis

A Report on Computer Network Course Lab 1

Hanyu Li, Huiping Lin, Zhenbang You

May 3, 2021

1 Project Summary

In this project, we collect 40 meeting records and corresponding network packets with different online meeting software in different network settings. Following the minimax principle, we choose environments with the worst network quality on campus to the best of our knowledge: the mealtime of Jiayuan Dining Center and the underground of Xintaiyang Student Center. We modify the code for metric computation, thereby improving efficiency by two magnitudes. We develop scripts for README generating, video processing, packet filtering and data generating. For analysis, we adopt both subjective observations and mathematical analysis methods, particularly techniques in time series analysis. We focus on general properties of the relationship between network quality and strategies video conferencing software at sender side (we call it "sender" below for briefness) follows.

Our result lies in a homogeneous one, i.e., we do not distinguish the network settings and meeting software we use. Our work shows that: 1. It is almost impossible to be in a stable network environment, and when the network environment is getting worse, the sender prefers to send smaller packets, which leads to freezing sometimes; 2. Changes in strategies for sending packets are not prompt in response to the sudden changes in the network; 3. We explore the features of video quality and strategies taken by the sender in network environments of different qualities. In a highly disconcerting network environment, freezing is the prominent feature, and the sender does little (or can do little) to improve the video quality at all. In a favorable network environment, freezing and blur happen randomly regardless of the strategy the sender takes; 4. We analyze the meaning of the baseline network speed. It turns out that the baseline network speed partially reflects the quality of the meeting but guarantees nothing on odd glitches of the network quality. Nevertheless, the glitches do not influence much on meeting quality.

The task partitioning is as follows.

1. Record collecting. Huiping Lin records all meetings and captures all packets. Hanyu Li and Huiping Lin collect real-person records. Zhenbang You and Huiping Lin collect records of virtual meetings.

2. Data processing. Zhenbang You modifies the code of metric computation and calculates the Metrics. Hanyu Li cuts off and transcodes the records. Huiping Lin processes the packets and plots all primitive results obtained from the data.

3. Data analysis. We discuss the factors to consider. Zhenbang You proposes a discretionary analysis. Hanyu Li makes a mathematical analysis based on techniques of time series.

2 Data Acquisition

In this section, we describe the approach to collecting data. On account of the requirements of the Lab, we need to collect video records by the following principles: 1. minimax principle, the poorer the network quality is, the more suitable it is for recording; 2. sequential recording.

We record 40 videos in total, with each associated with a `pcapng` file containing all the network packets captured during the meeting. Those videos are produced by the screen recorder X-Box, and the `pcapng` files are produced by Wireshark.

Our online meetings are conducted on four platforms: DingTalk, Tencent Meeting, Zoom and Teams. The network settings involved are 4G-to-4G and WiFi-to-WiFi, with One side on the third floor of Jiayuan Dining Center and the other on the negative second floor of Xintaiyang Student Center. We choose these two places because the network conditions there are among the worst ones on the campus. In addition, our meetings are mostly recorded at mealtimes, when the network payload of the Dining Center achieves a very high level.

The 40 meetings contain two stages: the real-person stage and the virtual-meeting stage. In the real-person stage, two sides chat at will, read selected novel passages, or play music pieces. These activities correspond to real online meeting situations. In the virtual meeting stage, we play two previously prepared videos, a Chinese video, and an English one. We also use two different virtual meeting methods: Open Broadcaster Software and Screen Sharing.

3 Data Processing

Given the diversity and abundance of the data we got, we deal with the data on the principle of automaticity, efficiency and robustness. We cut off and transcode the records, filter the packages, modify the processing code of metrics computation and run the code.

We encode each meeting in a specific order so that we can process the data automatically by scripts. The processing is done mostly on a server with the following specifications:

- System: Ubuntu 20.04.2 LTS (GNU/Linux 5.4.0-70-generic x86_64)
- CPU: Intel(R) Core(TM) i9-10980XE @ 3.00GHz, 18 Cores, 36 Threads
- Memory: 125GB DRAM

3.1 Record Processing

To make the computation more efficient, we have to decrease the quality of the records. To make the analysis more manageable, we have to align the data, i.e., cut off the videos so that all records have the same length.

We use a python script to call `ffmpeg` to cut off the records into 5-minute videos; we use Premiere to crop the videos to focus on the camera view of the other side; we use Media Encoder to transcode the videos into low-quality H.264 format with the size of 30MB.

3.2 Packet Processing

We use Wireshark to capture packets during the meeting process. When processing the data, we first scan all the packets and filter packets with UDP protocol. Afterward, we select packets with the correct source IP and destination IP which are determined by finding out their modes in the first 100 packets.

Then we count numbers of packet per second and then calculate the total packet size per second: This is achieved by corrupting each packet into bytes and calculating the byte number.

3.3 Metrics

We need to calculate four metrics, as given in the write-up: blur, freeze, noise and synchronization of audio and video. Example code has been given together with the source code. Therefore, our task is to modify the code so that it can run correctly and efficiently on the target machine and write some scripts when necessary. Note that we compute these metrics every 100 frames, namely 4 seconds.

3.4 Code Modification Phase I: Feasibility

Before deploying the project on a high-performance computer, feasibility should be guaranteed, which mainly refers to 2 aspects: first, the code should produce correct output; second, the cost of the code should be reasonable, e.g., it should not be too memory-hungry.

Some grammars in the given code are not compatible with our Python, such as indications of return values of functions, which can be readily solved by simply removing them.

Plus, the code computing synchronization of audio and video restricts running on machines without CUDA, definitely not suitable for tests on our local machines. The remedy is relatively straightforward: assign CPU to the PyTorch variable named "device" when CUDA is not available.

The third problem needs more thinking: the origin code loads all frames of a video into memory before computation begins, undoubtedly leading to memory overflow. Though the solution, namely loading frames in batches, is quite simple, we also need to strike a balance between memory occupation and disk traffic. Fortunately, it will not have a noticeable impact on this project even if we place no emphasis on this trade-off.

3.5 Code Modification Phase II: Performance Optimization

As seen by all, there is nothing complex in the first phase. However, when we finished that phase and deploy the project, it shocks us that finishing all the computation in this naive way is too costly. Among all the four metrics, ”blur” and ”synchronization of audio and video” are the most extreme two.

Generally speaking, for software engineers, there are mainly the following ways to improve the performance of code: 1. buying better hardware such as upgrading CPUs, which requires no work for programming but is the most costly way from an economic perspective; 2. make better use of hardware resources such as the multi-threading of modern high-performance CPUs, which means some modification of source code is needed and the workload of this is contingent on the degree of coupling; 3. changing programming language such as changing Python into a more efficient one like C; 4. optimizing algorithm or making a good approximation.

Specifically, we used all the methods above except for the third one. Our deliberation will be shown in the following paragraphs, with each method occupies one.

For hardware, the bottleneck lies in GPUs in terms of synchronization of audio and video while in CPUs for the other three. For GPUs, we turn to Google Colab, which offers GPUs with performance close to that of Nvidia RTX 3070 according to our experiences, and the offering is for free. For CPUs, we find a machine with Intel Core i9-10980XE, which offers powerful multi-thread performance though the single-thread performance of which is just mild.

After acquiring hardware resources, we are supposed to adapt the code to exploit the performance of hardware. For GPUs, there is hardly anything we can do since the offering of Google Colab only contains a single GPU, and we three are not able to optimize the underlying machine learning systems, whereas the multiple cores of 10980XE are truly something we are able to and ought to make a difference. Think about our current task more precisely. We have nearly half of a hundred videos, and the measurement of each one are totally decoupled, which is absolutely an excellent condition for Thread Level Parallelism (TLP). In contrast, if we perform computation based on a naive ”for” loop, it can only exploit the single-thread performance of a CPU, which is certainly not the strength of our target machine, i.e., i9-10980XE. On coming up with the above ideas, things become much easier now. A script is needed to create one thread/process for each video, and these threads/processes should be able to run on multiple cores in parallel. This requirement strikes out the scheme that simply uses APIs like "system" in C, but for a Linux server, "fork" in C is a perfect choice.

For programming language, it is pretty common to write a C version for high performance or use Cython or PyTorch instead of NumPy. However, due to the limit of time, this approach is not as realistic as the other three.

For the algorithm, we mainly focus on optimization of the calculation of "blur" here. Our goal is to find a good approximation, and there is one common way to do this: downsample. That being said, we just need to calculate the extent of blur of a small portion of frames, as long as a result is close to the original one enough. Base on our test, 20 times down sample works quite well. Specifically, we divide a video stream into 20-frame sections and discard all but the median frame.

3.6 Output and Performance of Metric Computation

The input videos of this part all last for 300 seconds, and there are 30 frames per second. In other words, we get 7,500 frames for each video.

In terms of blur, freeze, and noise, the output has one entry for every 100 frames. It just takes minutes to finish the computation of these three metrics. Compared with the original implementation, we obtain two orders of magnitude of performance improvement (the original one takes more than 20 hours to finish), one of which is procured by TLP, and the other of which is procured by approximate computation.

In terms of synchronization of audio and video, since Google Colab has a restriction for free users that the session will be disconnected forcibly every 12 hours, we only measure some of the videos we record. Each video takes approximately 15 minutes. By the way, as the code requires that the voice and image of the same speaker should appear in the video simultaneously and the ambient light should be appropriate, the output of our real-person videos and videos involving "zh-1" in this part are all empty. Thus we only calculate for videos involving "en-1" in terms of this metric.

We accomplish this part completely with scripts written in Python and C; otherwise, the workload of typing commands will be extremely formidable.

4 Data Analysis

4.1 A Discretionary View

Before our discussion in this part begins, we should give a disclaimer that all the analyses here are subjective and thus only qualitative. However, this does not necessarily mean this analysis is meaningless. In sharp contrast to this, our subjective feelings can give us great inspiration and intuition for our future objective and quantitative analysis. What is more, these subjective things can serve as a baseline so that the superiority of our rigorous and exhaustive quantitative analysis can be easily shown.

In terms of synchronization of audio and video, unfortunately, the output is a little confusing. To be more precise, the given code is not robust enough; as a consequence of this, all the videos starring an old artist result in empty output. Plus, the code only computes for the last fragment that the audio and the video match. It seems not that bad at the beginning since we may find every fragment manually. However, because of the horrible network condition we deliberately choose, some videos turn out to have plenty of freeze frames, which may be regarded as a disruption by the model. All in all, we are sometimes not able to tell the corresponding time from the output. In addition, just as the above has indicated, although all the videos have the same number of frames, the numbers of lines of outputs differ, which generates great obstacles for cross-video analysis.

Despite all the obstacles mentioned above, we can still procure some meaningful general conclusions. This can be greatly illustrated with some specific examples. For ”test9.mp4”, we find that there are virtually no offsets greater than 0.5 seconds between audio and video, which is consistent with our feeling. In contrast, the output for ”test23.mp4” even has some offsets of more than 20 seconds, and the extent of the freeze of the record is horrible. In short, although it is hard to get precise local information from this output, the general quality can still be well displayed.

4.2 Factor Selection

We chose two main factors in our analysis: Packet Number per second and Average Packet Length per second. These two factors imply network quality and packets sending strategy. If the packet number received per second becomes high, it indicates the network environment behaves well and vice versa. The average packet length per second implies how the center server sends packets. It's easy to imagine that when the network quality gets worse, decreasing packet size is a wise method to avoid frame freezing and response delay.

Besides, we also tested blur, freeze, noise, and synchronization of audio and video.

4.3 Result Plotting

The results of packets received in real person cases are shown in the following figures.

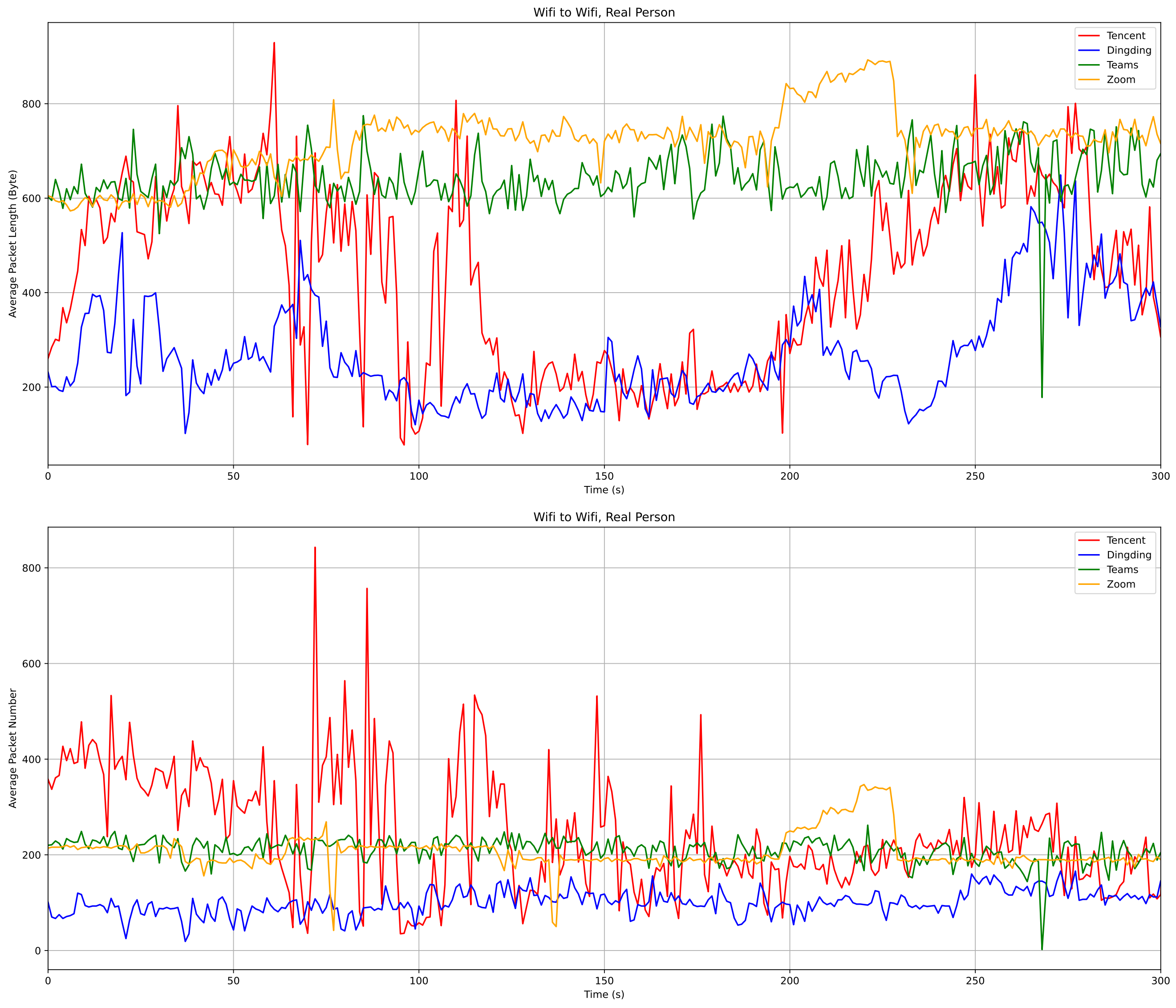
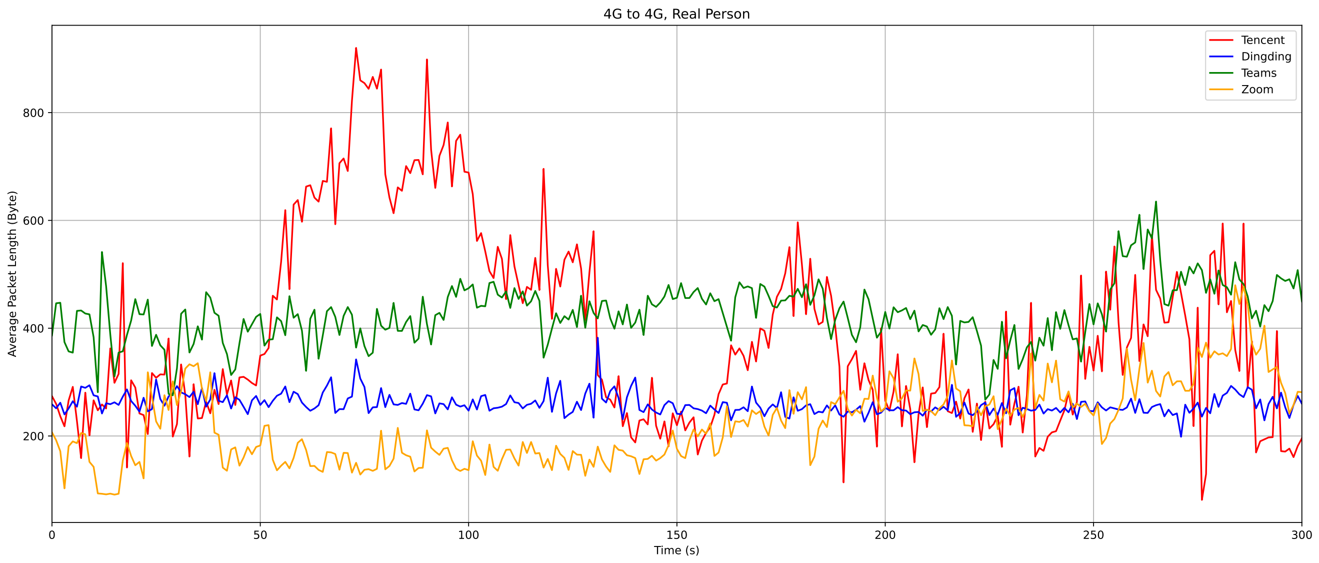


Figure 1: Results of Wifi to Wifi



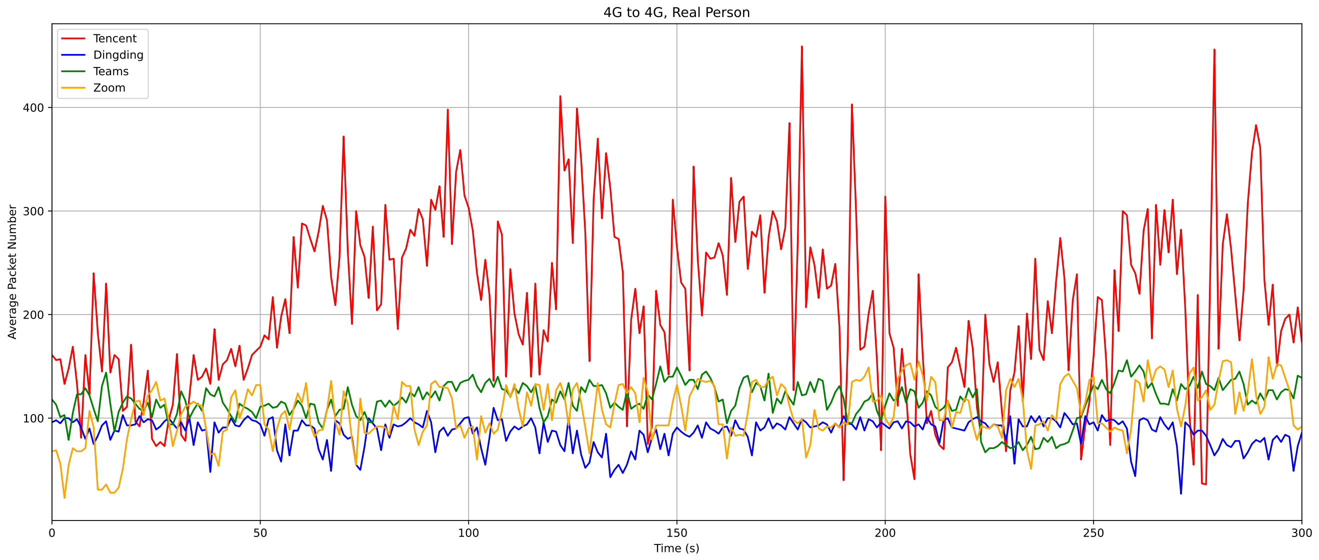


Figure 2: Results of 4G to 4G

The results of video quality on 3 in real person cases are shown in the following figures.

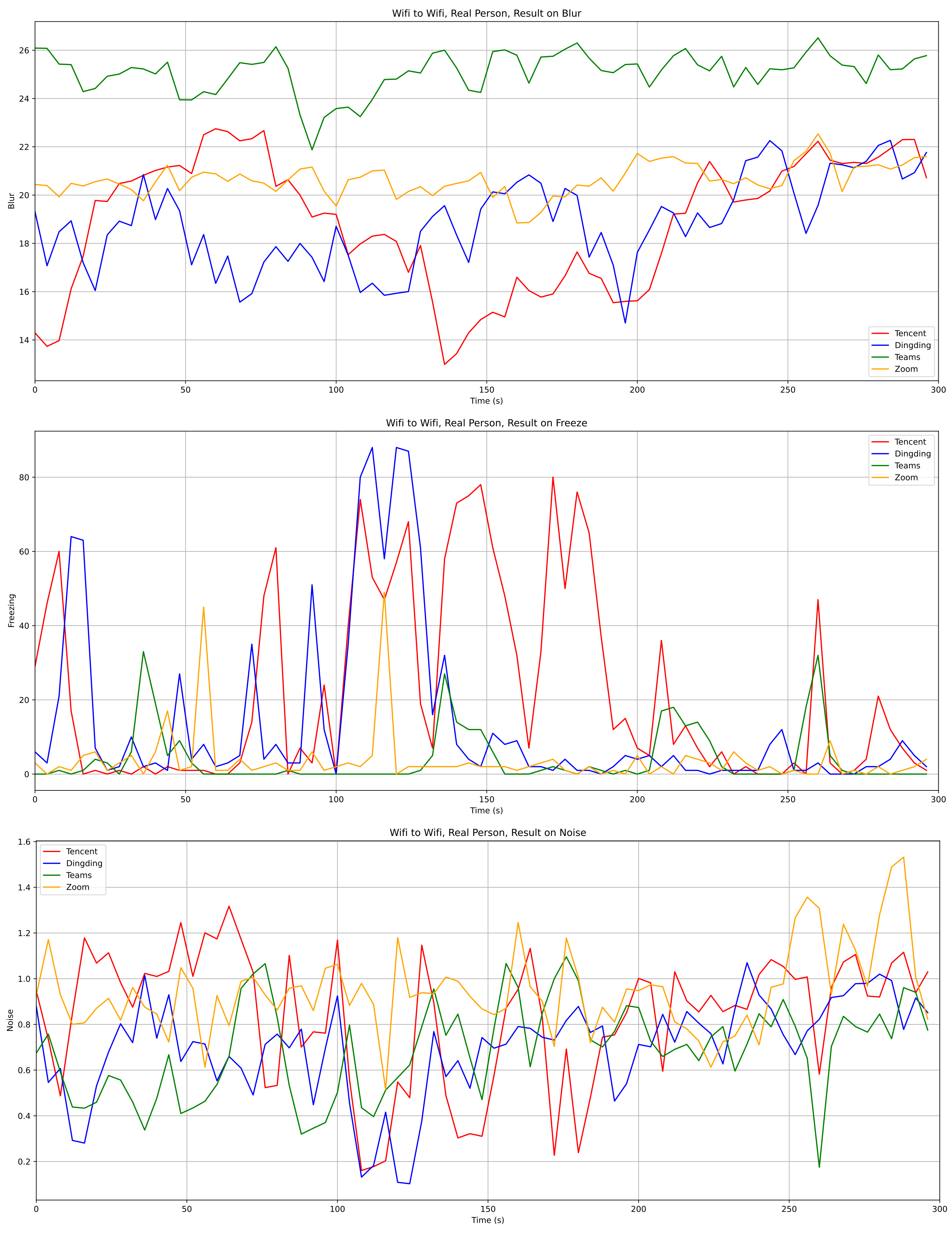


Figure 3: Results of Wifi to Wifi

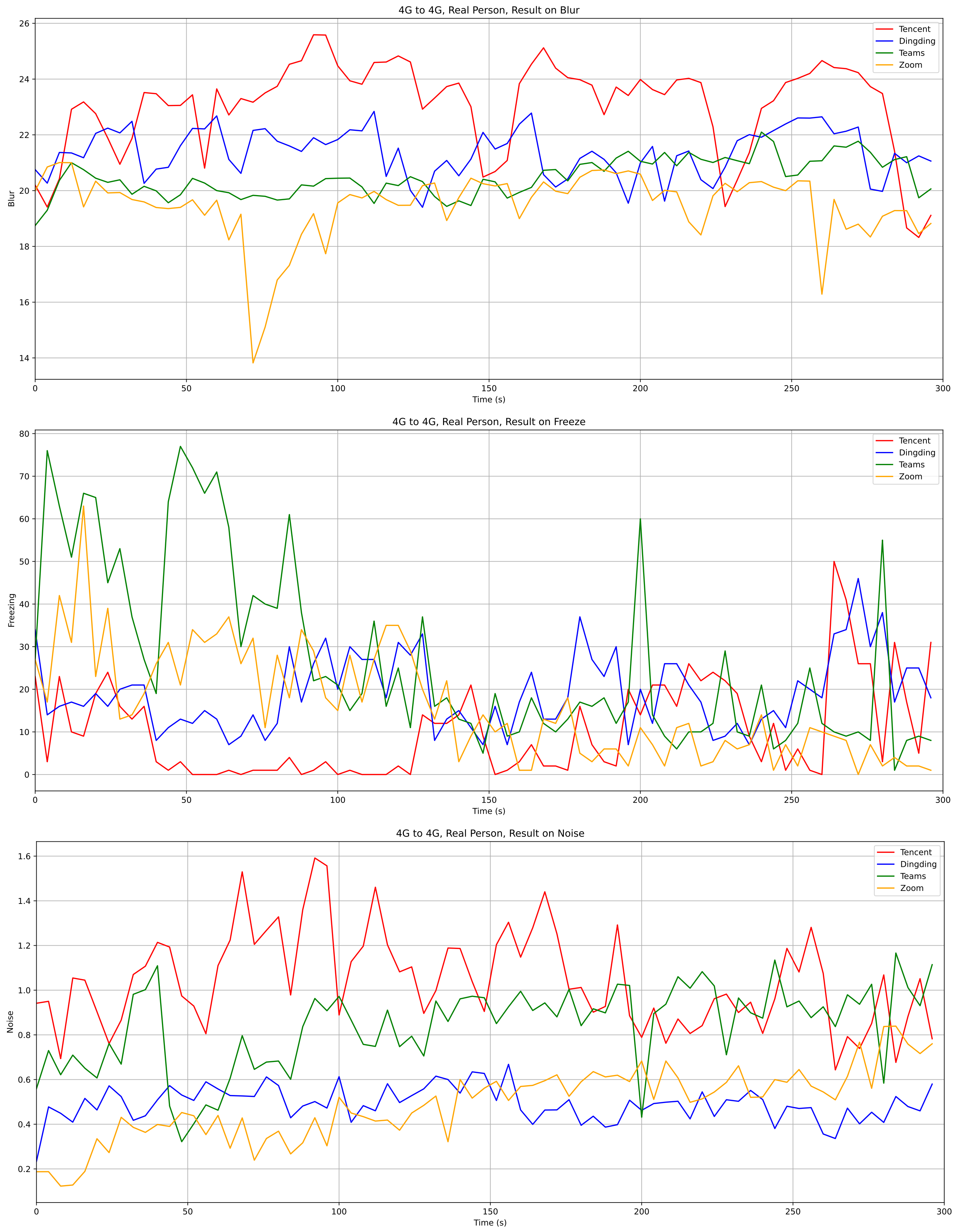


Figure 4: Results of 4G to 4G

4.4 A Crude Analysis

Though crude, we can still obtain some intuition from the statistics of Wireshark. With statistics displaying and video replaying simultaneously, we can quickly draw some simple and general conclusions regarding the statistics and the quality of the video. We focus on one factor below: packets per length, for we find this is among the most meaningful statistics we may procure from Wireshark.

At first glance, we can easily find some glitches in the broken line graph. There is hardly any salient anomaly in the video at the very point. Therefore, we believe that glitches do not tell much.

Then we focus on the general trend. Specifically, we focus on the fluctuation of packets per second. It proves to be quite apparent that videos with more significant fluctuation on this statistic have the worse quality.

Combined the above two observations together, we can safely conclude that short-time anomaly of network traffic does not matter; nonetheless, unstable network conditions are really disasters for video conferencing.

We shall leave more space for the following quantitative analysis, which will surely form a more solid foundation for our final conclusions.

4.5 A Time Series Approach

A time series is a series of random variables $\{X_t\}_t$. The index $t \in \mathbb{R}_+$ refers to the time a random variable appears. Statistical methods thrive to model and testify a time series, predict future values of a time series, and detect outliers in a time series. Therefore, network analysis is, by nature, a time series analysis. We quantitatively analyze the factors of packets and video quality using the methods from time series analysis. We only focus on homogeneous analysis, i.e., making no distinction between different network environments or meeting software. Our intuitions are from subject observations and coarse analysis. All codes used to analyze are written in R.

In the beginning, we are interested in the meanings of network factors: the speed of packet receiving (SPR) and the average length of received packets (ALPR). The length of a packet purely indicates the sender's strategy: she can either send a small packet with only audios or send a large packet with a high-quality frame. If the network overload is high, packet loss is more frequent, and thus fewer packets will arrive as expected, leading to a low SPR. The sender can also choose to send packets lazily, and a low SPR appears. So SPR profiles both the strategy of the sender and the network environment. A natural strategy for software is to choose a packet length proportional to the quality of the network. If this is indeed the fact, we can expect a positive correlation between SPR and ALPR. We verify it by correlation methods.

We cannot assume that the fluctuations of the factors follow normal distributions, and we are also concerned only with the rank property of SPR and ALPR, i.e., whether a low SPR leads to a low ALPR. So we use *Kendall correlations*, which fits our requirement the best. Given observation pairs $(x_i, y_i), i = 1, 2 \dots, n$, we say two pair (x_i, y_i), (x_j, y_j) are concordant if $x_i < x_j \Leftrightarrow y_i < y_j$, and they are discordant otherwise. Define Kendall correlation as

$$\tau = \frac{\#[\text{concordant pairs}] - \#[\text{discordant pairs}]}{n(n-1)/2}.$$

τ profiles the order properties of $\{(x_i, y_i)\}$: if they are totally concordant, then $\tau = 1$; if they are totally discordant, then $\tau = -1$. We test the correlativity of SPR and ALPR with a significant level $p = 0.05$. The result shows that *all* 40 records have correlated SPRs and ALPRs. The histogram of correlations is presented in Figure 5.

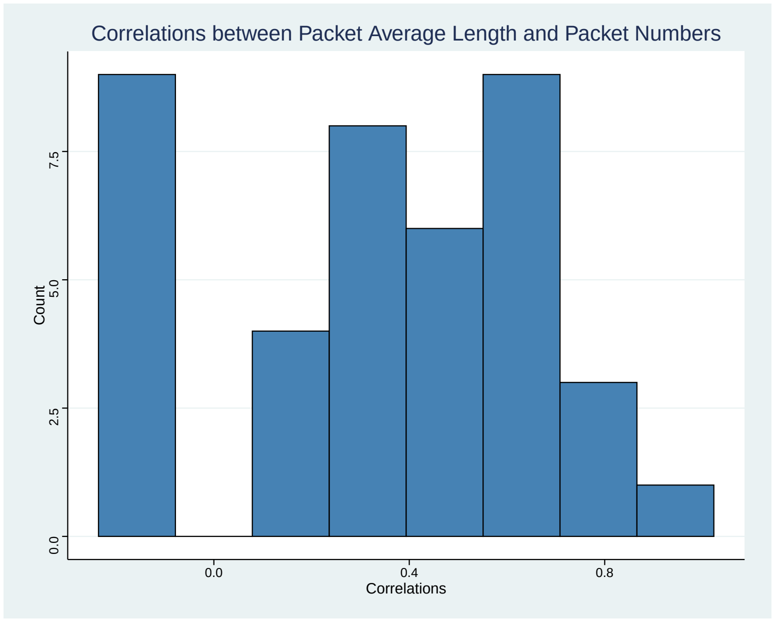


Figure 5: Histogram of Correlations between SPR and ALPR in 40 Records.

As the figure shows, over 75% of the correlations are positive, and over 25% correlations are greater than 0.6. The result suggests a positive correlation with strong confidence.

From the time series perspective, long-term observation is not credible since there are too many unknown factors affecting the model. A short-term model seems more random, hiding many latent effects, so a short-term analysis matters. We adopt the idea of rolling windows and analyze correlations in each 10-second window. The histogram of rolling correlations is shown in Figure 6.

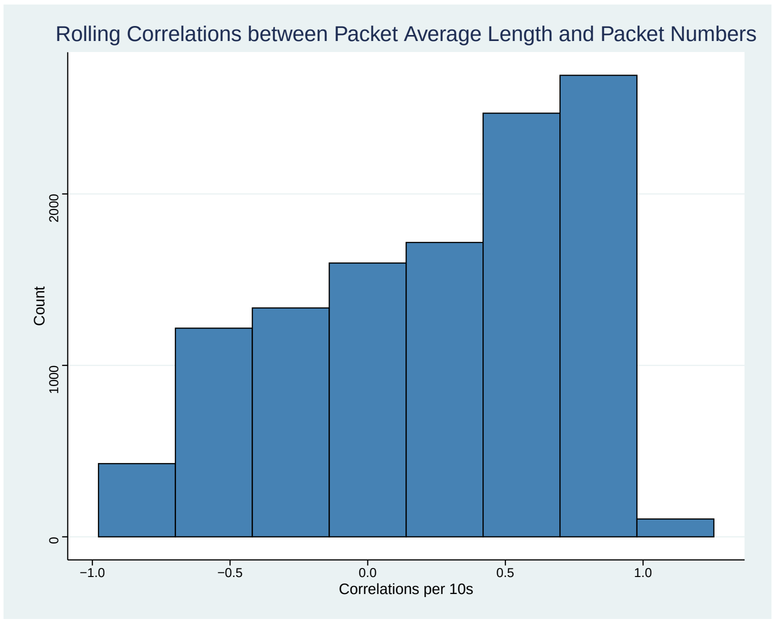


Figure 6: Histogram of Rolling Correlations between SPR and ALPR in 40 Records. The correlations are calculated in every 10s.

It can be seen that even in a short-term view, it remains true that there is a positive correlation between SPR and ALPR. So we conclude that all software takes the following strategy:

If the network environment is bad, send small packets.

Second, we turn to analyze the relationship between metrics of video quality (freeze, noise, and blur) and ALPR. In other words, when the strategy is changed, will the video quality change accordingly? Again, we resort to Kendall correlations. However, some preparations are needed. Specifically, due to randomness, factors are not stable, so we need to *smooth* the data, again, using rolling windows. In this way, we eliminate the noises in the data.

There are 300 entries (one for one second) of packets and 75 entries (one for 4 seconds) of Metrics per record. We aggregate ALPR into 75 entries and smooth the needed data every 12 seconds. We also use the Kendall test to verify the correlation of Metrics and ALPR in each record. We choose the significant level as $p = 0.05$. The result is presented in Table 1.

Type	#[Not Correlated]	#[Correlated]
Freeze	19	21
Noise	15	25
Blur	15	25

Table 1: Counts of Correlation between Metrics Factors (Freeze, Noise and Blur) and ALPR of Each Record. The significant level is $p = 0.05$.

The result implies that the video quality is weakly correlated to the strategy taken by the sender in general. We further plot the histograms for the count of correlation of each parameter for correlated records. The result is in Figure 7, 8 and 9.

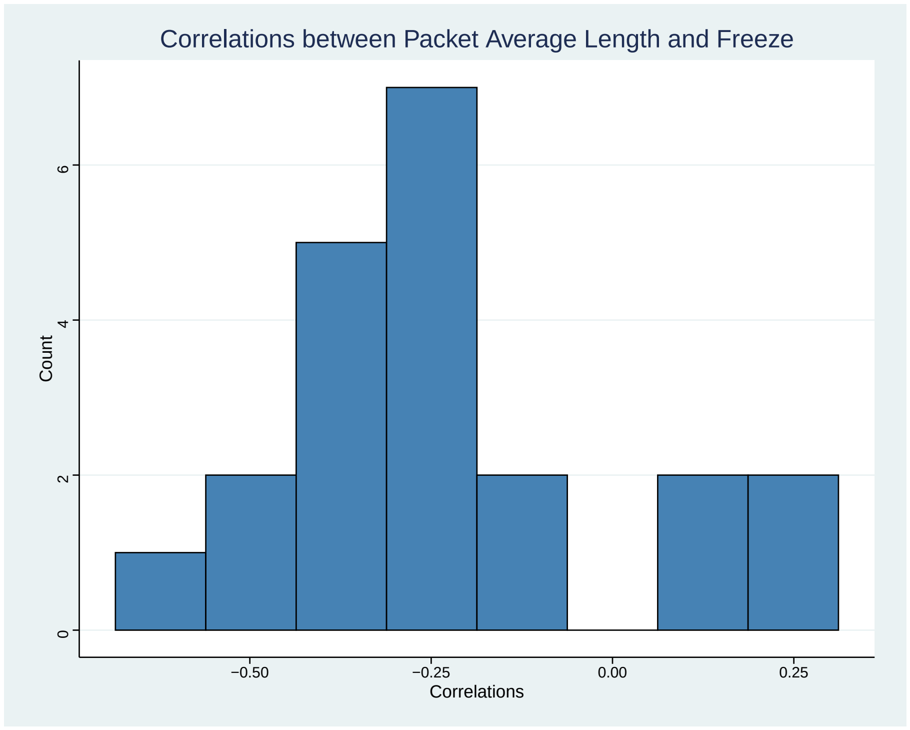


Figure 7: Histograms of Correlations between Freeze and ALPR.

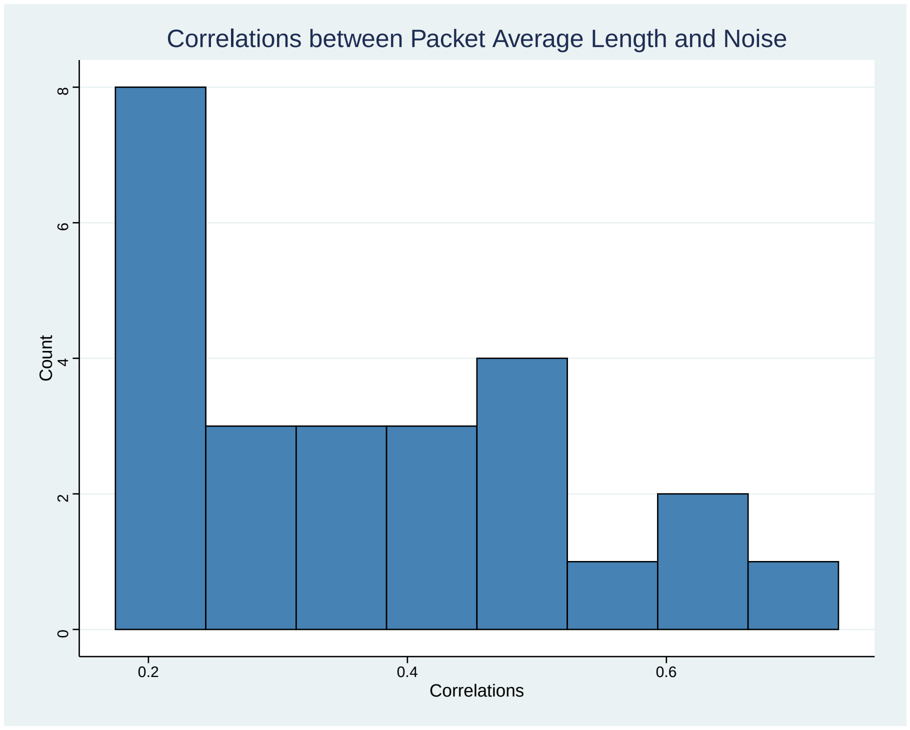


Figure 8: Histograms of Correlations between Noise and ALPR.

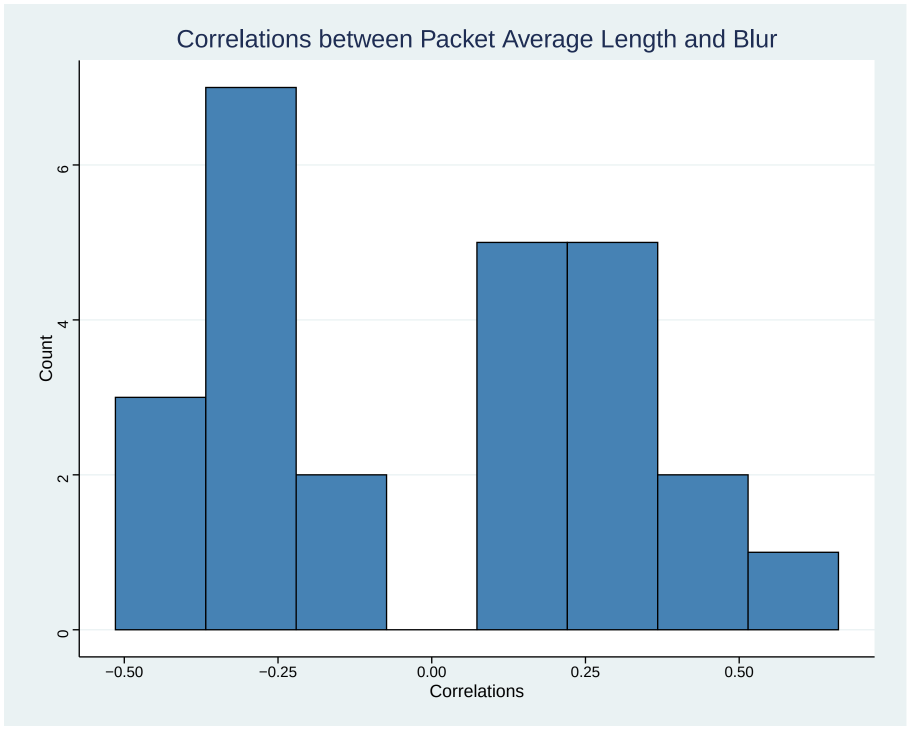


Figure 9: Histograms of Correlations between Blur and ALPR.

The majority of statistics of freeze show the correlated records have a medium negative correlation, indicating that smaller packets lead to more freezing. All the statistics of noise show positive correlations with various levels. Noise has no significant statistical meanings due to the compression of the videos and the low quality of the meetings. There is nothing to conclude for blur.

From the statistics observations above, we propose a conjecture as follows.

1. When the network environment is so bad that freezing is the major feature during the whole meeting, the algorithm detects stationary frames for a long time, so the video has no blur at all.

2. When the network environment is just okay, freezing, blur and noise occur sometimes, which indicates the taken strategy will not affect glitches of the video quality.

3. When the network environment is quite good, freezing, blur, and noise happen rarely, and statistical observation is not valid, so abnormal statistics occur.

We correspond the conjecture into statistical statements:

1. When freeze is negatively correlated to ALPR, blur and noise should behave uniform-randomly in the correlations to ALPR.

2. When freeze is not correlated to ALPR, blur and noise behave independently to each other.

3. When freeze is positively correlated to ALPR, there is no statistically valid result.

To verify these statements, we use the classic *contingency table* and *Pearson* χ^2 *test*. The contingency table of correlations between Metrics is presented in Table 2. The p-values of χ^2 tests are presented in Table 3.

Blur / Noise	NC	P	N
NC	9	5	1
P	6	1	3
N	0	0	0

(a) freeze = NC

Blur / Noise	NC	P	N
NC	0	0	1
P	0	0	1
N	0	0	0

(b) freeze = P

Blur / Noise	NC	P	N
NC	2	0	0
P	3	3	5
N	0	0	0

(c) freeze = N

Table 2: Contingency Table of Correlation between Metrics and ALPR. NC means "not correlated" or $|\tau| < 0.2$. P means "positively correlated". N means "negatively correlated".

Pair	p-value
τ_N, τ_B	0.1103
τ_N, τ_F	0.0318
τ_B, τ_F	0.0947
$\tau_N\|_{F=Ne}$, $\mathcal{U}[Po, Ne, NC]$	0.7351
$\tau_N\|_{F=NC}$, $\tau_B\|_{F=NC}$	0.1824

Table 3: p-values of Correlations. N is "noise". B is "blur". F is "freeze". NC is "not correlated", or $|\tau| < 0.2$. Ne is "negatively correlated". Po is "positively correlated". $\tau_A|_B$ means the conditional correlation of A on the condition B.

The third entry of Table 3, where $p \gg 0.05$, suggests a strong evidence that conditional distribution of τ_B is uniform when $F = Ne$, so statement 1 holds. When focusing on the fourth entry, $p > 0.05$, we do not deny the conditional independence of τ_N and τ_B. The second statement is also verified. Records with $F = P$ are scarce, so the third statement is convincing, too.

Third, having revealed the relationship between video quality and network factors, we are now able to evaluate whether the strategies are efficient given a specific network environment. We first introduce classic concepts and models in time series analysis. Then we show how these methods can be applied to SPR and ALPR.

A (weakly) stationary time series $\{X_t\}$ is a time series such that

1. $\sup_t Var(X_t) < +\infty$,
2. $EX_t = \mu$ is independent of time t,
3. $Var(X_t) = \gamma_0$ is independent of time t,
4. $\gamma_k = Cov(X_{t-k}, X_t), k = 1, 2, \ldots$ are independent of time t.

The relationship between stationariness and network quality is as follows. A stable network should have stable SPR and the fluctuations should be limited and stable. Also, a glitch means a sudden change in the correlation between different timestamps. So we propose that

A stable network is modeled as a stationary time series of SPR.

Similarly, we have

A stable strategy is modeled as a stationary time series of ALPR.

To test the stationariness, we adopt the model of *unit root test*, specifically, *Augmented Dicky-Fuller Test*. We omit the complicated mathematical details of this method but comment that a higher p-value means less possibility that a time series is stationary. We choose the significant level $p = 0.05$ and show the result in Table 4.

	Stationary	Non-stationary
SPR	2	38
ALPR	0	40

Table 4: Count of Stationariness of 40 records.

As the table suggests, most of the SPRs and ALPRs are non-stationary, which coincides with the principle we follow in recording data: the poorer, the happier. It is also worth noting that stationariness is by no means high SLR. In fact, the only two stationary time series are the two with really horrible SLR: stationarily poor.

Now we focus on the efficiency of the strategy taken by all software. Specifically, when the network environment changes, can the software change its strategy in time? This leads to two questions: 1. How to detect network environment strategy changes? 2. How to measure efficiency responding to changes?

The first question can be resolved by detecting outliers in SPRs and ALPRs. The more outliers a model has, the more changes a time series has. Since the fluctuations are unavoidable and time series are not stationary, we have to model it properly. We use a powerful time-series model called *Auto regressive Integrated Moving Average model (ARIMA)*, which is available to profile the non-stationary time series. Due to mathematical complexity, we omit the formal description of the model. Many parameters are needed to select in ARIMA. We use a selection algorithm by [2,3]. To detect outliers, we adopt the method by Chen and Liu [1].

To answer the second question, we use scatter plots. Particularly, we plot the scatter of number of outliers of SPR and ALPR for each record in Figure 10.

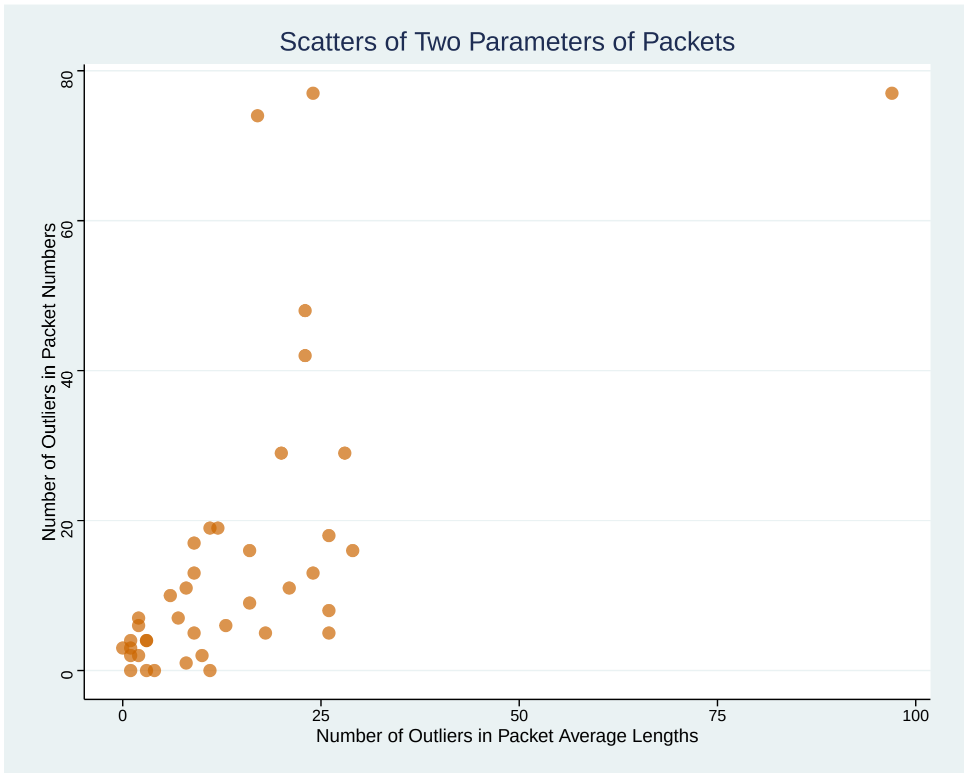


Figure 10: Scatters of Outlier Numbers of SPR and ALPR of Each Record.

As the figure shows, there are fewer outliers of ALPR than the corresponding SPR, so it can be inferred that meeting software is not acute to make changes towards network changes: maybe some lazy strategies guarantee enough performance for most of the time.

Last, we explore the relationship between baseline network speed and SPR as well as average receiving speed in KBps (ARS) and the number of outliers in SPR. Again, we use scattering plots and Kendall correlations. The results are in the following Figure 11, 12 and 13.

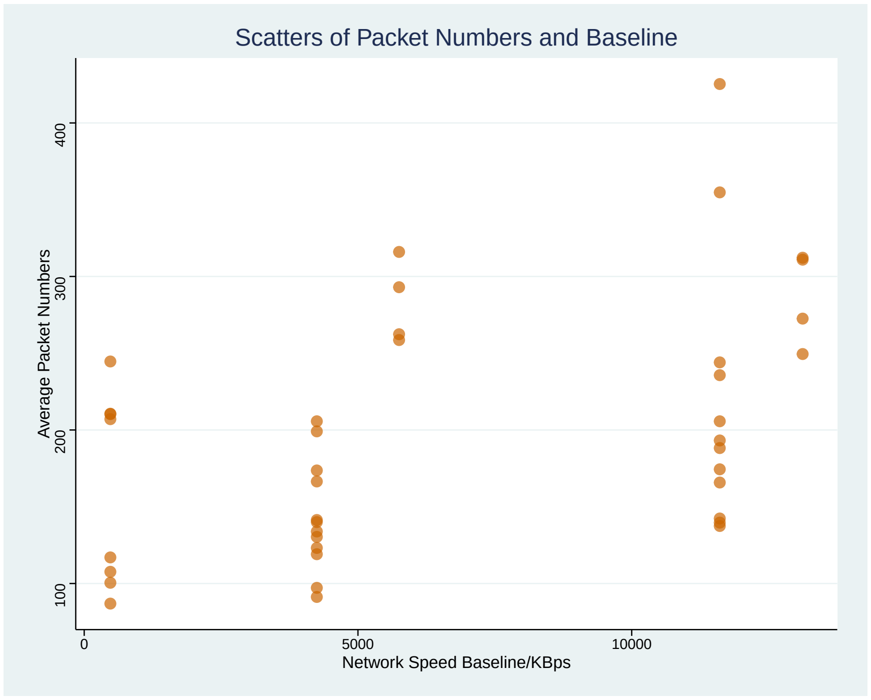


Figure 11: Scatters of Baseline and SPR of Each Record. $\tau = 0.3891667$.

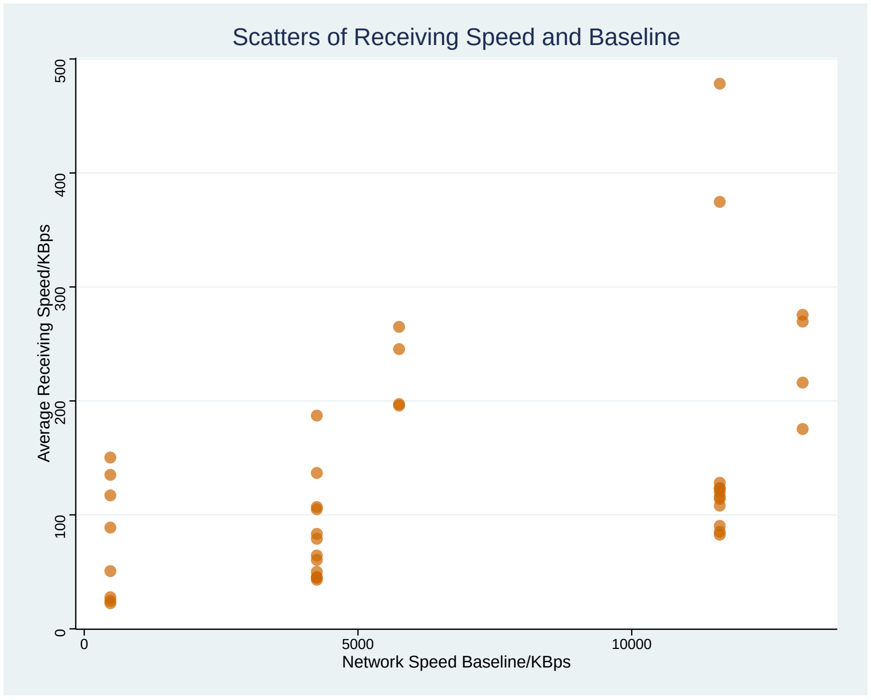


Figure 12: Scatters of Baseline and ARS of Each Record. $\tau = 0.4385386$.

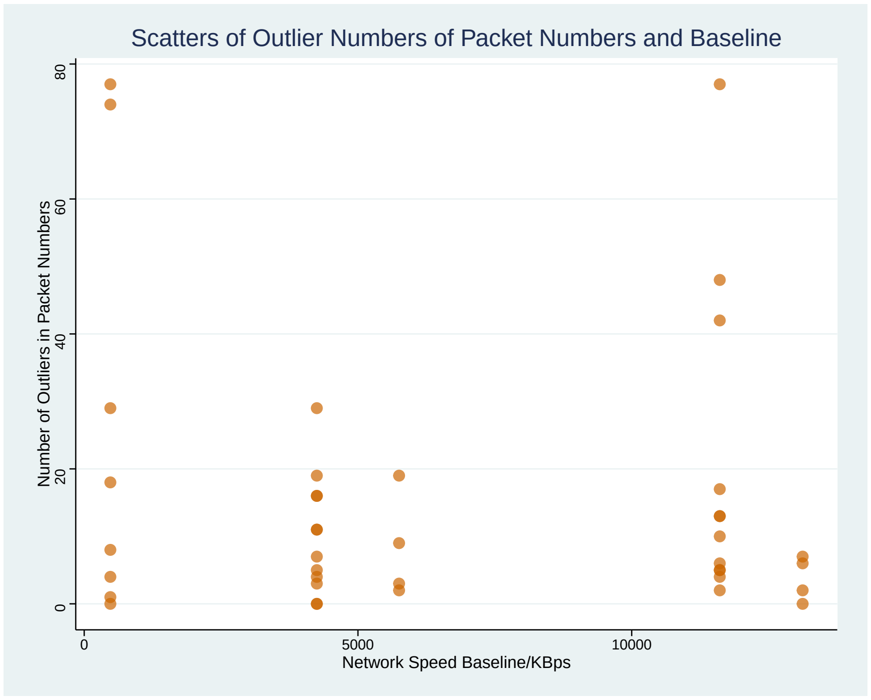


Figure 13: Scatters of Baseline and Outlier numbers of SPR of Each Record. $\tau = -0.07817415$.

As the result shows, a high baseline speed might mean a high SPR and ARS, hence a sound performance of the meeting. The conclusion is fragile, though. Abnormal fluctuations in a network are less relevant to the baseline speed. It can be explained by the difference between UDP and TCP or the difference between burst speed and continuous speed.

References

[1] Chung Chen and Lon-Mu Liu. Joint estimation of model parameters and outlier effects in time series. *Journal of the American Statistical Association*, 88(421):284–297, 1993.

[2] Rob J Hyndman, Yeasmin Khandakar, et al. Automatic time series forecasting: the forecast package for r. *Journal of statistical software*, 27(3):1–22, 2008.

[3] Xiaozhe Wang, Kate Smith, and Rob Hyndman. Characteristic-based clustering for time series data. *Data mining and knowledge Discovery*, 13(3):335–364, 2006.