CollabKit – A Multi-User Multicast Collaboration System based on VNC

Christian Beier Peter Ibach

Humboldt-Universität zu Berlin
Institut für Informatik
Lehrstuhl für Rechnerorganisation und Kommunikation
{beier,ibach}@informatik.hu-berlin.de

***Abstract*—Computer-supported real-time collaboration systems offer functionality to let two or more users work together at the same time, allowing them to jointly create, modify and exchange electronic documents, use applications, and share information location-independently and in real-time. Commonly, such collaboration systems are realised using remote desktop technology or are implemented as web applications. However, none of the examined existing solutions support concurrent multi-user interaction in an application-independent manner. Furthermore, when used in low-throughput shared-medium computer networks such as WLANs or cellular networks, most of the investigated systems do not scale well with an increasing number of users, making them unsuitable for multi-user collaboration of a high number of participants. Therefore in this paper we present a collaboration system that supports concurrent multi-user interaction with standard desktop applications and is able to serve a high number of users in low-throughput environments. Our multi-user multicast collaboration system named CollabKit, realised by integrating and extending existing technologies, was compared against a conventional unicast remote desktop system and found to significantly outperform it when several clients needed to be served. CollabKit supports application-independent concurrent operation by multiple users, per-user graphical annotations and window sharing and scales well with an increasing number of users.**

***Index Terms*—VNC; MPX; Multicast; Collaboration; CSCW**

I. Introduction

Collaboration means working together. Computer-supported real-time collaboration systems allow multiple users to simultaneously edit electronic documents, share multimedia content or use interactive applications – remotely or locally. The *real-time* properties of such systems enable users to concurrently ask and answer questions, brainstorm, and thus to rapidly draw, refuse, or accept conclusions. These characteristics make computer-supported real-time collaboration systems very useful in professional contexts – they enable knowledge workers and scientists to exchange information and to jointly create, share and modify electronic artifacts.

The first area in which common computer-supported real-time collaboration systems are limited though is support of *fully concurrent multi-user interaction*: Though there are current collaboration systems that support fully concurrent multi-user interaction, such systems are confined to one or a few built-in applications specifically designed for that system with multi-user support in mind, they do not allow users to interact with unmodified standard desktop applications. On the other hand, there is a second class of computer-supported real-time collaboration systems that allow participants to use any kind of desktop application, but they only support user interaction in a turn-taking mode where only one user at a time can be in control of the shared desktop and there is only sequential but no concurrent interaction.

The second area in which existing systems have shortcomings relates to *scalability*: when sharing applications or whole desktops – especially on low-throughput computer networks characterised by shared medium access such as wireless local area networks – the user-perceived performance degrades with an increasing number of connected users. This is because the same data is sent to each and every user individually: the more users are connected, the less throughput capacity is available to each one.

In order to address the first problem – *lack of fully concurrent multi-user operation in existing systems* – a computer-supported real-time collaboration system with support for fully concurrent multi-user operation was developed, implemented and tested. Our collaboration system dubbed CollabKit allows its users to *simultaneously* operate several applications on a shared desktop. To achieve this, existing technologies – namely the X11 windowing system and the Remote Framebuffer Protocol RFB used by VNC – were integrated to form a collaboration system with the desired features.

The second problem – *bad scalability in low-throughput networks* – could not be solved by simply integrating existing technologies though. Instead, this required enhancing the way data representing shared applications is delivered to the system's users. This meant designing and implementing an extension to the existing VNC remote desktop technology that would make data transmissions use the shared medium more efficiently. The chosen approach to accomplish this was to extend the RFB protocol with support for multicast data transmission. This allows a high number of users to efficiently use the created collaboration system on a low-throughput shared-medium network.

Figure 1 gives an outline of the envisioned system.

II. Related Work

During a detailed examination [8] of usable related work, it was found that almost all of the considered collaboration solutions support basic remote view and control features, but it was also evident that support for *fully concurrent* multi-user operation is relatively scarce: There are preceding works that support semi-concurrent multi-user operation by essentially time-sharing a single cursor between users [17], but MPX [12]

 ISBN: 978-1-61208-206-6

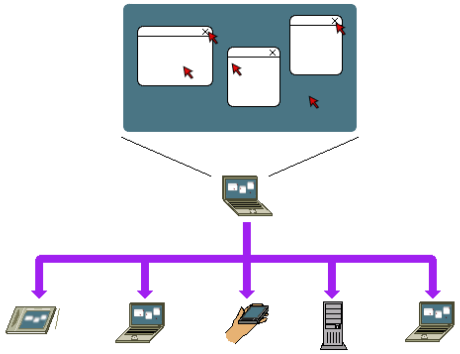


Figure 1: CollabKit is a computer-supported real-time collaboration system that supports *concurrent* multi-user interaction and transmits the shared desktop *once* to *all* clients using multicast.

is the only software supporting fully concurrent multi-user operation with mice *and* keyboards on a standard desktop. Others are either turn-taking, i.e. only supporting sequential operation [10, 13, 9], or confined to some special multi-user applications [5, 16, 6].

Since it was decided within the requirements analysis that *fully concurrent* multi-user support including cursors *and* keyboard foci is needed, the use of MPX for further work was considered somewhat mandatory. Although designed with network-transparency in mind, the X Window System X11 which MPX is based on is not suited well for sharing applications to several users: Within X11, an application can always just be connected to a single X server, without special measures it is thus impossible to display and remote control an application from two different remote computers. Then, X11's network protocol is a *stateful* one: If the connections fails, the application loses its X server and terminates. Finally, the X11 network protocol involves many round trips, which hampers performance on high-latency links [15].

Taking these findings about X11 into account, it was concluded that combining MPX with another remote desktop technology would be a more promising approach. It was also clear that for interfacing with MPX, this remote desktop software would have to be modified to support multiple pointers. Thus, only products with available source code were furthermore eligible. Another prerequisite was that there should exist a server implementation for the X Window System that could be interfaced with MPX.

There are already some remote desktop systems that to some extent support multicast data transmission: most of them are based on VNC [4, 7, 19], RTP [14] or custom protocols [9, 11]. Some of these are view-only and lack remote control support, others are not open source and thus not adaptable to concurrent multi-user support.

However, the main issue with these existing multicast remote desktop systems is that they lack complete multicast flow control with error handling. While the missing remote control features could possibly be added with maintainable effort, the lack of proper multicast flow control and error handling is a more serious problem – IP multicast is based on UDP instead of TCP and thus provides no built-in flow control nor reliable data transmission. These are important though when – to minimise resource usage – only updated parts of the screen are multicasted, as opposed to always multicasting full screenshots. Out of the considered multicasting solutions, only the newer incarnation of TeleTeachingTool [19] and VNCast [14] potentially provide multicast flow control since they are based on RTP, but they still lack multicast error handling.

Because of these shortcomings of existing multicast remote desktop software, it was decided to implement multicast support from scratch, extending an existing unicast remote desktop software. This way proper multicast flow control and error handling could be implemented in a clean fashion while keeping the existing unicast communication paths for loss-sensitive data and as a fallback.

After evaluating different candidates, the Remote Framebuffer Protocol used by VNC was identified as a good starting point: It is an already widely used protocol; thus enabling legacy non-multicast clients to connect as well. In contrast to multicast RTP that supports a lot of features that are not needed for a remote desktop application, VNC is simpler, resulting in a less complex system in the end. RTP has features that are needed for proper transmission of audio data, but are of little use for multicasting of simple image data: anti-jitter buffering, reordering of packets and timestamps. Buffering of incoming data in order to compensate jitter is essential for audio data, but not really necessary for image data. Since multicasted image payloads are split up into relatively small packets that are tagged with size and position information, the order in which packets arrive is irrelevant. Timestamps are essential for transmission of audio data, but of little use for image data. VNCs simplicity also is the reason it was chosen over Microsoft's RDP – it is a lot less complex and the core protocol is open and extensively documented. As a reliable code base to build upon the LibVNCServer/LibVNCClient [3] library was chosen because of its good coverage of standard VNC features *and* useful protocol extensions on the server and client side.

III. CollabKit Design

The different non-functional and functional requirements a real-time collaboration system should meet were identified in [8]. This meant designing CollabKit to provide features useful for concurrent multi-user collaboration while ensuring that the system would still offer the same levels of performance with many user connected. Thus, CollabKit design focused on *multi-user support* on one hand and *multicast transmission of remote desktop data* on the other hand.

A. Multi-User Support

1) Concurrent Multi-User Operation : could be achieved by extending the server application x11vnc included within the LibVNCServer distribution and interfacing it with MPX: when a client connects, it gets its own MPX master pointer and keyboard focus which can be operated independently from other MPX master device pairs. It was found that using differently coloured cursors for mouse pointers is imperative in order not to confuse users. To properly route input events to the client's assigned master devices, all functions, variables

 ISBN: 978-1-61208-206-6

and data structures in the VNC server sources that deal with client input had to be extended to be device-aware.

2) Multi-User Graphical Annotations: were made possible by extending the annotation tool Gromit [2] with multi-pointer support. With Gromit, graphical annotations in different colours can be drawn onto an X display. Since the widget toolkit used by Gromit, GTK+, has MPX support in its most recent versions, the remaining task was to change the application itself to be aware of multiple pointers.

3) Client-to-Server Window Sharing : Because VNC is used for distributing the server's screen to connected clients, it was obvious to use the same technology to export client windows to the central desktop. In CollabKit, this is done by using a mode often called »reverse VNC«: The CollabKit server machine runs a VNC viewer that listens for incoming connections. CollabKit clients run a VNC server software that supports sharing single windows instead of the whole screen.

4) Multicast Transmission of Image Data: Since one of the main uses of the system is to transmit rather bulky image data, the underlying network's maximum throughput poses a fundamental constraint. When used on wireless LANs with 54 MBit/s (802.11a/g) or just 11 MBit/s (802.11b) gross data rate, it is apparent that delivering 25 fps of RGB image data to multiple participants will quickly exhaust the network's capacity. Compression can only alleviate the consequences of this problem but not solve it.

Since the image data representing the shared desktop is the same for all connected participants, an obvious approach to avoid the constraints posed by limited network capacity is to use *multicast* data transmission instead of unicast.

The *Remote Framebuffer* protocol used by ordinary VNC only supports unicast data transmission, relying on TCP at the transport layer. It was deemed adequate to let the extension only transmit framebuffer update messages using UDP multicast since other messages defined by the protocol do not consume nearly as much throughput capacity as these. This way the common TCP unicast communication paths can be used for loss-sensitive data while bulky image data is transmitted to clients via UDP multicast, providing significant channel capacity savings when several clients are connected. As suggested by the RFB protocol specification, the multicast VNC extension was realized by introducing a new pseudo-encoding. This way the protocol can be extended in a backward compatible fashion. The full specification of the MulticastVNC protocol extension is presented in [8]. Nevertheless, since the integration of flow control and error handling sets MulticastVNC apart from other multicasting remote desktop solutions, the next Subsection discusses MulticastVNC flow control.

5) MulticastVNC Flow Control: Flow control is the process of managing the rate of data transmission between network nodes to prevent a fast sender from overwhelming a slow receiver. Since UDP does not provide built in flow control like TCP does, an application layer multicast flow control scheme had to be integrated into the *MulticastVNC* protocol extension: MulticastVNC uses a rate-based flow control scheme coupled with a *NACK*-based error handling mechanism: a message retransmission request by a receiver is interpreted as an indication to lower the send rate. Regarding the actual flow control algorithm, MulticastVNC uses a modified form of the send rate adaptation algorithm proposed in [18]. The modifications to the original algorithm are as follows:

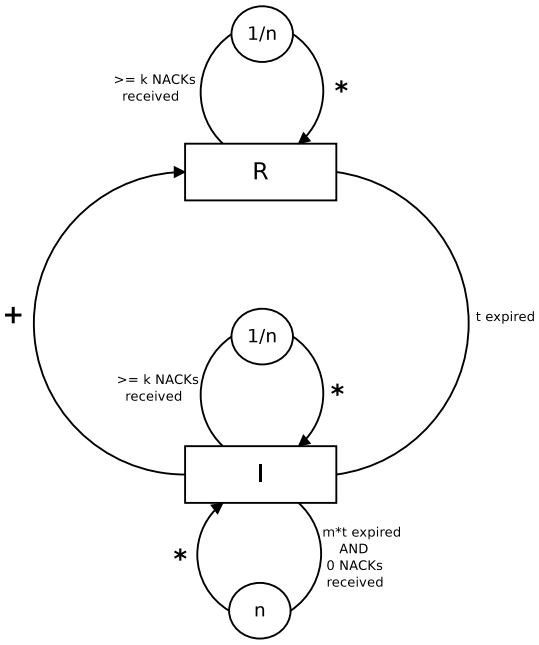


Figure 2: Send rate adaptation algorithm used by MulticastVNC. The send rate R is additively *increased* by a certain increment value I on expiration of time t. R and I are *decreased* on receipt of k or more significant NACKs. The constant parameters m and n are used to balance the algorithm.

For a rate decrease to occur, MulticastVNC requires a burst of k or more significant NACKs. This modification was made because during evaluation it became apparent that the original flow control scheme did not consider networks characterised by relatively high packet loss probability such as WLAN. The NACKs generated for these losses caused the transmission rate to be decreased to a much too low value.

While the original approach used a variable timer value T, the MulticastVNC flow control scheme uses a constant timer value t. This change was made because with the original flow control scheme, the send rate is increased too slowly when its low but too fast when its rather high. Furthermore, there is no reason to let the increment timer depend on the current send rate: While it is true that more NACKs are generated at a higher send rate because more messages are sent, this does not mean that the send rate has to be increased faster because the number of *significant* NACKs as defined in [18] does not change substantially.

An outline of the flow control scheme used by MulticastVNC is depicted in Figure 2.

IV. CollabKit Evaluation

A. Evaluation of Multi-User Functionality

Unlike other systems which only provide turn-taking, CollabKit features *concurrent* multi-user remote control of a shared desktop, concurrent graphical annotations as well as client-to-server window sharing.

 ISBN: 978-1-61208-206-6

1) Concurrent Multi-User View and Control: The modified x11vnc server used in CollabKit provides every participant with their *own* independent mouse cursor and keyboard focus, allowing users to interact with objects on the server's desktop *jointly* and *simultaneously*, as can be seen in Figure 3

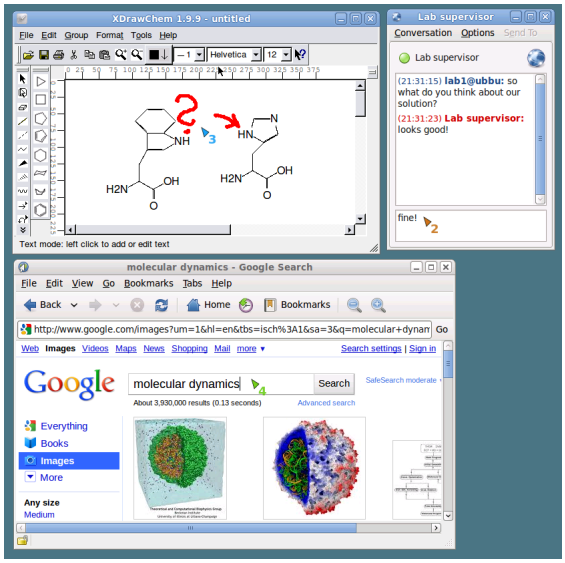


Figure 3: A scientific collaboration use case: three remote participants *concurrently* operating applications on a shared desktop. One is drawing on-screen annotations.

It is important to state though that at the time of writing legacy applications can only be operated flawlessly by *one* user at a time, concurrent interaction on the same desktop is only possible with *different* legacy applications. However, new applications can be designed with multi-device control in mind and existing legacy applications can be modified to be made multi-device aware. In the simplest case, it may be sufficient to just link against a multi-device aware version of the underlying widget toolkit, such as GTK+ 3.0.

2) Multi-User Graphical Annotations: On-screen annotations can be used to either explain something more clearly or to be able to ask more specifically about something on the shared desktop. By using a heavily modified version of the annotation tool Gromit, on-screen annotations can be done *concurrently* by *all* or *some* users: it is possible that only a few clients annotate – others are still able to operate the shared desktop, as can be seen in Figure 3.

3) Client-to-Server Window Sharing: When connecting to a CollabKit server using the CollabKit client, the client-to-server window sharing functionality is easily accessible from within the client application's user interface: after having established a connection to the server, the user may select the »share window« entry out of the »window sharing« menu. The selected window will then appear on the shared desktop. It can be freely dragged around on the shared desktop and also be operated by other participants.

B. Evaluation of the MulticastVNC Extension

The remainder of this paper documents how well MulticastVNC performs compared to traditional unicast VNC. In order to make well-founded statements on unicast versus multicast performance, extensive real-world tests with a total of eight computers were carried out. Up to seven client machines were employed, measuring throughput and latency as well as MulticastVNC NACK and loss ratios.

A single test unit was defined to last exactly 3 minutes, resulting in 180 samples of throughput and latency taken by each participating client instance. In order to put load on the clients, the server machine in all tests constantly sent 640x480 pixels of 32-bit image data with a desired frame rate of 15 frames per second. The VNC encoding used for the majority of tests was Raw encoding. Ultra encoding as the default was considered as well, but ultimately dismissed because with the relatively weak server machine used in the experiments the achievable throughput was found to be CPU-bound instead of being limited by network characteristics and method of data transmission.

1) Throughput Properties: The expected throughput properties of unicast versus multicast data transmission can be formalized as follows: For the *unicast* case, the maximum throughput observable by a client cl can be defined as

$$T_{cl} = \min\left(T_p, \frac{T_{sp}}{N_{sp}}\right) \quad (1)$$

In this metric T_{cl}, the expression T_p describes a concave metric that defines the maximum throughput limited by the characteristics of the network path from server to client: Let $T(n_i, n_j)$ be a metric describing the achievable throughput between two network nodes n_i and n_j and let $p(n_1, n_2, ..., n_m)$ be the path between server node n_1 and client node n_m. Then T_p can be expressed as

$$T_p = \min\left(T(n_1, n_2), T(n_2, n_3), ..., T(n_{m-1}, n_m)\right)$$

Similarly, the expression T_{sp} describes the achievable throughput on the path sp which is the subset of p that the client cl shares with $N_{sp} - 1$ other clients. It can clearly be seen that T_{cl} decreases with an increasing N_{sp}.

However, when using *multicast* data transmission, the maximum client-observable throughput becomes independent of the number of clients that share the same path. The metric then evaluates to a rather simple

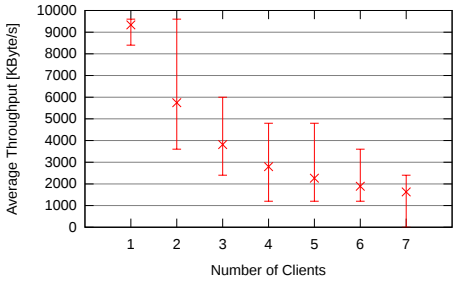
$$T_{cl} = T_p \quad (2)$$

showing that the maximum throughput observable by cl is now independent of the number of other clients it shares the network path to the server with.

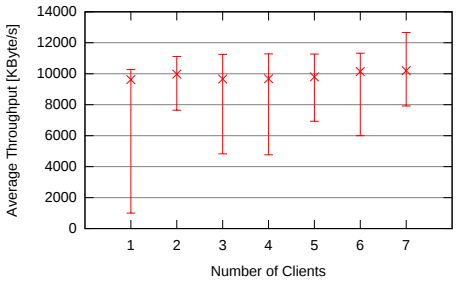
To be able to compare the throughput characteristics of VNC and MulticastVNC on a sound basis, experiments were carried out that measured throughput as seen by clients in different configurations: This included varying the number of connected clients, testing in different network environments (Fast Ethernet LAN and 802.11b WLAN) and changing between traditional unicast VNC and MulticastVNC.

The basic methodology for each test run was to start with one client and increase the number of clients over time. As noted above, this was done every 3 minutes, resulting in 180

 ISBN: 978-1-61208-206-6



(a) VNC.



(b) MulticastVNC.

Figure 4: Average per-client throughput of 1 to 7 clients in a Fast Ethernet LAN using Raw encoding. It can be seen that for unicast data transmission average per-client throughput decreases with an increasing number of connected clients. With MulticastVNC instead, average per-client throughput is independent of the number of connected clients.

samples per client count. This paper presents a subset of the experimental findings.

The first test series measured achieved throughput in a LAN, with all machines being connected through a Gigabit Ethernet switch. Figure 4 shows the results for 1 to 7 connected clients for the VNC and the MulticastVNC case. The graphs show test runs each lasting 21 minutes where an additional client would connect 180 seconds after its predecessor. Values on the y axis are averaged throughput, computed as the arithmetic mean of the samples taken by all active clients during the corresponding 180-second time span. The upper and lower ends of the error bars denote the biggest and smallest values sampled.

With traditional unicast data transmission (Figure 4a), per-client throughput decreases with each new client joining the session. In contrast to the unicast measurements, Figure 4b shows that with MulticastVNC, per-client throughput is not as affected by the number of clients as it is when using traditional VNC. In fact, the graph shows that throughput seen by each client is around 10,000 KByte/s, independent of the number of clients in the session. This matches the theoretical predictions of the metric in equation 2.

2) Latency Properties: Multicast data transmission was also expected to be beneficial for the **latency** of communication taking place, because it cuts down on server answer time. First, for the *unicast* case, the delay or latency observed by a client cl can be defined as

$$L_{cl} = L_p + t_{srv} * (N_{sp} - 1) \quad (3)$$

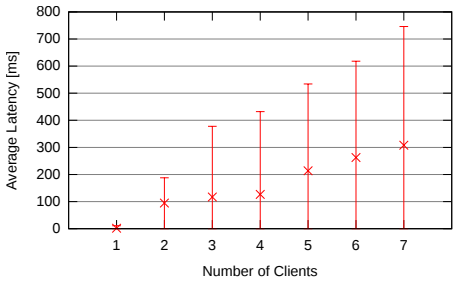
Within L_{cl}, the expression L_p describes an additive metric defining the latency of the client's connection to the server: Let $L(n_i, n_j)$ be a metric that describes the latency between two network nodes n_i and n_j and let $p(n_1, n_2, ..., n_m)$ be the path between server node n_1 and client node n_m. Then L_p can be expressed as

$$L_p = L(n_1, n_2) + L(n_2, n_3) + ... + L(n_{m-1}, n_m)$$

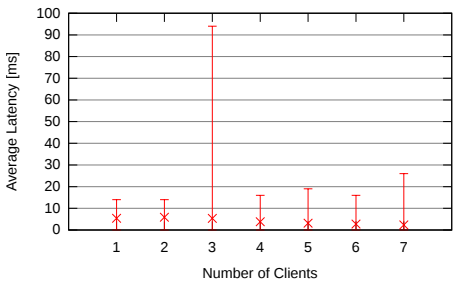
The term t_{srv} in L_{cl} describes the time the server needs to transmit data to a single client. N_{sp} is defined as in the throughput metric above. It can be seen that given a non-zero t_{srv}, L_{cl} increases with an increasing N_{sp}. The higher t_{srv}, the stronger the effect.

The benefit of multicast data transmission is that it eliminates the possible delay a client might encounter while waiting for others to be served: Because data is now sent only once instead of N_{sp} times, the latency observed by cl when using multicast data transmission is described by

$$L_{cl} = L_p \quad (4)$$



(a) VNC.



(b) MulticastVNC.

Figure 5: Average per-client latency of 1 to 7 clients in a Fast Ethernet LAN using Raw encoding. For the unicast case, the graphs show that average per-client latency increases the more clients are connected. For multicast data transmission, test results show a constant average per-client delay.

The latency occurring for different configurations, i.e. a varying number of clients using unicast VNC or MulticastVNC in different network environments, was measured employing the same test methodology as was used in the throughput experiments. The measured latency values reflect the data packet round trip time of the network in use *plus* the time the server takes to reply.

Figure 5 shows the results for 1 to 7 clients in a Fast Ethernet LAN using Raw VNC encoding. Again, the averages are computed as the arithmetic mean of valid samples taken by all active clients, the upper and lower ends of the error

 ISBN: 978-1-61208-206-6

bars denote the biggest and smallest sampled values. As can be seen in Figure 5a, average per-client server answer time increases linearly with more clients to be served, matching the predictions of equation 3. For MulticastVNC, the test results depicted in Figure 5b show a constant average per-client delay unaffected by the number of connected clients as predicted by equation 4.

3) Effectiveness of Multicast Flow Control: MulticastVNC flow control builds upon related work done in [18], but makes two important modifications, described in Section III-A5. This section explains why these changes were necessary and shows that the resulting multicast flow control scheme is in fact working in both wired and wireless network environments.

The corresponding test runs all followed the same basic procedure: A client connected to the server and ran at full receive rate for 30 seconds. After that, it throttled its receive rate to circa 50% and ran in this configuration for another 30 seconds. On expiration of that time span, the client unthrottled again and ran like this for a final 30 second interval. During execution of these high-low-high profiles, the transmission rate of the server's network interface was observed to see how the server adapted its send rate to the respective new situation. Tests were carried out in both a Fast Ethernet LAN and a 802.11b WLAN. The receive rate throttling on the client side was done using the Linux netem emulation layer which allows rate-limiting of incoming traffic by means of a token bucket queuing discipline.

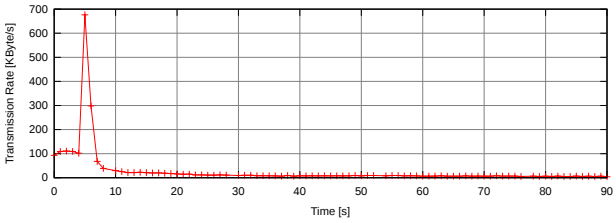
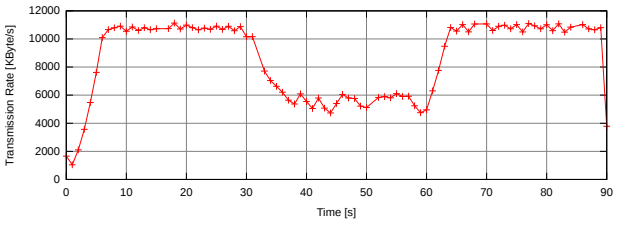


Figure 6: MulticastVNC server transmission rate in a 802.11b WLAN with $m = 10$, $n = 1.2$, $k = 1$, $T = \frac{100{,}000Byte}{R}$. With the unmodified original flow control scheme, the server send rate is almost immediately throttled down to around zero in a WLAN.

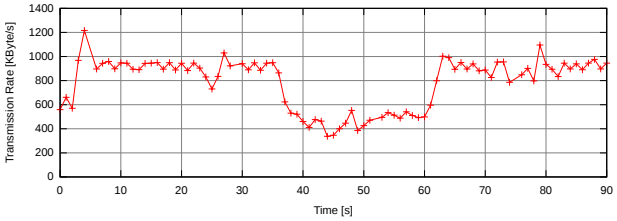
The first incarnation of the MulticastVNC flow control scheme was closely modeled after the original algorithm proposed in [18] and described in detail in Section III-A5. The parameter values of $m = 10$ and $n = 1.2$ used in [18] were adopted for the MulticastVNC flow control. The exact choice of parameters for the variable timer T is left somewhat unclear in the paper though. $T = \frac{100.000Byte}{R}$ was chosen after some testing in a LAN.

While this multicast flow control scheme worked reasonably well in a Fast Ethernet LAN, it failed completely when tested out in a WLAN, as shown by the diagram in Figure 6: The rise in transmission rate at the beginning of the test stems from the client framebuffer initialization which is done via unicast, but then the server send rate is almost immediately throttled down to around zero. Since flow control worked well in a Fast Ethernet LAN before, it was suspected that NACKs generated from occasional packet loss in the WLAN were misinterpreted by the algorithm as a sign for receive buffer overflow at the client side, resulting in the server send rate to be lowered.

This misbehaviour could be fixed by changing the algorithm to only decrease the send rate upon receipt of a *burst* of NACKS. The reason this works better is that the patterns of NACKs generated by occasional packet loss and receive buffer overflow differ: In the former case, NACKs are mostly evenly distributed over time and tightly packed NACK bursts are relatively rare. However, on receive buffer overflow at the client side a relatively large number of packets is dropped at once, resulting in a burst of NACK messages arriving at the server. A burst value of $k = 3$ was found to be adequate for both WLAN and LAN environments.



(a) Fast Ethernet LAN.



(b) 802.11b WLAN.

Figure 7: MulticastVNC server transmission rate in a Fast Ethernet LAN and an 802.11b WLAN with $m = 10$, $n = 1.2$, $k = 3$, $t = 50ms$. With both modifications applied, the revised flow control scheme now works well in both wired and wireless network environments.

While this modification fixed flow control on WLANs, there still was a problem with either too frequent rate increases at high send rates or too slow send rate recovery at low send rates. To solve this, a fixed send rate increment timer value t was introduced instead of the variable timer T that depended on the current send rate R. In fact, there actually is no reason for the send rate increment timer value to depend on the send rate itself: While it is true that more NACKs can be generated at a higher send rate since more packets can be lost, this does not mean that the send rate has to be increased faster – the algorithm distinguishes between significant and meaningless NACKs and marks too high send rates as already decreased. This way, additional NACKs for a certain send rate have no effect. Figure 7 shows that the modified flow control scheme with NACK bursts and fixed send rate increment timer works well in both Fast Ethernet LAN and 802.11b WLAN, respectively. A parameter set of $m = 10$, $n = 1.2$, $k = 3$ and $t = 50ms$ was found to be adequate for both cases. These also are the values used in the throughput and latency experiments presented above.

 ISBN: 978-1-61208-206-6

V. Conclusion and Future Work

By integrating existing technologies and extending them where needed, a computer-supported real-time collaboration system that supports *concurrent multi-user operation* and *scalable multicast transmission of image data* could be created. Source code and documentation are available at the project web site [1].

As opposed to sequential turn-taking where only one user at a time is in control, CollabKit integrates the Multi-Pointer-X extension MPX with the remote desktop technology VNC to give remote participants a per-user cursor and keyboard focus, allowing them to *simultaneously* operate several applications on a shared standard desktop *in parallel*. To make user interaction more expressive, CollabKit furthermore implements simultaneous per-user graphical annotations. For electronic teaching and assistance use cases as well as for professional remote collaboration, a window sharing feature was added that enables users to show local windows to others by exporting them to the remote desktop. A full-featured CollabKit client application is available for Windows and Linux; there is an app for Android and an alpha-stage Mac OS X implementation.

To address scalability with an increasing number of participants and make CollabKit perform well even with a high number of users in a low-throughput network, we extended VNC with support for multicast data transmission. Experiments showed that compared to its unicast counterpart, the MulticastVNC extension performs significantly better when several client computers are connected to the system: While with unicast per-client throughput decreases with additional clients, the use of MulticastVNC makes average per-client throughput largely independent of the number of connected clients. Regarding latency, test results showed that in the unicast case average per-client delay goes up with an increasing number of connected clients while with MulticastVNC it stays at a constant low level. Finally, unlike other examined multicast-enabled remote desktop systems, CollabKit also implements multicast flow control and error handling using a NACK mechanism in order to be able to deploy the system in changing network environments without reconfiguration and to deliver an accurate representation of the shared desktop to its users.

While CollabKit already is a prototype shared view desktop conferencing system, there are further enhancements and feature additions conceivable: Multi-user operation of the shared desktop currently lacks a fine-grained floor control scheme. This could include a concept like window or application ownership where users can take exclusive control of a particular application, which can then be shared with others, passed on or released. Regarding the MulticastVNC extension of the RFB protocol, future work could focus on implementing other VNC encodings than Raw and Ultra or investigate possible encryption schemes for the pixel data sent via multicast. Another interesting topic could be to examine how multicast can be applied to the NACK mechanism so that lost datagrams are not necessarily retransmitted by the server but by other clients that have the requested data available.

References

[1] CollabKit web site. http://wiki.informatik.hu-berlin.de/nomads/index.php/CollabKit – Retrieved: 03, 2012

[2] Gromit web site. http://www.home.unix-ag.org/simon/gromit – Retrieved: 03, 2012

[3] LibVNCServer web site. http://libvncserver.sf.net – Retrieved: 03, 2012

[4] MulticastVNC web site. http://www2.in.tum.de/~ziewer/multicastvnc/ – Retrieved: 03, 2012

[5] PaintChat web site. http://www.paintchat.jp/

[6] SubEthaEdit web site. http://www.codingmonkeys.de/subethaedit/ – Retrieved: 03, 2012

[7] TightProjector web site. http://www.tightvnc.com/projector/ – Retrieved: 03, 2012

[8] Beier, C.: *CollabKit - A Multi-User Multicast Collaboration System based on VNC*, HU-Berlin, Thesis, 04 2011. – http://edoc.hu-berlin.de/docviews/abstract.php?id=39389 – Retrieved: 03, 2012

[9] Boyaci, O. and Schulzrinne, H.: BASS Application Sharing System. In: *Tenth IEEE International Symposium on Multimedia, 2008. ISM 2008*, 2008, pp. 432–439

[10] Coulthart, D. ; Das, S. and Kim, L.: THINCing Together: Extending THINC for Multi-User Collaborative Support. (2009)

[11] Hasan, S. M. ; Lewis, G. J. ; Alexandrov, V. N. ; Dove, M. T. and Tucker, M. G.: Multicast Application Sharing Tool for the Access Grid Toolkit. In: *UK e-Science All Hands Meeting, Nottingham, UK*, 2005

[12] Hutterer, P. and Thomas, B. H.: Groupware support in the windowing system. In: *Proceedings of the eight Australasian conference on User interface-Volume 64*, 2007, pp. 39–46

[13] Masahiro, T. and Lowe, N.: *DrawTop*. ViSLAB School of Information Technologies, University of Sydney, 2006

[14] Ng, C. J. ; Takatsuka, M. ; Smith, S. and Lowe, N.: VNCast web site. http://wiki.vislab.usyd.edu.au/moinwiki/VNCast – Retrieved: 03, 2012

[15] Packard, K. and Gettys, J.: X Window System network performance. In: *FREENIX Track, 2003 Usenix Annual Technical Conference*, 2003

[16] Tse, E. and Greenberg, S.: Rapidly prototyping single display groupware through the SDGToolkit. In: *Proceedings of the fifth conference on Australasian user interface-Volume 28* Australian Computer Society, Inc. (Conf.), 2004, pp. 101–110

[17] Wallace, G. ; Bi, P. ; Li, K. and Anshus, O.: A multi-cursor x window manager supporting control room collaboration. In: *Computer Science Report No. TR-0707-04, Princeton University* (2004)

[18] Yamamoto, M. ; Sawa, Y. ; Shinji, F. and Ikeda, H.: NAK-based flow control scheme for reliable multicast communications. In: *Global Telecommunications Conference, 1998. GLOBECOM 98. The Bridge to Global Integration. IEEE*, 1998, pp. 2611–2616

[19] Ziewer, P. and Seidl, H.: Transparent teleteaching. In: *Proceedings of ASCILITE*. 2002, pp. 749–758

 ISBN: 978-1-61208-206-6