SIMVIKI: A TOOL FOR THE SIMULATION OF SECURE VIDEO COMMUNICATION SYSTEMS

Oge Marques, Phillip Auger and Liam M. Mayron
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL – USA
{omarques, pauger1, lmayron}@fau.edu

ABSTRACT
Recent advances in the field of video communications have increased the need for the simulation of security-related scenarios. This paper presents the philosophy, motivation and design choices that led to the implementation of *SimViKi*, a tool for simulating contemporary video communication systems. SimViKi leverages the best aspects of other popular simulation tools, particularly OMNeT++ and MATLAB. The design and use of the tool are illustrated.

KEY WORDS
Video communications, modeling and simulation tools.

1 Introduction

The field of secure video communications is both new and dynamic and require a new tool for simulating related techniques and scenarios. We introduce SimViKi to address these shortcomings. It combines the advantages of a popular network simulation package (OMNeT++) and a widely used prototyping tool (MATLAB), allowing for a rich, lively, interactive, framework for simulation of multimedia communication systems. Full download and installation instructions are available at http://mlab.fau.edu.

This paper is structured as follows: Section 2 provides background information on secure video communication systems. Section 3 describes the methodology behind the design of the tool and presents its key features. Section 4 explains the use of the tool through a guided tour. Finally, Section 5 presents concluding remarks and directions for future work.

2 Background and context

This section introduces key aspects of contemporary video communication systems, discusses some of the most relevant security aspects associated with them, surveys existing tools for simulation of such systems and points out their limitations. It also elaborates on the motivation for the development of the tool described in Section 3.

2.1 Issues in secure video communications

The increasing availability of digital media and the rise of digital telecommunication technologies has resulted in explosive growth of multimedia communications over the past few years. These developments introduce a number of security risks such as copyright violation, prohibited usage and distribution of digital media, secret communications, and network security. As a result, concerns regarding security, scalability and manageability of existing systems become more acute as current solutions may not satisfy the demands of multimedia communications [16].

2.1.1 Video encryption

Encryption plays a key role ensuring confidentiality in most implementations of security for video communications. However, general purpose encryption algorithms (e.g., AES) are typically not optimal for video encryption because these algorithms do not conform to requirements of video application. In order to overcome this problem a significant number of encryption algorithms specifically designed for digital videos have been developed [14]. These algorithms take into account video-specific aspects such as the level of security and perception, format compliance, bitstream expansion, and error tolerance [14].

2.1.2 Video authentication

Video authentication is a process used to ascertain the trustworthiness of digital video. A video authentication system uses a combination of techniques such as digital signature, digital watermarking, error correction coding, and cryptographic hash functions to ensure the integrity of digital video, and verify that it has not been tampered with [4]. Several video authentication solutions have been proposed during the past few years. While most focus on frame-based video authentication (e.g., [4]), other approaches implement MPEG-4 compliant object-based authentication (e.g., [8]).

2.1.3 Video and image watermarking

Watermarking is the process of embedding data into a multimedia element such as image, audio and video. Watermarks can be embedded into multimedia directly or after the multimedia element has been transformed by a mathematical transform such as DCT, DWT or DFT. Performance issues include robustness against attempts to remove the watermark, capacity (how watermark data can be hidden) and how transparent the watermark is under normal conditions [5]. There is an important difference between encryption and watermarking in enforcing protection against unauthorized use. With encryption-based technology it is possible to protect audiovisual content from eavesdroppers because only the authorized user will have access to the decryption keys. Watermarks do not preclude access to the watermarked content, but are used to protect the content owner against unlawful actions by malicious users such as tampering with the original contents or misrepresenting the contents as their own.

2.1.4 Secure transcoding and secure scalable streaming (SSS)

The combined use of scalable video coding techniques, video streaming and video transcoding has allowed transmission of video contents through computer networks to a wide variety of clients, with different screen size, mobility aspects, and quality requirements. Extending these techniques to a secure scenario presents a number of challenges that are the topic of active research in secure transcoding, SSS, and related efforts [3, 17]. Typical solutions involve exploring selected portions of the encoded video stream in such a way that allows the transcoders' basic functions to be performed without a need for decryption and subsequent encryption, i.e., without allowing for the possibility of the transcoders themselves becoming a security vulnerability.

2.2 Related work

The starting point for our work was an interface for the OMNeT++ simulator (VideoInterface) written by Luca Signorin from the University of Ferrara, Italy [13]. The interface is capable of detecting different video trace file formats and feeding the data into the OMNeT++ simulator. A simple demonstrative application included three main components: the transmitting device, the Internet cloud, and the receiving device. While this framework represented a way to simulate video traffic based on video traces, it does not allow the evaluation of video data available in video files.

Video traces are an alternative to bit streams in which only the number of bits used for the encoding of the individual video frames is provided. As a result, there are no copyright issues, but the ability to infer information about the visual quality of the encoded / received video sequence is lost [12]. Video traces have become very popular among video communication and networking researchers, with several video traces from public video trace libraries [12, 1, 6] currently available. Today, the issues of quality and their correlation to the video frame sizes are of great importance, driven by the need for differential quality of service (QoS), requiring new traces to be made available as well as the existing interfaces to network simulation tools (such as OMNeT++) to be updated and adapted [7] .

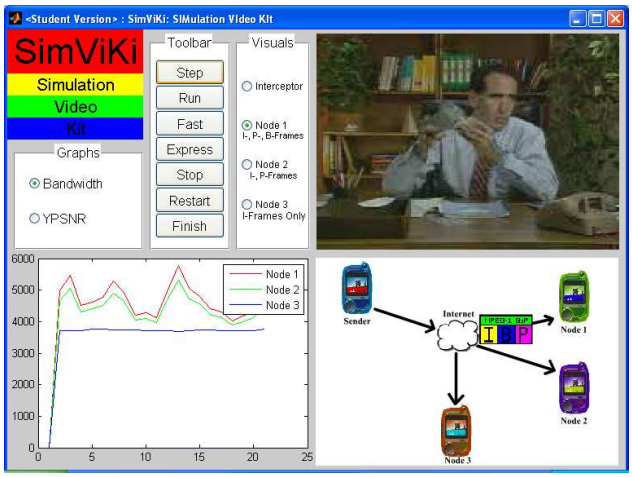

Figure 1. SimViKi displaying a decrypted view.

Another framework, NECTAR [10], allows a person to feed an actual RealNetworks bitstream and manipulate the output with packet-error and bit-error-rate parameters. Our tool extends these frameworks by redesigning and integrating them with MATLAB. The implementation of MATLAB adds quicker prototyping using the MATLAB scripting language, where both previous frameworks relied on C++.

A third related tool is TraceGraph [9], a tracefile analyzer for the popular NS-2 network simulator. It is similar to SimViKi in that it also uses MATLAB, but also suffers from the limitation of only using tracefiles.

3 Design and implementation aspects

SimViKi integrates a diverse range of tools into an effective package. These individual components are detailed in the subsections that follow.

3.1 Languages

SimViKi uses several languages, including: C++, Perl, MATLAB Script, Java, and JavaScript.

OMNeT++ provides a component-based architecture for models. Components (modules) are programmed in C++, and then assembled into larger components and models using a high-level language. OMNeT++ has extensive GUI support, and due to its modular architecture, the simulation kernel (and models) can be embedded easily into

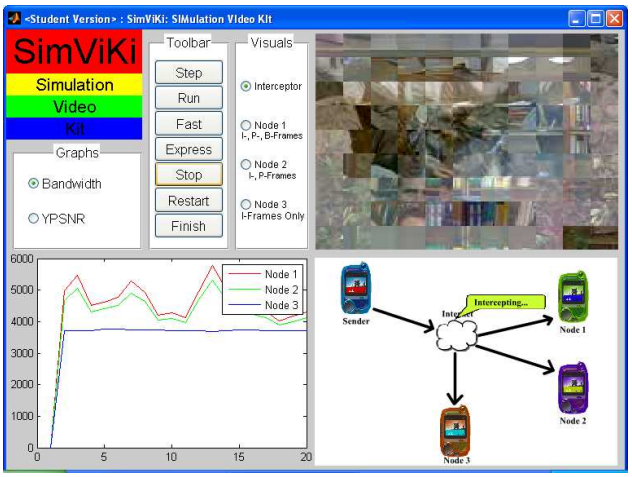

Figure 2. SimViKi displaying an encrypted view.

other applications [15]. In our tool, OMNeT++ provides the network simulation core functionality.

Perl was used because of its batch-like quality of interacting with applications. Much of the MATLAB Engine Interface API and the OMNeT++ API was compiled with PERL scripts. All of the unimportant details, like library linking and compiling directives, are already done for the user so they do not have to bother with these complexities and can delve directly into video processing simulation. The PERL scripts we have created also edit some C++ source files before compilation to solve generation gaps within OMNeT++.

C++ was used because it is required by OMNeT++. It was OMNeT++ (along with the VideoInterface program from ASU) that gave us an entire base of how to go about designing SimViKi. C++ is also the language used to actually run the MATLAB Engine (when the MATLAB Component Runtime isn't used). Since much C++ programming is already done for the user, one doesn't have to have total knowledge to find examples to base code from.

M-Script, or MATLAB Script, is used because of its ability to create useful plots within MATLAB. Also, many image processing functions (with the optional Image Processing Toolbox) can be used for processing video frames. MATLAB itself has much functionality to control everything from outside itself. MATLAB has much interfacing functionality to work with Java, Perl, Fortran, C, C++, etc.

JavaScript has a dual purpose in SimViKi. It was first used to make an ActiveX control for OMNeT++ via Flash as seen in the lower right-hand corner of SimViKi SIMULATOR (the Flash scripting language is called ActionScript). Because Flash takes over much of the graphics functionality, the user will only have to know the guist of JavaScript to make their own network. They can look at the code of the included Flash file to learn how to make an interface without having to learn Flash from scratch. On another note, JavaScript is also used so that the user can output a web page that dynamically shows a video frame-by-frame if a user does not have an appropriate codec.

Because SimViKi sometimes includes other programs, such as FLAVOR, Java programming was required at times. In summary, despite incorporating a number of different tools written in different languages, SimViKi integrates these components so that the typical user does not have to learn any particular language to execute normal tasks.

3.2 Control Scripts

As previously noted, much of SimViKi involves scripts to solve much of the technical non-video-processing aspects of creating simulations. The user will thus usually not edit or interact with these scripts directly.

However, their is one main script that the user should be aware of. It is the omnetpp.ini script file. This file includes all the configuration parameters that a user would want to edit. Though the SimViKi WIZARD is extremely useful for the novice user, there would be no other way for a person to configure the simulation. It just so happens that the SimViKi WIZARD produces this script. With this script, the user can keep a copy to show the settings they used to produce the simulation results for assignments and experiments.

Other than the omnetpp.ini file, the other scripts that the user will want to edit are the MATLAB files. This only happens once the user has an idea of an experiment that the user wants to perform. Because SimViKi is open source, anything is possible.

3.3 Applications

The major programs used to bring SimViKi alive are MATLAB 7.1, MATLAB Component Runtime (MCR), OMNeT++ 3.2p1, and Adobe Flash 9.0.

As said before, the inspiration for SimViKi came from OMNeT++. We originally started out just using trace files, which while boring, were made much more interactive and interesting with the iconic action of OMNeT++. A user may be aware of Ptolemy II and NS2, but OMNeT++ was simple and came with an example framework — VideoInterface — that made it already set up for the type of experimentation we wished to perform.

Eventually, the traces still revealed what was missing the most: visuals. We began to look at graphic libraries to do this in, but they all lacked the "industry-standard" feel. We discoved we could interact with MATLAB, the popular mathematics suite by MathWorks, via an Engine API. Later, we discoved that we could actually compile a stand-alone application using the MCR that is part of the MATLAB Compiler. This is when SimViKi went from a bunch of loose figures to one main windowed easy-to-use application.

Because we wanted to make SimViKi an all-in-one application (i.e. no broken windows), we didn't want to

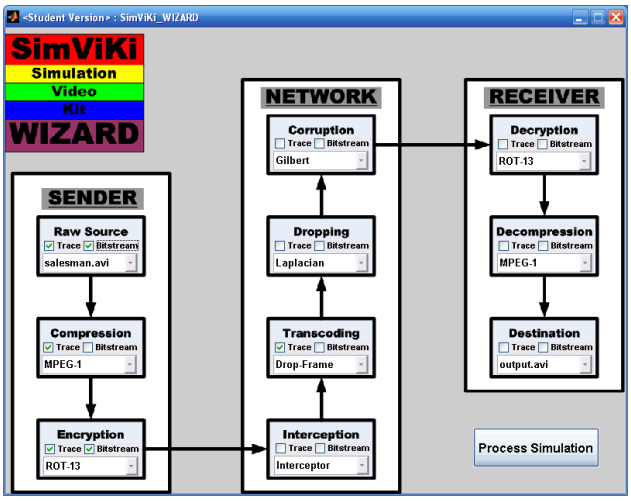

Figure 3. The SimViKi wizard.

use OMNeT++ Tkenv. As OMNeT++ didn't have an ActiveX control, we had to come up with our own which was not too complicated. Not many network simulators come as an ActiveX control, but instead come with Tcl/Tk-type interfaces. We achieved our goal with the Shockwave Flash ActiveX control.

In the end, MATLAB came with its own installation of Perl and Java, so one didn't have to download them separately from the web. If the MCR is used, the user would have to download Java from the Internet, as the GUI's that SimViKi contain require the Java Runtime Enviroment. One actually doesn't program the GUI's in Java, but in MATLAB Script which translates itself into Java code. GUIDE was not used in the production of the GUIs because it causes the figure to be dramatically affected in appearance from one monitor resolution to another. Therefore, the GUIs were programmed manually.

4 Guided tour

This section takes the reader through a step-by-step walkthrough of running a SimViKi simulation. This walkthrough represents how any simulation would be performed.

First, the user will download and install the executable. It will automatically setup all required components, including Adobe Flash 9, MATLAB Component Runtime, and Java Runtime Environment 5. Appropriate icons appear so the user has easy access to the four major programs of SimViKi: SimViKi WIZARD, SimViKi PROCESSOR, SimViKi SIMULATOR, and the SimViKi ANALYZER.

4.1 Simulation design

The user begins by executing the "SimViKi WIZARD" application. By default, all options are pre-set to the Secure Scalable Streaming demo. Had the user wished to change any of the options, they would do so via the "trace" and "bitstream" checkboxes located in every step of the network. If the user wishes to change the settings of each step, like the codec used for compression or the cipher used for encryption, they would click on the respective drop-down menu located in each node. The checkboxes merely record the actions set forth by the drop-down menu. For example, in the "encryption" node, if the user chooses the selective cipher and clicks the "trace" checkbox, a resulting trace file will be made that keeps statistics on the encrypted video bitstream. If the user checks the "bitstream" checkbox, they will be able to view (if format compliant) the actual encrypted video, which of course should be degraded in some fashion. Bitstreams will always take up much more space on a users hard-drive than traces, as video is actually saved opposed to a text file.

The SimViKi WIZARD produces an omnetpp.ini file. The user can also edit a simulation from this file and later on, if the user wishes to perform a very complex simulation with lots of options, they can add their options to the onetpp.ini file instead of using the SimViKi WIZARD. This concept of beginner-GUI/advanced-Script is a popular concept that research-and-development companies like to produce to help newcomers get used to their research tools while also being productive at the same time.

Once all options are set in the SimViKi WIZARD (or omnetpp.ini file), the user will press the "Process Simulation" button. This will close the SimViKi WIZARD and run the SimViKi PROCESSOR according to the configured options. A dialog gives a time estimate to completion. Nothing graphical happens at this point. The time taken to complete the simulation usually depends upon the length of the video. The video for the demo should take about 15 minutes to process. Obviously, had a trace file been fed to the simulator instead of a bitstream, the length would significantly be reduced. The SimViKi PROCESSOR is where all the heavy back-end processing takes place so as to produce results that the user can review. As a user becomes more familiar with the SimViKi framework, they may choose to actually delve into the source code of the SimViKi PROCESSOR and hand-edit the scripts to produce unique results for research. The code has been put in to a structure of documented and commented code for this purpose.

4.2 Simulation execution

Once the SimViKi PROCESSOR has completed, the user will be advised that they can look at the results in the SimViKi ANALYZER (Figure ?efanalyzer) or the SimViKi SIMULATOR. These two programs are also available from the Start Menu if the user chooses to close this dialog.

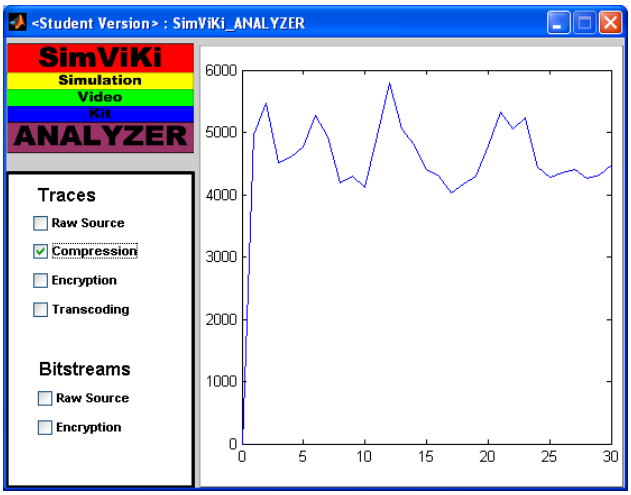

Figure 4. The SimViKi analyzer.

The SimViKi SIMULATOR will give the user the option to review the entire simulation in a much faster timeframe (i.e. you can now watch a 12 second video in 12 seconds rather than 15 minutes, along with its graph and network activity). The SimViKi SIMULATOR has a graph for plotting statistics from traces, a monitor for displaying a view of the bitstream, and a network that shows the location and state of the traveling network packets. All of these run simultaneously so the user can get a "feel" for how every step works and what results. The user can proceed through the simulation via the "step", "run", "fast", and "express" buttons in the toolbar. The user can switch between different views or different graphs that they chose to process from within the SimViKi WIZARD. Because of this all-inclusive-concept of this simulation environment, SimViKi SIMULATOR is an excellent tool for students studying video processing and network transmission concepts. Tutorials are included specifically for exploiting this purpose. The SimViKi SIMULATOR is also a good tool for a researcher trying to "debug" problems of video processing and transmission, as they can "step" or "run" through the simulation to find trouble spots.

4.3 Analysis of results

The SimViKi ANALYZER will give the user the option to quickly look at the end results of the simulation. This is for the user who is solely interested in the final state of a plot of a graph rather than watching the graph being created dynamically in the SimViKi SIMULATOR. The SimViKi ANALYZER is simple enough in that all it contains is a left-hand side column of checkboxes and a right-hand column that holds the graph or view. If the user wishes to see the view, an embedded MPlay media player comes up and plays the movie. If the user wishes to see the graph, a plot will instead come up. The user also has the option of checking multiple graphs to overlay the plot lines for comparison.

The user does not need to use the SimViKi ANALYZER to retrieve the final results. The results of each trace are saved as individual graphs within simulation directory. If the user wishes to save results of overlapping graphs, though, they must use the SimViKi ANALYZER. The SimViKi ANALYZER is also capable of saving the graphs in formats other than the default. This will allow the user to share their graphs with users who do not have the MATLAB environment.

When the simulation is complete an analysis interface (Figure 1), the heart of SimViKi, appears. From here, the user can review visuals and graphs of the just-processed simulation in real-time, or step-by-step, allowing any user to tailor their experience for studying and comprehending the simulation. This control is performed via the toolbar. The user can choose to see the graph of the YPSNR or bandwidth of the transmitted packet-frames of the video (lower left-hand corner). Simultaneously, the user see the video that corresponds to the respective YPSNR or bandwidth (upper right-hand corner), as well as watch the current location of that same packet-frame within the overall network (lower right-hand corner). Note that the network, an ActiveX control implementation of Tkenv, is based off of VideoInterface [13]. If the user wishes to see a larger picture of the graph, network, or view, they can double-click on the respective area to bring up a larger secondary window.

5 Concluding remarks

This paper discussed contemporary issues in secure multimedia (particularly video) communications influencing the design of a new tool. Ongoing and future work include expanding the tool's functionality to a broader number of external (encoding, encryption, watermarking) algorithms and standards. We are also developing secure multimedia communications experiments for both teaching and research. We believe our framework will be an enticing tool for educational purposes in digital image, video processing, and network security classes.

We presented SimViKi, a new framework for simulation of video communication systems that builds upon the OMNeT++ [15] platform and the video traces research developed at Arizona State University (ASU) [1], integrates it with MATLAB [2], allowing the use of actual video files (in addition to pre-existing video traces [12]), different encoding and security-related (e.g., encryption) algorithms, therefore improving and extending the functionality of OMNeT++ and making it a unique tool for video communications teaching and research purposes.

The developed tool overcomes some of the limitations of conventional video traces. Instead of enhancing the contents of the video trace (as in [11]), we expand the scope of the simulation environment to allow use of actual video sequences (if their size and copyright issues allow), easy inclusion and configuration of codecs and security-related

processing blocks (e.g., encryption/decryption algorithms), and visualization of results – using MATLAB's image processing and viewing capabilities – that demonstrate what the video sequence would actually look like at the receiving end or in any meaningful point within the network.

References

[1] Arizona State University Video Traces Research Group. http://trace.eas.asu.edu/.

[2] MATLAB. http://www.mathworks.com/products/matlab/.

[3] J. Apostolopoulos. Secure media streaming & secure adaptation for non-scalable video. *IEEE ICIP*, October 2004.

[4] P. K. Atrey, W.-Q. Yan, and M. S. Kankanhalli. A scalable signature scheme for video authentication. *Journal of Multimedia Tools and Applications*, 2006.

[5] A. M. Eskicioglu and E. J. Delp. An overview of multimedia content protection in consumer electronics devices. *Signal Processing: Image Communication 16*, 2001.

[6] F. H. Fitzek and M. Resslein. MPEG-4 and H.263 video traces for network performance evaluation. In *TKN - Telecommunication Networks Group.*

[7] F. H. P. Fitzek, P. Seeling, and M. Reisslein. Using network simulators with video traces, May 08 2003.

[8] D. He, Q. Sun, and Q. Tian. A secure and robust object-based video authentication system. *EURASIP Journal on Applied Signal Processing*, 2004.

[9] Jaroslaw Malek. Trace Graph - Network Simulator NS-2 trace files analyser. http://www.tracegraph.com/.

[10] Karl Martin, Cindy Guo. NECTAR - Multimedia Laboratory. http://www.dsp.toronto.edu/nectar/projects/ssms/.

[11] O. A. Lotfallah, M. Reisslein, and S. Panchanathan. A framework for advanced video traces: Evaluating visual quality for video transmission over lossy networks. *EURASIP Journal on Applied Signal Processing*, 2006.

[12] P. Seeling, M. Reisslein, and B. Kulapala. Network Performance Evaluation Using Frame Size and Quality Traces of Single-Layer and Two-Layer Video: A Tutorial. Technical report, Arizona State University, 2004.

[13] Signorin Luca. Video Interface. http://trace.eas.asu.edu/tools/.

[14] D. Socek, H. Kalva, S. S. Magliveras, O. Marques, D. Culibrk, and B. Furht. A permutation-based correlation-preserving encryption method for digital videos. *ICIAR 2006 - International Conference on Image Analysis and Recognition*, September 2006.

[15] A. Varga. The OMNeT++ discrete event simulation system. *Proceedings of the European Simulation Multiconference (ESM'2001)*, June 2001.

[16] S. Voloshynovskiy, O. Koval, F. Deguillaume, and T. Pun. Multimedia security: Open problems and solutions. *In Proceedings of NATO - Advanced Study Institute: Security Through Science Program, Nork, Yerevan, Armenia*, October 2005.

[17] S. Wee and J. Apostolopoulos. Secure scalable streaming enabling transcoding without decryption. *IEEE ICIP*, October 2001.