DESIGN AND VERIFICATION OF COMMUNICATION PROTOCOLS FOR PEER-TO-PEER MULTIMEDIA SYSTEMS

Senem Velipasalar, Chang-Hong Lin, Jason Schlessman, Wayne Wolf

Princeton University, Dept. of Electrical Engineering, Princeton, NJ 08544
{svelipas, chlin, jschless, wolf}@princeton.edu

ABSTRACT

This paper addresses issues pertaining to the necessity of utilizing formal verification methods in the design of protocols for peer-to-peer multimedia systems. These systems require sophisticated communication protocols, and these protocols require verification. We discuss two sample protocols designed for two distinct peer-to-peer computer vision applications, namely multi-object multi-camera tracking and distributed gesture recognition. We present simulation and verification results for these protocols, obtained by using the *SPIN* verification tool, and discuss the importance of verifying the protocols used in peer-to-peer multimedia systems.

1. INTRODUCTION

Distributed multimedia systems are being used in an increasing number of application areas such as multi-camera systems for surveillance and *smart* rooms in addition to multimodal configurations. As networks of distributed processors become economically viable and increasingly useful for multimedia applications, some concerns are made manifest. A first concern is the topology of the distributed processors, that is, with whom a given processor can communicate. Server-based systems have a bandwidth scaling problem, since the central server can quickly become overloaded by the increase in requests commensurate with an increased number of nodes. This necessitates the use of peer-to-peer systems, as they provide an increased number of communication channels and thus scalability. A second concern is the manner in which the nodes will communicate, that is, what data will be sent in what fashion, and what events trigger the data transfer. As a result, peer-to-peer systems require efficient communication protocols. Also, the lack of a central server within peer-to-peer systems mandates a sophisticated communication protocol. These protocols find use in real-time systems, which tend to have stringent requirements for proper system functionality. Hence, the protocol design for these systems necessitates transcending typical qualitative analysis using simulation; and instead, requires verification. The protocol must be checked to ensure it does not cause unacceptable issues such as deadlocks and process starvation, and has correctness properties such as the system eventually reaching specified operating states. Formal verification methods of protocols can be derived from treating the individual nodes of a system as finite state automata. These then emulate communication with each other through the abstraction of a channel. At this point, rules for communication dependant upon individual automata states are developed. With the system modeled, a number of formal techniques exist for verifying the models.

Verification of communication protocols has been pursued previously, particularly for security and cryptographic systems. Karlof et al. [1] analyzed the security properties of two cryptographic protocols and discovered several potential weaknesses in voting systems. Evans and Schneider [2] introduced time into communication sequential processes and verified time-dependent authentication properties of security protocols. Vanackère [3] modeled cryptographic protocols as a finite number of processes interacting with a hostile environment and proposed a protocol analyzer TRUST for verification. Finally, a burgeoning body of work exists pertaining to the formal verification of networked multimedia systems. Bowman et al. [4] described multimedia stream as a timed automata, and verified the satisfaction of quality of service *QoS* properties including throughput and end-to-end latency. Sun et al. [5] proposed a testing method for verifying QoS functions in distributed multimedia systems where media streams are modeled as a set of timed automata.

In this paper, we introduce and exhaustively verify two communication protocols designed for different peer-to-peer computer vision applications. In the following section we briefly discuss *SPIN*, the tool used for our work.

2. THE *SPIN* VERIFICATION TOOL

SPIN is a powerful software tool used for the formal verification of distributed software systems. It can analyze the logical consistency of concurrent systems, specifically of data communication protocols. A system is described in a modeling language called *Promela* (Process Meta Language). Communication via message channels can be defined to be synchronous or asynchronous. Given a *Promela* model, *SPIN* can either perform random simulations of the system's exe-

This work has been funded by NSF ITR grant 325119, NSF grant CCR-0329810, and ARO grant #W911NF-05-1-0480.

cution or it can perform exhaustive verification of correctness properties [6]. It goes through all possible system states, enabling designers to discover potential flaws while developing protocols. We used this tool to analyze and verify two different communication protocols introduced in Sections 3 and 4.

3. SCCS: A SCALABLE CLUSTERED CAMERA SYSTEM

SCCS is a peer-to-peer multi-camera system for multiple object tracking [7]. Different CPUs are used within this system to process inputs from different cameras. Instead of transferring control of tracking jobs from one camera to another, each camera in the system performs its own tracking and keeps local tracks for each target object, providing fault tolerance.

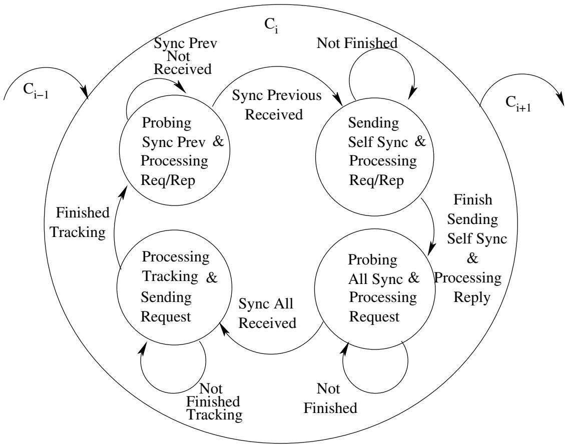
In this system, a camera will need information from the other cameras when: a) a new object appears in its field of view, or b) a tracker cannot be matched to its target object. These events are referred to as *new_label* and *lost_label* events, respectively. If one of these events occurs within a camera's field of view, the processor processing that camera needs to communicate with the other processors. As communication can be expensive, this requires the design of an efficient communication protocol, which can also alleviate the potential problems caused by communication delays. Furthermore, as a system such as this might be utilized in a real-time critical environment, it mandates a communication protocol which provides robustness and guarantees of the prevention of issues such as deadlocks and process starvation.

3.1. SCCS Communication Protocol

The SCCS protocol uses non-blocking send and receive primitives for message communication. Non-blocking messages are used since for each camera it is difficult to predict when and how many messages will be received from other cameras. Also, they provide a higher level of efficiency when compared to blocking communications.

This protocol also addresses the issue of synchronization, which is important for multi-camera systems. As the processors will have different amounts of processing to do and may also have different processing speeds, it is possible that one processor can be ahead of/behind the others during execution. To ensure the transfer of coherent vision data between cameras, the processors must be synchronized. To achieve this, our protocol provides *synchronization points*, where all nodes are required to wait until every node has reached the same point. These points are determined based on a synchronization rate, *synch_rate*, where *synchronization points* occur every *synch_rate* frames. Between two such points, each camera focuses on performing its local tracking tasks, saving the requests that it will make at the next synchronization point.

Fig. 1 shows a diagram of an extended version of the synchronization mechanism introduced in [7]. The extensions applied pertain to real-time concerns. This figure illustrates the camera states at a synchronization point. In the first state, the camera finishes its local tracking, and the processor sends out all of its saved requests. Then, the camera enters the second state and begins to probe to see if a *done* message has been received from the previous camera. If not, this camera probes for requests from the other cameras and processes them while waiting for the replies to its own requests. When the *done* message is received from the previous camera the camera enters the third state. When all of its own requests are fulfilled, it sends out a *done* message to the next camera. In the fourth state, each camera node still processes requests from other cameras, and keeps probing for the overall *done* message. Once it is received, a new cycle starts and the node returns back to the first state. In this extended version, *done* messages are sent by using a *ring* type of message routing.



C_i
Sync Prev Not Received
Not Finished
C_{i-1}
C_{i+1}
Sync Previous Received
Probing Sync Prev & Processing Req/Rep
Sending Self Sync & Processing Req/Rep
Finish Sending Self Sync & Processing Reply
Finished Tracking
Processing Tracking & Sending Request
Sync All Received
Probing All Sync & Processing Request
Not Finished Tracking
Not Finished

Fig. 1. Camera States at Synchronization Point

3.2. SPIN Simulation and Verification Results

To analyze and verify our communication protocol, we described our system by using *Promela*. We modeled three different scenarios: (a) a 2-processor system with full communication, (b) a 3-processor system, where the first processor can communicate with the second and third, second processor can only communicate with the first, and third one only replies to incoming requests, and (c) a 3-processor system with full communication, where full communication means every processor in the system can send requests and replies to each other. The reason we tried scenario (b) is clarified below.

First, we performed random simulations. With random simulation, every run may produce a different type of execution. In all the simulations of all three scenarios, all the processors of the model terminated properly. However, each random simulation goes through one possible set of states. Thus, an exhaustive search of the state space is needed to guarantee that the protocol is error-free. We performed exhaustive verification of the three different scenarios with different synchronization rates. We also inserted an assertion into the model to ensure a processor starts a new synchronization interval *only if* every processor in the system has sent a *done* message at the synchronization point. All three cases have been verified with no errors. Table 1 shows the results obtained, where the *synch_rate* is 1, and there are 4 synchronization points. (a), (b) and (c) correspond to the scenarios described above. As can be seen in the table, when 3 processors are used with full communication, the number of states becomes very high compared to other scenarios, thus the search requires more memory. Scenario (b) was modeled so that we can compare scenario (c) to (b), and see the increase in the number of states and memory requirement. The total memory usage in the table is the "total actual memory usage" output of the *SPIN* verification. This is the amount after the compression performed by *SPIN*, and includes the memory used for a hash table of states. Table 2 shows another set of results for a *synch_rate* of 2.

	No. of States	State-vector Size (bytes)	Total Memory Usage (MB)	Depth
2 Proc. (a)	12259	496	3.017	159
3 Proc. (b)	19369	1252	3.217	182
3 Proc. (c)	5846880	1252	146.417	202

Table 1. Comparison of exhaustive verification outputs for different number of processors and different communication patterns. The *synch_rate* is 1 and there are 4 synchronization points.

	No. of States	State-vector Size (bytes)	Total Memory Usage (MB)	Depth
2 Proc. (a)	20305	496	3.217	200
3 Proc. (b)	196767	1252	7.817	600
3 Proc. (c)	15016100	1252	418.417	657

Table 2. Comparison of exhaustive verification outputs for different number of processors and different communication patterns. The *synch_rate* is 2 and there is 1 synchronization point.

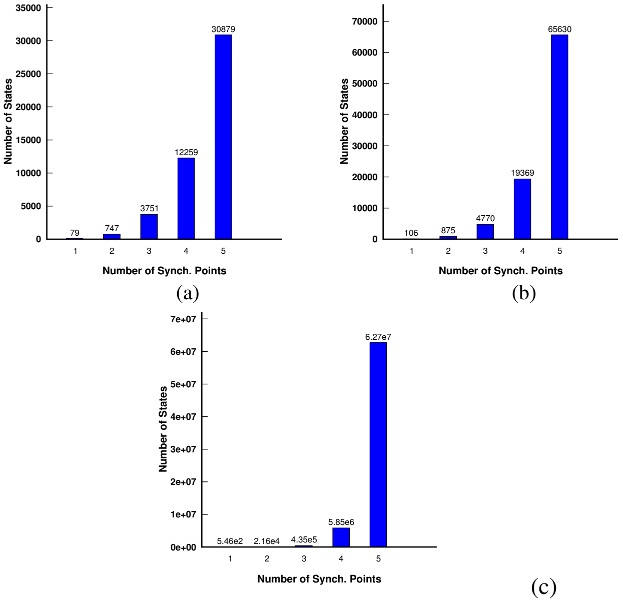
Fig. 2 shows the number of states reached with scenarios (a), (b) and (c), and different number of synchronization points. For the 3-processor and full communication scenario, the number of states increases very fast with increasing number of communication points. Since the memory requirement increases with the number of states, the scenario (c) requires the most amount of memory for verification.

In addition, when the *synch_rate* is increased, the number of states increases for the same number of synchronization points, as the requests of the local trackers are saved until the next synchronization point, and then sent out.

These results show that verification of complicated protocols is not a straightforward task. Also, careful modeling of the large systems, having many possible states, is very important for exhaustive verification. The models of our three scenarios have been verified exhaustively without any errors.

4. DISTRIBUTED GESTURE RECOGNITION

We also analyzed and verified a distributed gesture recognition system as an example. Details of our gesture recognition system considered were discussed previously [8]. The software of the system can be broken into several stages, as illustrated in Fig. 3. For each camera, the algorithm consists of two parts: intra and inter-frame processing. The intra-frame processing performs human body detection and extracts abstract graph parameters. It starts with region segmentation, which performs background elimination and skin-tone detection to identify foreground objects and skin regions. The system then extracts contours on the boundaries of detected regions, and uses abstract ellipse parameters to model various regions. The ellipses are then mapped to different human body parts. The inter-frame processing uses Hidden Markov Models to determine movements of the body parts, and uses a distance classifier to detect specific gestures.



(a)

(b)

(c)

Fig. 2. Number of states reached during the verification of (a) 2- and (b), (c) 3-processor scenarios.

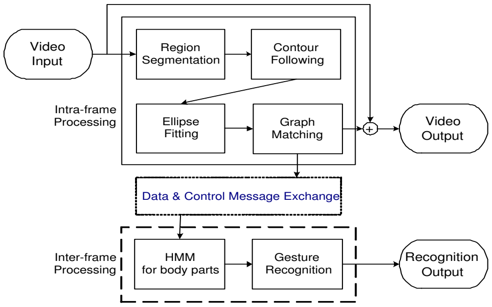


Fig. 3. Software architecture of gesture recognition algorithm.

With multiple cameras, camera nodes have to exchange data and control information about video streams with their neighbors. This communication helps perform recognition as a whole and eliminates redundant operations. The entry points of each processing stage are candidates for data exchanging, and as shown in Fig. 3, we assume communication occurs after intra-frame processing. Other than the recognized body parts, camera nodes also exchange control messages to determine which node should perform gesture recognition for a particular individual. Here, we assume that camera orientation remains stable, and the control messages consist of timestamps for synchronization, and control tokens to determine the ownership of detected people. As a result, only the owner camera performs inter-frame processing.

We propose a communication protocol within the application layer useful for numerous structures. For example, TCP/IP can be used as a networking middleware, with a datagram of our protocol serving as a payload of packets. Both data and control messages can be integrated within a single datagram, and sent to other cameras simultaneously. When a node receives packages, the data has to be combined with its own captured data set, and uses the received control signals to determine subsequent procedures. The camera nodes would first find matching body parts within received and captured data, with the match containing more pixels in the *head* portion considered as the dominant camera. Camera nodes then use received control messages and dominant camera information to determine the ownership of detected people.

4.1. Control Authentication Protocol

Although our system can handle multiple simultaneous objects, for control authentication each target should determine their ownership independently with awareness of other objects. We propose target-centric modeling to determine ownership of each person inside the scene. For each detected person, our system would spawn a separated service to recognize the gesture of the person. The services take received messages to identify the current system state for each target, and perform operations accordingly. The cameras seeing the target person are identified first, and the camera to perform gesture recognition is then chosen based on proposed protocol.

As shown in Fig. 3, control and data message exchanges occur after intra-frame processing. After graph matching, camera nodes would wait and check if there is a pending data packet from its neighbors. If no message is received, or the message has an out of date timestamp, the camera node then performs gesture recognition on the captured body parts. If the target is not owned by the current node, the node would claim temporary ownership, in the case of a delayed or lost message from its neighbors. When a node receives messages from its neighbors, it first checks the timestamp of the packet, and updates its own clock if a faster timestamp is received. The body parts in the received messages are matched to the captured data set, and the dominant cameras for each detected person are then determined. There is an owner token for each target, and the camera owning the token will perform gesture recognition for that person. The ownership changes when the current owner no longer dominates the target for a certain period of time. For a camera without owner tokens, if the camera dominates a target, it upgrades its ownership, otherwise it does nothing. The protocol is illustrated in Fig. 4.

To ensure correctness and sufficiency of our protocol, we again use *SPIN*. We claim that as long as a person is detected, there will be at least one camera which recognizes its gesture, and only one camera will be performing gesture recognition when all the messages are received correctly within a frame. The *SPIN* exhaustive verification proves our claim, and shows no redundant state and undesired loops in our system.

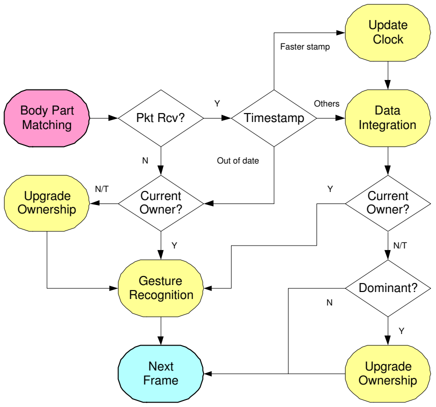


Fig. 4. Protocol for distributed gesture recognition.

5. CONCLUSIONS

Peer-to-peer systems require sophisticated communication protocols that can handle processing and communication delays, and system failure. These protocols need to be evaluated and verified against potential deadlocks, and their correctness properties need to be checked. We introduced two communication protocols designed for different peer-to-peer computer vision systems, and then analyzed and verified them by using the *SPIN* verification tool. As large systems have many states with larger-sized state vectors, their verification is not a straightforward task, and requires careful modeling of the system. We verified our protocols exhaustively without any errors and redundancies. We also showed the verification outputs for three different scenarios of the distributed multi-camera multi-object tracking protocol. These results illustrate the increase in the number of states for different number of processors and different communication scenarios.

6. REFERENCES

[1] C. Karlof, N. Sastry and D. Wagner,“Cryptographic Voting Protocols: A Systems Perspective,” *USENIX Security Symp.*, 2005.

[2] N. Evans and S. Schneider, “Analysing Time Dependent Security Properties in CSP Using PVS,” *ESORICS*, 2000.

[3] V. Vanackère, “The TRUST Protocol Analyser, automatic and efficient verification of cryptographic protocols,” *Verification Workshop*, 2002.

[4] H. Bowman, G. Faconti and M. Massink, “Specification and Verification of Media Constraints Using UPPAAL,” *Eurographics Workshop, DSV-IS*, 1998.

[5] T. Sun, K. Yasumoto, M. Mori and T. Higashino, “QoS Functional Testing for Multimedia Systems,” *IFIP FORTE*, 2003.

[6] G. J. Holzmann, *The Spin Model Checker - Primer and Reference Manual*, Boston: Addison Wesley, 2004.

[7] S. Velipasalar, J. Schlessman, C-Y. Chen, W. Wolf, and J. P. Singh, “SCCS: A Scalable Clustered Camera System for Multiple Object Tracking Communicating via Message Passing Interface,” *IEEE ICME*, 2006.

[8] C.H. Lin, T. Lv, W. Wolf, I.B. Ozer, “A peer-to-peer architecture for distributed real-time gesture recognition,” *ICME*, 2004.